Int. J. Appl. Math. Comput. Sci., 2021, Vol. 31, No. 4, 697–712
DOI: 10.34768/amcs-2021-0048

AN ANN–BASED SCALABLE HASHING ALGORITHM FOR COMPUTATIONAL CLOUDS WITH SCHEDULERS

JACEK TCHÓRZEWSKI [a,b,*], AGNIESZKA JAKÓBIK [a], MAURO IACONO [c]

[a]Faculty of Computer Science and Telecommunications
Cracow University of Technology
ul. Warszawska 24, 31-155 Cracow, Poland
e-mail: {jacek.tchorzewski,ajakobik}@pk.edu.pl

[b]Faculty of Computer Science, Electronics and Telecommunications
AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Cracow, Poland

[c]Department of Mathematics and Physics
University of Campania "Luigi Vanvitelli"
viale Lincoln 5, 81100 Caserta, Italy
e-mail: mauro.iacono@unicampania.it

The significant benefits of cloud computing (CC) resulted in an explosion of their usage in the last several years. From the security perspective, CC systems have to offer solutions that fulfil international standards and regulations. In this paper, we propose a model for a hash function having a scalable output. The model is based on an artificial neural network trained to mimic the chaotic behaviour of the Mackey–Glass time series. This hashing method can be used for data integrity checking and digital signature generation. It enables constructing cryptographic services according to the user requirements and time constraints due to scalable output. Extensive simulation experiments are conduced to prove its cryptographic strength, including three tests: a bit prediction test, a series test, and a Hamming distance test. Additionally, flexible hashing function performance tests are run using the CloudSim simulator mimicking a cloud with a global scheduler to investigate the possibility of idle time consumption of virtual machines that may be spent on the scalable hashing protocol. The results obtained show that the proposed hashing method can be used for building light cryptographic protocols. It also enables incorporating the integrity checking algorithm that lowers the idle time of virtual machines during batch task processing.

Keywords: hashing algorithm, artificial neural network, scalable cryptography algorithm, computational cloud, task scheduler.

1. Introduction

Cloud computing (CC) systems data should be collected, processed, and stored according to international standards and regulations like NIST (Bowen *et al.*, 2006), Cloud Security Alliance (CSA, 2011), ENISA (ENISA, 2009) or ISO (ISO/IEC, 2015). Lately, CC systems have also been consuming the data from smartphones, tablets, IoT and processing them in data centres or at the edge of the cloud, which creates many challenges like systems performance or scaling (Podolskiy *et al.*, 2019). Nevertheless, the security of all processes is still mainly the responsibility of cloud providers. Tasks that CC processes may have different security demands (SDs) considering ownership of data or data sensitivity. Assuring the proper SD of tasks is a crucial part of the workload management process (Grzonka *et al.*, 2018; Jakóbik, 2016). Personalization of cloud services enables users to change the features of virtual machines (VMs) that provide the different security trust levels (TLs). A critical process is mapping tasks to suitable computing units that offer the proper TL to fulfil the SD required by tasks.

The main idea of this paper is to present a scalable security service that is combined with a scheduler inside

*Corresponding author

the cloud. We have started considering this topic after analyzing various CC services and resources that may be scaled on demand. Users may set the memory, disk space, computing power of their VMs instances and change them according to their needs. Our main goal is to propose a cryptography service that may also be scalable on demand. Assuring security in modern systems requires designing algorithms and tools defending user privacy, data and communication security. Our service is offering a hashing procedure. Such an algorithm may be used for data integrity control by checking unauthorized data modification, for verification or as a part of digital signatures.

The ability of changing the hash length results in increasing or decreasing the time spent on security operations. Having control over this stage of computation may support processing the task with deadlines or processing large batches of tasks when the system is waiting for the last task in the pool to be processed. Additionally, using the scalable hashing method, we are erasing redundant and unnecessary security operations. This method enables using the same algorithm with different parameters for a small tablet and large data centre. A hash function is a popular tool for checking data integrity and calculating digital signatures. It is beneficial in the detection of unauthorized changes and the data sender and receiver verification. It is a function that maps data of an arbitrary size into data of a fixed size.

The general framework behind our idea is to construct scalable hashing functions and then use a scheduling algorithm to fit the desired hash length to each single processed task to fulfil the desired optimization criterion. This is performed locally after the tasks are assigned to the chosen computing unit. The main novelty of this system is the integration of secure scheduling and a scalable cryptography service: our solution is designed to allow the length of the hash output to be chosen accordingly to users' needs. If a task is delegated to be processed in a cloud edge, a light and fast version can be beneficial due to the limited computing capability of edge units. If a task is stored for a long time, longer hashes are necessary, as longer hashes are more resistant to possible attacks. We founded our solution on artificial neural networks (ANNs), instead of relying on commonly used hashing functions. Using ANNs enables us to set the hash length by choosing the network size. Additionally, we incorporated the ANN ability to learn chaotic behaviours to obtain the desired level of hash randomness. ANNs are proved to be high-speed and effective tools for mimicking very complex behaviours. After training, they can transform the input string into an output of a certain length.

Our solution is based on the Mackey–Glass (M–G) time series that can represent chaotic dynamics. Then artificial neural networks (ANN) are used to mimic that chaotic behaviour, and play the role of a hashing tool. Changing the ANN architecture (that is, adding neurons to the output layer or deleting them) enables obtaining different hash lengths. This feature is not offered by hashing functions that are being used in computational clouds. Our primary motivation for adopting the M–G equation is to provide a solution that offers well-proved randomization of the output. A hashing function should not be computationally distinguishable from a random oracle for messages of fixed length: most generators produce only pseudo-random numbers that are actually deterministic, whereas the usage of M–G provides a much stronger foundation.

Our proposal supports assuring the security of the data and cloud infrastructure: it may be implemented as a part of IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) models, where the responsibility for ensuring security in the cloud is based on CC provider actions (CSA, 2011). The system was developed to follow international cloud security guidance such as ENISA (2009) or ISO/IEC (2005); it is composed of an ANN, responsible for hashing, and two evolutionary algorithms: the global evolutionary scheduler (GES) with security constraints and the local evolutionary scheduler (LES). The GES maps all tasks to be executed into available computing units, while the LES maps different hash lengths into specific tasks. The appropriate schema is presented in Fig. 1. Using secure scheduling and a scalable cryptography service enables the fulfilment of the task security requirements while optimizing the usage of the computational units.

The proposed approach is demonstrated by applying three well-known statistical tests that enable evaluating fast hashing ANN strengths and weaknesses and using a use-case in which we apply a scalable hashing function to lower the idle time of the virtual machines in a cloud.

This paper presents new developments of our research on algorithms designed to enable adding security constraints into the process of task scheduling in CC, presented by Jakóbik *et al.* (2016; 2017a), Jakóbik and Wilczynski (2017), or Fernandez-Cerero *et al.* (2018). With respect to our previous results, the novel contributions provided by this work are the following:

- the solution that combines ANNs with Mackey–Glass chaotic time series to allow creating a potentially fully scalable hashing algorithm;
- a method of mixing messages with chaotic time series;
- the design and implementation of tests of hash quality obtained by different ANN architectures;
- the presentation of the results of a performance test campaign in the cloud testbed with the global scheduler.

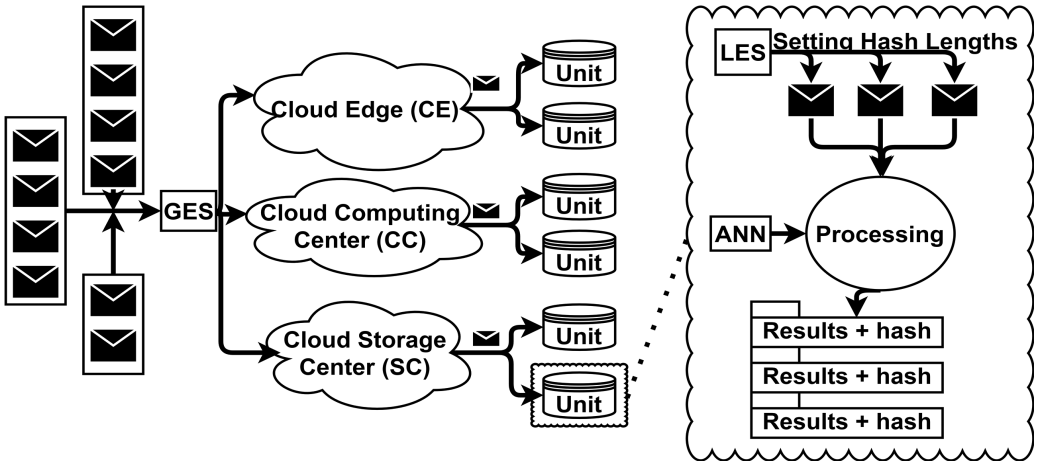

Fig. 1. Integration of GES, LES and the hashing algorithm inside the cloud infrastructure.

The rest of the paper is organised as follows: Section 2 presents the current state of research and points out the novelty of the presented approach; in Section 3 a theoretical model for the proposed hashing method, including security demands, trust levels aspects, flexible hashing, and a two-level scheduling mechanism is presented; Sections 5 and 6 describe experiments design and evaluation of the components of our algorithm; Section 7 contains a summary, conclusions based on experimental results, and planned future work.

2. Related work

Introducing security objectives is a new trend for building scheduling algorithms. Saleh and Dong (2013) combined real-time scheduling with security service enhancement. The scheduling unit uses the differentiated-earliest-deadline-first queues, and security enhancement is realized by adding a congestion control mechanism. Mohan *et al*. (2014) also show that security requirements can be specified as scheduling constraints. They present a solution based on fixed-priority (FP) real-time schedulers. Kaur *et al*. (2009) present a database transaction scheduler that considers two levels of security: a low-security level and a high-security level. Tadokoro *et al*. (2010) present the process of scheduling across VMs as constructed to be more immune to DoS attacks by dedicated isolation among VMs.

All those solutions do not allow fitting specific security services for a given batch of tasks: they include security constraints primarily for scheduling algorithms that use task queues. Also, the considered security levels that can be served are very limited. Our solution overcomes these drawbacks.

As for the hashing algorithms themselves, they enable calculating a fixed-length hash, for example, with 256, 386 or 512 bits (NIST, 2015; Dworkin, 2015). Specific functions, called light cryptography hash functions, offer smaller hashes, for example, from 60 to 160 bits (Gong, 2016). In almost all popular hashing methods, to change the output string length a user needs to change the hashing algorithm type.

Artificial neural networks (ANNs) are good candidates for hashing procedures due to their ability to mimick very complex behaviours (Haykin, 1998). Many earlier attempts to use ANNs for hashing purposes exist. Turcanik (2017b) used a recurrent neural network (RNN) composed of three layers of neurons and synapses between them for hashing with 160, 192, and 256 bits output. The author reports that "*the structural complexity of artificial neural networks creates big obstacles for their using in the real applications*" (Turcanik, 2017b). Yang *et al*. (2009) applied a cellular neural network with hyper-chaos characteristics for hashing. The chaos sequence is generated by iterating the cellular neural network with a Runge–Kutta algorithm. The authors generated and tested only a 128-bit hash.

In the work of Turcanik (2017a) an ANN with 512 neurons in the input layer, 128 neurons in the hidden layer, and 128 neurons in the output layer was tested. The network has a recurrent connection between the output and input layers to mimic the fact that the calculation of the hash function involves data from the previous step. The required properties of an ANN behaviour are obtained by the output layer of the ANN. The authors applied a piece-wise linear chaotic map for several times; only the one-way property of the presented hashing method was tested.

Abdoun *et al*. (2016) apply ANNs having two chaotic maps (a discrete skew tent map and a discrete piece-wise linear chaotic map) as transfer functions for neurons for hashing. The authors utilized a two-layer neural network structure without a hidden layer. They tested collision resistance and message sensitivity and compared results

with SHA-2. The main drawback of the presented solution is the fact that the sensitivity was tested for five different modifications of the given message only; additionally, the chaotic behaviour of the ANN was checked only for a short text consisting of 4 sentences.

In the work of Yee and De Silva (2002) the applicability of using a multilayer-perceptron (MLP) network as a hash algorithm is investigated: the MLP network considered consists of a hidden layer and an output layer. The hidden layer contains 64 neurons with 641 inputs, and the output layer contains 128 neurons. The network was tested to show the difficulty of recovering an input from the output. Additionally, collision resistance and resistance to birthday attacks (Bellare and Kohno, 2004) were tested. This solution was not compared with any widely used hashing method. Unfortunately, all these hashing procedures were tested very briefly, and tests provided by the authors are not sufficient to use their hashing algorithms for cryptography protocols.

There are very few hashing functions that offer flexibility in output length. Singh and Garg (2009) use such hashes for checking the integrity of data located in hard drives. The main drawback of this algorithm is that it is not cryptographically secure, therefore, it cannot be used for digital signatures. The solution introduced by Du *et al*. (2016), namely, the segmented hash algorithm, enables the usage of multiple logical hash tables. A modification of the hash function internal structure is obtained by having several hash tables with a different number of buckets each. Even if this method does not support obtaining a different length of the hash, it supports the computational effort of an algorithm change.

Wang and Li (2015) presented an algorithm that processes an input message with random lengths and produces various output lengths, such as 128, 160, 192, 224, or 256 bit, and processes messages in 1024-bits blocks. The algorithm is based on SHA design principles and does not use ANNs. Aggarwal and Verma (2015) offer a hashing procedure that is based on the RC6 algorithm: it can generate a hash value of various lengths. The algorithm presented in this paper is compared with SHA-256 and SHA-512: the authors compared the throughput of the considered algorithms and the time needed to produce hash values for files of different sizes. The security properties of the presented algorithm are based on the fact that this hash algorithm exploits the symmetric block cipher RC6; therefore, it inherits the properties of this cipher.

In the work of Kidon and Dobai (2017), hashing is based on the usage of a genetic algorithm: the length of the hash output is regulated by using the population size variable. This evolutionary method produces non-cryptographic hash functions.

The idea of using chaotic systems for building hash functions is the basis of the proposal presented by Rajeshwaran and Anil Kumar (2019): cellular automata are used to build hash functions; however, the resulting algorithm only enables the construction of long hashes. Huang and Wang (2019) use a two-dimensional logistic mapping and a two-dimensional Chebyshev mapping; nevertheless, this solution does not provide any possibility to arbitrarily set the hash length. Chugunkov *et al*. (2019) presented stochastic transformations of DOZEN family. This algorithm makes it possible to obtain a hash length of 512 bits. Most recent advances in the field of light cryptography standards presented in (NIST, 2019) does not offer scalable hashing methods, either.

In summary, in this work, the current state of the art is extended by proposing a hashing method that

- allows fitting security services by chaining the length of the hash output string;
- can be used for strong cryptography services producing 1024 or more bits hashes and as a light cryptography hash function producing hashes that may be smaller than 160 bits;
- allows changing the output string length without requiring a modification of the hashing algorithm type;
- is supported by a thorough and focused test campaign: the provided tests are sufficient to enable the use of the presented hashing algorithm for cryptography protocols;
- combines both chaotic time series and an ANN;
- enables considering many security levels.

3. Theoretical framework of the proposed system

In this section, all details related to the proposed system are presented. The system design enables us to utilize flexible hashing functions. In Section 3.1 information flow in the system is presented; in Section 3.2 tasks and system security parameters are described; in Section 3.3 the process of task mapping into computational units is presented.

3.1. Workflow of the system. In distributed environments, tasks may be gathered into batches, and each batch may contain many tasks. Tasks in a particular batch may differ in computational demands. Some of them demand more resources than others, and some are light to compute. Very often, a trade-off between the time of computation and the power of available VMs is considered; however, from a cryptographic point of view, the most demanding tasks are not those that utilize more resources to complete computations, but those whose

results are simply bigger in size, because cryptographic operations are done on the results of tasks, increasing overall computational effort.

In this paper, we extend our previous security-driven model of the computational units and tasks (Jakóbik *et al.*, 2017b; Fernandez-Cerero *et al.*, 2018; Barbierato *et al.*, 2019). In this model, a computational unit i ($i = 1, 2, \ldots, m$) is described by the following parameters:

- cc_i: computing capacity, expressed in floating point operations per second;
- tl_i: trust level, in the range of [0, 1].

The single task T_j is described by the following parameters:

- wl_j: workload parameter, in floating point operations [FLO];
- sd_j: security demand parameter, in $\{0, 1, 2, 3\}$;
- $D \in \{CE, CC, SC, \textit{undefined}\}$: task destination parameter;

with $j = 1, 2, \ldots, n$.

Our model proposes adding three novelty elements to the cloud system, to enable mapping the tasks into computing units and set the hash length for each task to perform the hashing operation (see Table 1); these elements will be described in the following.

3.2. Modeling task security demands and security services offered by VMs. Cloud providers offer VMs that can be configured according to users' needs: for example, Amazon Cloud offers several types of VMs, that differ in computational power, number of CPU cores, and available memory (Amazon, 2019). A user may implement his or her scheduling policies by defining a rule for the *RunTask* action. On the other hand, to minimize the cost of processing tasks in VMs, the user needs to optimize the time spent on security operations because he or she is charged for the working time of VMs: therefore, the user needs to consider the appropriate balance between security requirements and the financial cost.

To model such a situation and to provide a solution for secure task processing, we considered an SD vector describing the security demands of all tasks that will be processed in the single task batch, (Grzonka *et al.*, 2018):

$$SD = [sd_1, \ldots, sd_n], \tag{1}$$

where n is the number of tasks to be processed. Each component of the vector, sd_j, is the security demand value of the j-th task. We also defined a TL vector describing the security trust levels offered by the computing units (Grzonka *et al.*, 2018):

$$TL = [tl_1, \ldots, tl_m], \tag{2}$$

Table 1. Elements of the presented system and responsibilities of system actors.

Action	Actor
schedule tasks inside CE, CC, SC	GES
maps hash lengths into tasks	LES
task integrity checking by hashing	hashing ANN

where tl_i is the trust level offered by the i-th computing unit, and m denotes the number of all available units that may process tasks inside the cloud. Each task may be executed on a particular unit that offers a tl equal to or greater than the sd of that task. Following NIST (Bowen *et al.*, 2006), in this paper we assumed four levels of security demands and trust levels, that is $sd_j, tl_i \in \{0, 1, 2, 3\}$. According to this assumption, the security level zero means no security algorithms are involved in task processing. Examples of higher levels may be the following:

- level 1: light cryptography supporting on-line processing (Menezes *et al.*, 1996), hash length up to 180 bits;
- level 2: data processing cryptography (Schneier, 1995), advanced cryptography protocols, hash length between 512 to 1024 bits;
- level 3: data at rest cryptography (Google, 2016), the strongest cryptography (Al-Hamdani, 2011), hash length longer then 1024 bits.

Here $sd_j = 1$ means that cryptography level 1 must be applied to process the j-th task in a secure way; $tl_i = 2$ means that the i-th VM is equipped with services for security level 2.

3.3. Mapping tasks into computational units. The mapping of a particular task into the chosen computing unit can be performed to obtain the shortest possible time of the whole batch of task processing. The execution time of a single task depends on the workload of that task wl_j, and the computing capacity of the unit that will be processing this task. The ratio

$$\frac{wl_j}{cc_i} \tag{3}$$

defines the number of seconds that unit i will spend on computing task j (Kołodziej, 2012; Grzonka, 2018; Grzonka *et al.*, 2018; Jakóbik *et al.*, 2017a). Each schedule can be written as the list of tasks that are assigned to consecutive computing units:

$$\textit{Schedule} = \{T_{\sigma(1)}, T_{\sigma(2)}, \ldots, T_{\sigma(n)}\}, \tag{4}$$

where σ denotes the permutation of the set of task numbers $\{j = 1, 2, \ldots, n\}$. The schedule which will be chosen to be executed must fulfil the scheduling criterion. One example, it can be the shortest time of whole batch of task processing:

$$\mathit{Time}_{\min} = \min_{S \in \mathit{Schedules}} \max_{j \in \mathit{Tasks}} \frac{wl_j}{cc_i}. \tag{5}$$

Security demands and trust levels enforce additional constraints in the above minimization problem. Various optimization methods can be used to solve the scheduling optimization problem (see Eqn. (5)).

4. Theoretical framework of the flexible hashing function

In Section 4.1 the flexible hashing function concept and its security parameters are described. In Section 4.2 details related to the chosen hash lengths are presented. In Section 4.3 the process of hashing ANN training and testing is described.

4.1. Flexible hashing function. The system incorporates a scalable cryptographic protocol in the form of a scalable hashing function h. Such a function can be represented as

$$\forall x \in X, h : x \to y \wedge \neg(\exists g : y \to x) \tag{6}$$

with the additional restriction

$$h : \{0,1\}^* \to \{0,1\}^n, \quad n \geq 1. \tag{7}$$

In this paper, we consider only one way hash functions (OWHFs) by Merkle, having the following properties (Merkle, 1979):

- it does not have any constraint on its input data size;
- its output has a constant length;
- it is computable with small effort;
- it is pre-image resistant: by knowing h and $h(x)$, it is computationally infeasible to determine x;
- it is second pre-image resistant: by knowing h and x, it is computationally infeasible to find an $x' \neq x$ such that $h(x) = h(x')$;
- it is collision resistant: it is hard to find a pair (x, x') of values, $x \neq x'$, which have the same hash value $(h(x) = h(x'))$;
- the probability of finding a second pre-image for a randomly chosen hash function may be neglected.

These requirements guarantee that the OWHF can be used for building cryptography protocols (Tchórzewski and Jakóbik, 2019).

The number of output bits may depend on the SDs given by the user. On the other hand, the TLs offered by the computing units are fitted to the incoming tasks SDs.

Once the schedule is known, each of the computing units has to process the set of tasks. Without loss of problem generality, we may assume that tasks for the unit number i were re-numerated as $T_1, T_2, \ldots, T_{n^i}$ and their workload are denoted by $wl_1, wl_2, \ldots, wl_{n^i}$, respectively. $\mathit{Time}_{\min}$ for this batch of tasks is known; therefore, the idle time for this unit is

$$T_{\mathit{idle}}(i) = \mathit{Time}_{\min} - \sum_{j=1}^{n^i} \frac{wl_j}{cc_i}. \tag{8}$$

In our model, we assume that the hash lengths offered by unit i are $n^i_{\min}, \ldots, n^i_{\max}$ with a reasonable equal step between them denoted by Δ^i bits. Those parameters may depend on the computing capacity of the unit or the software which is installed.

4.2. Setting the hash length. The local scheduler ELS operates on the basis of the estimated number of operations to be computed for the j-th task on the i-th computing unit:

$$o(i,j) = \Bigg\{ N(sd_j, wl_j, tl_i, cc_i, \mathit{inputSize}_j, \mathit{outputSize}_j, n^i_{\min} + k\Delta^i) \times \frac{\mathit{outputSize}_j}{n^i_{\min} + k\Delta^i} \Bigg\}. \tag{9}$$

The scheduling procedure aims to maximize the number of operations spent on hashing, which can be defined as follows:

$$HO_i = \max_{S \in \mathit{Schedules}^i} \Big\{ \max_{j \in \mathit{Tasks}_i} \{o(i,j)\} \Big\}. \tag{10}$$

Tasks_i is the set of tasks in the batch that were assigned to computing unit i and $\mathit{Schedules}^i$ is the set of all schedules which can be generated for the tasks that should be computed by unit number i considering the security mapping: $o(i,j) = \mathit{NaN}$ (not a number) for i and j such that $sd_j > tl_i$.

An additional constraint is in the form of the estimated idle time of the unit:

$$\frac{HO_i}{cc_i} \leq T_{\mathit{idle}}(i), \tag{11}$$

for all $i = 1, 2, \ldots, m$.

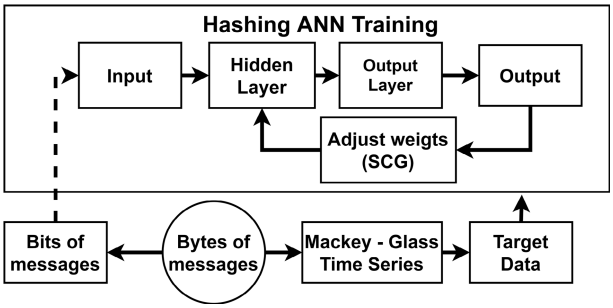

Fig. 2. Hashing ANN training scheme.

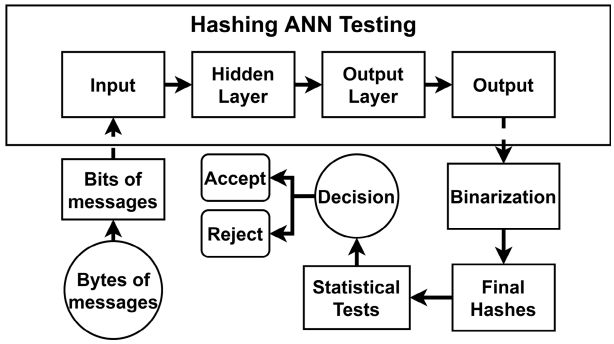

Fig. 3. Hashing ANN testing scheme.

4.3. Hashing ANN. All ANNs considered are two-layer feed-forward networks. From among many different learning algorithms (cf., e.g., Jankowski and Linowiecki, 2019) the scaled conjugate gradient (SCG) method was chosen due to the large training set size. Variable *hLen* stands for the chosen hash length. The ANN structure can be represented as I-HL-OL-O, where I denotes the input matrix containing binary vectors of length *hLen*, HL denotes the hidden layer and contained *SN* sigmoid neurons, OL denotes the output layer containing *hLen* linear output neurons, and O denotes the output matrix, where each row contains *hLen* real values produced by the network (Haykin, 1998). Different combinations of *hLen* and SN have been tested (see Table 2). The process of hashing ANN training is presented in Fig. 2.

Firstly, random strings of bytes are generated. Each byte stream has the same length equal to $hLen/8$ and represents a random message to be hashed. Those bytes strings are mapped into chosen time series (see Section 5.1). In order to obtain chaotic behaviour Mackey–Glass (M–G) time series have been chosen (Gholipour *et al*., 2006; Mahjoob *et al*., 2008):

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = -\beta x(t) + \frac{\alpha x(t-\tau)}{1+x^n(t-\tau)}, \tag{12}$$

where $x(t)$ denotes the state value at time t. With parameters $n \sim 10$ and $\tau > 16.8$ (which assure M–G chaotic behaviour), the following discrete form of (12) was used for the generation of the target data for the hashing ANNs:

$$x_{t+1} = -\beta x_t + \frac{\alpha x_{t-\tau}}{1+x_{t-\tau}^n}, \tag{13}$$

where the time difference between x_{t+1} and x_t is equal to Δt. Note that the M–G equation demands specification of initial data stored in the time frame $[-\tau, -\Delta t]$. Time 0 indicates the start of computations. These data will be further called the *history vector*. Equation (13) was numerically solved via the 4th order Runge–Kutta method (RK4) (Atkinson, 1978; Butcher, 2008; Cococcioni, 2021).

For each message, the M–G equation is generated and solved separately. Time series results represent ANN target data that are potential hashes. After target generation, data are converted into bit strings of length *hLen*, and those bit strings represent ANN input data. The process of hashing ANN testing is presented in Fig. 3.

The testing process can be divided into three steps:

Step 1: Random messages bytes are generated. Each message consists of $hLen/8$ bytes. Generated vectors are transformed into their binary representation and gathered into one input matrix. Each input matrix row contains *hLen* binary values.

Step 2: A trained ANN was used to generate an output matrix containing real values.

Step 3: The values returned by the ANN are transformed into binary values and evaluated with the use of statistical tests. Test results determined whether or not a particular ANN could serve as a hashing function.

All technical details related to the ANN training, ANN testing, statistical tests, and Mackey–Glass equation parameters are presented in Sections 5 and 6.

5. Experimental design

In Section 5.1, parameters of the Mackey–Glass equation are presented along with the method for coding messages into the equation, tested combinations of hash lengths and ANN structures. In Section 5.2 the generation of training and testing data is described. In Section 5.3 tasks and their security demands are discussed.

5.1. Adjustment of Mackey–Glass parameters. The parameters of Eqn. (13) were set to $\alpha = 0.2$, $\beta = 0.1$, $n = 10$. Furthermore, Δt and τ depended on the chosen hash length *hLen*. The point at time 0 (start of computations), that is x^0, was a random number from range $[0, 2]$. The

number of sigmoid neurons in the hidden layer, (*SN*, was also dependent on *hLen*. All tested combinations of *hLen*, Δt, τ and *SN* are presented in Table 2.

Parameters Δt and τ determined the size of a Mackey–Glass (M–G) history vector (initial data in the time frame from $-\tau$ to $-\Delta t$). The number of elements that could be stored in the history vector (HV) was computed as

$$HV_{size} = \frac{\tau}{\Delta t}. \tag{14}$$

The size of input messages was in all cases the same as the chosen hash length. Thus, for example, for 128-bit hashes, 16 random byte messages were generated. In such a case, in accordance with (14) and data presented in Table 2, the history vector contained 16 free spaces for each message. Respecting assumptions assuring M–G chaotic behaviour, message bytes could be used as a history vector. This solution enabled us to algorithmically bond each message with the equation itself. The same solution was used for longer hashes, that is in 256-bit hashes and, respectively, 32 byte messages, 512 bit hashes and 64 byte messages, etc.

In each testing batch described in Section 5.2 each message was used to generate unique M–G results. Formally one initial message M can be presented as

$$M = [B_1, B_2, B_3, \ldots, B_{\frac{hLen}{8}}], \quad B \in [0, 255] \cap \mathbb{N}, \tag{15}$$

where B_1 denotes the first byte of message M, B_2 the second one, and, so on. This corresponds to the message M history vector

$$HV_M = [M[0]^{-\tau}, M[1]^{-\tau+\Delta t}, M[2]^{-\tau+2\Delta t}, \ldots, M[i]^{-\Delta t}], \quad i = 1, \ldots, \frac{hLen}{8}. \tag{16}$$

5.2. Hashing ANN training and testing set generation. For each tested hash length *hLen* (see Table 2), ANN input, target, and test data were generated in the same manner. MB denotes the matrix of 5000 different random byte strings representing messages

$$MB[i] = M_i, \quad i = 1, \ldots, 5000, \tag{17}$$

where M_i denotes the bytes of the i-th message (see Eqn. (15)). For each message M_i, the bytes of the message were coded into an M–G history vector (HV_{M_i}) and the equation was solved. In all cases 24000 samples of the solution were generated. As a final result, a matrix containing 5000 rows, and 24000 real values from the range $[0, 2]$ in each row was obtained,

$$MG[i] = [x_1, x_2, \ldots, x_{24000}], \quad i = 1, \ldots, 5000, \quad x \in [0, 2]. \tag{18}$$

Table 2. Tested combinations of the hash length $hLen$ given in bits, the number of neurons in the hidden layer SN, and the M–G parameters Δt and τ.

	ANN	M–G	
hLen	SN (No)	Δt	τ
128	25, 50, 75, 100, 128, 150, 175, 200, 225	10	160
256	50, 100, 150, 200, 256, 300, 350, 400, 450	1	32
512	200, 300, 400, 512, 600, 700, 800	1	64
1024	200, 400, 600, 800, 1024, 1200, 1400, 1600, 1800	1	128

The hashing ANNs target data *train*^{target} were generated in accordance with

$$train^{target}[i][j] = MG[i]\left[j \times \left\lfloor \frac{24000}{hLen} \right\rfloor\right], \quad i = 1, \ldots, 5000, \quad j = 1, \ldots, hLen. \tag{19}$$

The whole space of solutions was used in target generation to avoid local dependencies between samples (e.g., samples on an ascending or descending slope).

After target generation, messages from array MB were converted from byte strings into binary strings. Those binary strings were used as the ANN input training set *train*^{input}:

$$train^{input}[i] = [MB[i]_1, MB[i]_2, \ldots, MB[i]_{hLen}], \quad MB[i]_* \in \{0, 1\}, \quad i = 1, \ldots, 5000, \tag{20}$$

where $MB[i]_1$ is the first bit of byte B_1 (cf. (15)) of message M_i, $MB[i]_2$ is the second bit of byte B_1, $MB[i]_9$ is the first bit of byte B_2, and so forth. Finally, a pair (*train*^{input}, *train*^{target}) could be used as an ANN training set. The main advantage of the utilization of the M–G equation in ANN training data preparation is in the history vector. Each generated (cf. (18)), and compressed (cf. (19)) time series *train*$^{target}[i]$ was algorithmically bonded with the corresponding message *train*$^{input}[i]$, through the history vector HV_{M_i}. Generated series represented deterministic chaos but related to a particular input message.

Input test data *test*^{input} were generated analogously to *train*^{input}. Another matrix of random byte strings was generated, cf. Eqn. (17), and converted to its binary form cf. Eqn. (20). The size of *test*^{input} was also equal to 5000, and each message had *hLen* bits. However, none of the newly generated messages was present in the training set. In both sets, messages also did not repeat.

For a chosen hash length, training and testing data were generated once and were used for all ANNs

generating this particular hash length. The only difference in ANNs generating hashes of the same length was in the number of sigmoid neurons (*SN*) in their hidden layer. Experimental results are presented in Section 6.2.

5.3. Tasks and their security demands. In order to obtain diversity in workload, we considered tasks that were based on processing images. Twenty tasks with different workloads were considered. To obtain them, we generated seven types of images that differ in size. A single task was applying a masking function (blurring) or ciphering via the RSA algorithm single image. The value of SD of each task j, that is sd_j, was randomized over the set $\{1, 2, 3\}$. We did not consider $sd_j = 0$ which represents the lack of security demands. The destination of the j-th task was randomly chosen from the three possible computing units $D(j) \in \{CE, CC, SC\}$. A hashing procedure was applied to the original images to check their integrity and inviolability. The minimal hash length was set to 60, and the maximal was set to 1024 bits. The computational effort of each task was calculated based on the size of the image in bytes.

6. System evaluation

All our security algorithms have been implemented in Java; thus CloudSim testbed (`www.cloudbus.org`), also based on Java, was used as a testing tool. It enables the implementation of scheduling algorithms and monitors the time spent on each tested cloud operation. The examined workload was generated according to the day-and-night pattern (Fernandez-Cerero *et al*., 2018) to represent a realistic workload of a CC system. Hashing ANNs were implemented in MATLAB 2017. Section 6.1 presents the ANN output data format, the binarization method, and statistical tests used to assess the ANN performance. In Section 6.2 results of tests are discussed, and conclusions are presented. In Section 6.3 the whole system performance is discussed.

6.1. Hashing ANN testing procedure. The testing procedure was the same for all generated ANNs. A pair ($train^{input}$, $train^{target}$) was generated once for all ANNs producing the same hash length (see Section 5.2), and this pair was used as a training set. Then $input^{test}$ (also generated once for ANNs producing the same length hash) was passed as an input for trained ANNs. The output returned by the ANN is

$$output[i] = [o_1, o_2, \ldots, o_{hLen}], \tag{21}$$

where $o \in [0, 2]$, $\quad i = 1, \ldots, 5000$. $Output[i]$ represents the value returned by the ANN for the corresponding message $test^{\text{input}}[i]$. To form a hash, the output had to be binarized. This was done accordingly to

$$hashes[i][j] = \begin{cases} 1 & \text{if } OUTPUT[i][j] \geq AVG_j, \\ 0 & \text{otherwise,} \end{cases} \tag{22}$$

where AVG_j represents the average value in column j. If the value in cell $OUTPUT[i][j]$, that is the i-th row ($i = 1$, $\ldots$, 5000) and the j-th column ($j = 1, \ldots, hLen$), was greater then the average value of the j-th column, the bit was set to 1, otherwise it was set to 0; in this way, a hash matrix was generated for each tested ANN.

To determine whether or not a particular ANN can serve as a hashing function, three statistical tests were performed as follows.

Bit prediction test (BPT). It measures the probability of successful prediction of a bit value on a particular position.

The probability of having a bit 1 in a particular position in the $output$ matrix can be computed as

$$PO_1(i) = \frac{1}{5000} \sum_{j=1}^{5000} output[j][i], \tag{23}$$

where $i = 1, \ldots, hLen$, and the results can be gathered into the probability vector

$$PV = [PO_1(1), PO_1(2), \ldots, PO_1(hLen)]. \tag{24}$$

Our test assumed calculation of statistic Z with the usage of a t-Student test with significance level $\alpha = 5\%$. The expected value is equal to 50%.

Hamming distance test (HDT). It measures the Hamming distance between a message and its corresponding hash.

The Hamming distance is given by the number of ones in a string, $test^{input}[i] \oplus output[i]$. The expected distance is equal to $hLen/2$. The Hamming distance was calculated for each message-hash pair separately. The results were gathered into the vector

$$dist = [d_1, d_2, \ldots, d_{5000}], \tag{25}$$

where d_i is the Hamming distance in the i-th message-hash pair. This vector was also tested using the t-Student test with $\alpha = 5\%$.

Series (Wald–Wolfowitz) test (ST). It is performed on the returned hashes determining the hashes randomness.

The series test was performed on each hash in the *output* matrix separately. Then the percent of hashes that failed was calculated. The matrix *output* was passing the test when no more than 5% of its hashes failed the series test.

Collision test. A collision is a situation when a hashing function produces the same hash value for two different messages. All hashing ANNs were free from collisions.

Interested readers can find all details about the tests (algorithms and methodology) in (Tchórzewski *et al.*, 2019). In all statistical tests the significance level α was equal to 5%. The sample sizes were big enough to assume the standard distribution of data ($N(0,1)$). Thus the confidence interval was equal to $[0, 1.96)$ for each test statistics $|Z|$. The results are presented in Table 3.

6.2. Summary hashing ANN test results. This section describes the results presented in Table 3. The performed tests were:

- 128-*SN*-128: one network passed all the tests (128-50-128). This type of network had satisfactory results in BPT and ST, but one of the worst results of the HDT. This may indicate that generated hashes were sometimes too similar to the original messages or too random.
- 256-*SN*-256: the best candidate was 256-100-256, which was very close to the 5% limit in ST and passed the rest of the tests; ST this type of network had the weakest results in ST; this may indicate that some hashes did not have random distributions and potentially were not free from internal dependencies.
- 512-*SN*-512: none of the ANNs passed all tests. However, many candidates should be still considered. BPT and HDT results are the best in comparison with the rest of ANNs: the only potential problem is related to slightly weaker results of ST, still relatively close to the established critical point.
- 1024-*SN*-1024: this type of network had the worst ST results but, in most cases, acceptable results of the rest of the tests; this may indicate that the longer the hash, the less randomly distributed it is.

The test data indicate that the statistically best results appeared in ANNs with sigmoid neurons and *SN* close to 40% *hLen*. One ANN passed all tests, but some candidates slightly exceeded the ST limit and passed the rest of the tests. Those ANNs may be potentially used to create short-term hashes (for example, signing temporary data in a cloud but not storing passwords). Tchórzewski and Jakóbik (2019) presented results of the same tests performed on certificated hashing functions (these are SHA-1, SHA-512, and SHA3-512) (note that SHA-1 is currently not recommended to be used). However, all of them passed all tests.

6.3. Scalable hashing performance tests in the cloud with a global scheduler. To check how the flexible hashing may influence the computational effort for the batch of processed tasks, we have simulated a multi-cloud environment using the infrastructure shown in Table 5 and the CloudSim Simulator, `http://www.cloudbus.org/cloudsim/`. These tests aimed to examine how much idle time a cloud system may devote to hashing procedures and to compare flexible hashing methods with fixed hash length algorithms. 256 or 512 bit length hashings were considered as the fixed hashing method references to compare with the internationally standardized SHA hashing functions family.

At first, the number of instructions necessary for computing the single block of a hash input was determined (see Table 4).

Table 3. Statistical tests of hashing functions. BPT stands for the bits prediction test and HDT for the Hamming distance test. Both of these tests are passed when the value in the column is less than 1.96. The series test shows percent of hashes that failed the test. The test is passed when the value is less than 5%.

No	ANN structure	BPT	HDT	ST (%)
1	128-25-128	2.52	3.24	4.82
2	128-50-128	0.98	1.55	4.14
3	128-75-128	0.08	9.80	4.90
4	128-100-128	0.05	9.69	5.12
5	128-128-128	1.41	4.17	4.64
6	128-150-128	1.24	9.37	4.62
7	128-175-128	3.84	4.72	4.58
8	128-200-128	0.57	9.68	4.90
9	128-225-128	1.32	8.58	4.48
10	256-50-256	0.04	4.75	4.96
11	256-100-256	1.66	0.36	6.42
12	256-150-256	0.76	2.82	10.06
13	256-200-256	1.20	2.45	11.22
14	256-256-256	0.55	0.65	14.10
15	256-300-256	0.41	0.25	15.24
16	256-350-256	1.82	1.81	14.14
17	256-400-256	0.90	0.04	14.02
18	256-450-256	0.76	3.00	11.10
19	512-200-512	1.86	0.04	5.42
20	512-300-512	0.27	0.97	5.94
21	512-400-512	0.68	0.02	6.50
22	512-512-512	0.35	2.26	7.42
23	512-600-512	1.53	4.72	7.24
24	512-700-512	1.77	1.60	7.30
25	512-800-512	1.50	0.45	6.34
26	1024-200-1024	2.76	3.24	9.94
27	1024-400-1024	0.92	2.79	9.08
28	1024-600-1024	0.10	4.58	35.42
29	1024-800-1024	1.38	0.31	59.38
30	1024-1024-1024	0.01	1.47	71.16
31	1024-1200-1024	0.75	0.33	64.42
32	1024-1400-1024	1.88	1.01	50.42
33	1024-1600-1024	1.44	0.92	32.90
34	1024-1800-1024	0.62	0.31	19.38

Table 4. Number of instructions necessary for computing a single block of hash for chosen ANNs. Here $f = N(sd_j, wl_j, tl_i, cc_i, \mathit{inputSize}_j, \mathit{outputSize}_j, n^i_{\min} + k\Delta^i)$.

ANN structure	$N()$	f
64-32-64	2277	$\mathit{inputSize}_j = 64$, $n^i_{\min} + k\Delta^i = 64$
128-64-128	7316	$\mathit{inputSize}_j = 128$, $n^i_{\min} + k\Delta^i = 128$
256-128-256	26317	$\mathit{inputSize}_j = 256$, $n^i_{\min} + k\Delta^i = 256$
512-256-512	144995	$\mathit{inputSize}_j = 512$, $n^i_{\min} + k\Delta^i = 512$
1024-512-1024	493735	$\mathit{inputSize}_j = 1024$, $n^i_{\min} + k\Delta^i = 1024$
2048-1024-2048	2417071	$\mathit{inputSize}_j = 2048$, $n^i_{\min} + k\Delta^i = 2048$
4096-2048-4096	21502772	$\mathit{inputSize}_j = 4096$, $n^i_{\min} + k\Delta^i = 4096$
8192-4096-8192	83477520	$\mathit{inputSize}_j = 8192$, $n^i_{\min} + k\Delta^i = 8192$

The number of operations for a single block consumed by a hashing function was found based on numerical tests performed on a PC with 16 GB RAM, processor Intel Pentium CPU G4400 @ 3,30 GHz, and HDD ST1000DM010-2EP102. This computing unit speed was chosen as the base for implementing 5 VMs in the CloudSim simulator having the speed of tested physical computing unit, multiplied by factors of 0.2, 0.5, 1, 2, 4 (see Table 6).

Batches of tasks were generated according to the day and night pattern (Fernandez-Cerero *et al.*, 2018). The workload of tasks was increased and decreased as depicted in Table 6.

Each batch consists of 100 tasks. The evolutionary scheduler GES was assigning tasks to particular VMs. The scheduling problem was solved using the evolutionary algorithm proposed by Jakóbik *et al.* (2016; 2017b). This scheduler was selected because it offers the mapping between tasks and VMs for all VMs in a single experiment run. Table 7 presents the results of the simulation for a randomly chosen batch of 100 jobs.

For example, applying 64-bit hash for task D(1) results in 3341550000 operations; applying 128-bit hash consumes 548700000 operations; using 512-bit hash adds 2718656250 operations; using 1024-bit hash adds 4628765625 operations; using 2048-bit adds 11330020312 operations; for 4096-bit hash, it needs 50397121875 operations; and for 8192-bit hash 97825218750 operations. That results in respectively 14.5, 2.3, 11.8, 20, 49, 219, 425 second of VM2 computational power. For hashing task D(2) we obtained the same amount of time because both tasks have the same output bits length. VM2 must wait 150 second for VM1 to complete the batch (see Table 8). The longest hash that we may apply is the 2048-bit hash; then VM2 idle time is equal to $150 - 49 - 49 = 52$ seconds. For all VMs, we can find the longest hashing procedures that will not exceed the makespan of the batch of all tasks.

For all VMs used for simulation, we enable them to increase data processing security by providing the fingerprints of stored data. Additionally, the system idle time was decreased by 54%.

7. Conclusions and future work

We presented an intelligent hashing algorithm based on an ANN which learned to mimic a chaotic Mackey–Glass time series. We provided a simple example of the algorithm used for batch task processing in computational clouds with schedulers, based on consuming the idle time of the VMs waiting for the batch completion. Our algorithm supports security services in computational clouds and can improve the quality of security cloud services during data integrity verification or digital signature generation.

To obtain a scalable hashing algorithm, it is necessary to prepare a set of ANNs. The number of neurons in the output layers should be equal to the hash length that we would like to apply. The next stage of preparation of hashing ANN is learning: the training set is generated based on the discretized Mackey–Glass equation with the parameters ensuring chaotic behaviour.

We have tested the hashing algorithm using three independent tests. The Hamming distance test measured the Hamming distance between the ANN outputs and inputs. The bit prediction test detected whether a particular bit of outputs can be predicted: the test is based on calculating the probability of unity being present in every bit position. We have also applied the Wald-Wolfowitz series test on ANN outputs to detect whether all hashes were generated randomly.

The presented experimental results confirm that the ANN-based hashing algorithm presented in this paper fulfills all requirements to be a hashing function creating short-term hashes. The CloudSim based simulation for the chosen task sets demonstrates the usage of the hashing procedure and confirms the effectiveness of the presented system.

The presented system may be used by cloud service providers and cloud consumers using IaaS or PaaS models where schedulers are applied to balance the cloud loading. The proposed approach produces the best advantages in cloud systems that are unbalanced as far as idle time is concerned; it enables to increase data and communication security without changing the makespan: this can be

Table 5. Tasks and their characteristics in terms of workload, SD value, and minimal and maximal hashing computational effort: $N(sd_j, wl_j, tl_i, cc_i, inputSize_j, outputSize_j, n^i_{\min}+k\Delta^i)$, $k=0$ for $n_{\min}=60$ was denoted as $N(\ldots,\min)$, $k=2$ for $n_{\min}=512$ was denoted as $N(\ldots,\max)$. Security demand for a 200×200 pixel image was randomly chosen. For each image of 400×400 pixels or greater requiring RSA 2048 ciphering $SD=3$, for images greater than 200×200 pixels and smaller than 800×800 pixels requiring RSA 1024 ciphering $SD=2$, for greater images $SD=3$. The destination for each task was randomly chosen according to the following rules: if a task required RSA 2048 ciphering, D was set to 2 or 3 (CC or SC). Tasks requiring the blurring operation were set to 1 or 2 or 3, except 200×200 and 1400×1400 pixel images (the smallest and the biggest), where $D=1$ or $D=2$. For RSA, the 1024 ciphering task destination was set to $D=2$ or $D=3$.

j	Dest.	Task descript.	wl_j [FLO]	$inputSize_j$ [b]	sd	o min	o max
1	D(1)=1/2	200×200 blur	180281484	960 000	$sd_1=1/2/3$	N(...,min)×16000	N(...,max)×938
2	D(2)=2/3	200×200 RSA1024	29814507241	960 000	$sd_2=1/2/3$	N(...,min)×16000	NN(...,max)×938
3	D(3)=2/3	200×200 RSA2048	95619636130	960 000	$sd_3=1/2/3$	N(...,min)×16000	N(...,max)×938
4	D(4)=1/2/3	400×400 blur	366694660	840 000	$sd_4=1$	N(...,min)×64000	N(...,max)×3750
5	D(5)=2/3	400×400 RSA1024	114485889326	840 000	$sd_5=2$	N(...,min)×64000	N(...,max)×3750
6	D(6)=2/3	400×400 RSA2048	383009262064	840 000	$sd_6=3$	N(...,min)×64000	N(...,max)×3750
7	D(7)=1/2/3	600×600 blur	582192590	8 640 000	$sd_7=1$	N(...,min)×144000	N(...,max)× 8438
8	D(8)=2/3	600×600 RSA1024	263890855494	8 640 000	$sd_8=2$	N(...,min)×144000	N(...,max)×8438
9	D(9)=2/3	600×600 RSA2048	861590710700	8 640 000	$sd_9=3$	N(...,min)×144000	N(...,max)×8438
10	D(10)=1/2/3	800×800 blur	935976091	15 360 000	$sd_{10}=1$	N(...,min)×256000	N(...,max)×15000
11	D(11)=2/3	800×800 RSA1024	469494309713	15 360 000	$sd_{11}=3$	N(...,min)×256000	N(...,max)×s15000
12	D(12)=2/3	800×800 RSA2048	1534304245283	15 360 000	$sd_{12}=3$	N(...,min)×256000	N(...,max)×15000
13	D(13)=1/2/3	1000×1000 blur	1381754383	24 000 000	$sd_{13}=1$	N(...,min)×400000	N(...,max)×23438
14	D(14)=2/3	1000×1000 RSA1024	734069416861	24 000 000	$sd_{14}=3$	N(...,min)×400000	N(...,max)×23438
15	D(15)=2/3	1000×1000 RSA2048	2407671820331	24 000 000	$sd_{15}=3$	N(...,min)×400000	N(...,max)×23438
16	D(16)=1/2/3	1200×1200 blur	1907429016	34 560 000	$sd_{16}=1$	N(...,min)×576000	N(...,max)×33750
17	D(17)=2/3	1200×1200 RSA1024	1057640189975	34 560 000	$sd_{17}=3$	N(...,min)×576000	N(...,max)×33750
18	D(18)=2/3	1200×1200 RSA2048	3471082534894	34 560 000	$sd_{18}=3$	N(...,min)×576000	N(...,max)×33750
19	D(19)=1/2	1400×1400 blur	2541112486	47 040 000	$sd_{19}=1$	N(...,min)×784000	N(...,max)×45938
20	D(20)=2/3	1400×1400 RSA1024	1520914612793	47 040 000	$sd_{20}=3$	N(...,min)×784000	N(...,max)×45938

Table 6. Characteristics of the simulated cloud and of the VMs used for simulation. The tested VMs' computational capacities were chosen according to the typical computing power for commercial clouds, e.g., Amazon Cloud.

Cloud Edge, CE	cc	tl$\in \{0,1,2,3\}$
VM number, type	[MIPS]	min:max
VM1, tiny	$0.2\times461.9= 92.38$	0:1
VM2, tiny	$0.5\times461.9=230.95$	0:2
Cloud Computing Center, CC	cc	tl$\in \{0,1,2,3\}$
VM number, type	[MIPS]	min:max
VM3, small	461.9	0:2
VM4, large	$2\times$ 461.9=923.8	0:3
Cloud Storage Center, SC	cc	tl$\in \{0,1,2,3\}$
VM number, type	[MIPS]	min:max
VM5, medium	$4\times$ 461.9=1847.6	0:3

Table 7. Scheduling 100 jobs into VM1, ... ,VM5. Results of applying the flexible hashing functions: mean values of idle time of the VMs for a chosen batch. The percentage indicates the system initial idle time that remained idle after applying flexible security hashing.

CC part	no hashing	512 hashing	flexible
Edge Computing Center	100%	84%	34 %
Storage Center	100%	97.5%	48.7%
	100%	98.7%	57%

an advantage in systems made of computational units that vary in their computing abilities. In such a case, slower units use shorter hash lengths, whereas faster units calculate longer hashes based on the same presented algorithm. This enables better load balancing inside the cloud system itself.

As future work, we intend to improve the presented model by:

- considering time series defined by other chaotic systems;
- further analyzing the impact of the ANN learning algorithm;
- testing the usage of recurrent or deep learning ANNs instead of a simple two-layer feed-forward ANN;
- validating the benefits of using scalable hashing function to build digital signature services;
- introducing local scheduling schemes supporting batches of tasks with time constraints other than those presented.

References

Abdoun, N., El Assad, S., Taha, M.A., Assaf, R., Deforges, O. and Khalil, M. (2016). Secure hash algorithm based on efficient chaotic neural network, *2016 International Conference on Communications (COMM), Bucharest, Romania*, pp. 405–410.

Aggarwal, K. and Verma, H.K. (2015). Hash RC6—variable length hash algorithm using RC6, *2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, India*, pp. 450–456.

Al-Hamdani, W.A. (2011). Elliptic curve for data protection, *Proceedings of the 2011 Information Security Curriculum Development Conference, InfoSecCD'11, New York, USA*, pp. 1–14.

Amazon (2019). Amazon EC2 instance types, *Amazon Web Services*, `https://aws.amazon.com/ec2/instance-types/`.

Atkinson, K.E. (1978). *An Introduction to Numerical Analysis*, 2nd Edn, Wiley, Hoboken.

Barbierato, E., Gribaudo, M., Iacono, M. and Jakóbik, A. (2019). Exploiting CloudSim in a multiformalism modeling approach for cloud based systems, *Simulation Modelling Practice and Theory* **93**: 133–147.

Bellare, M. and Kohno, T. (2004). Hash function balance and its impact on birthday attacks, *in* C. Cachin and J.L. Camenisch (Eds), *Advances in Cryptology, EUROCRYPT 2004*, Springer, Berlin, pp. 401–418.

Bowen, P., Hash, J. and Wilson, M. (2006). *Information Security Handbook: A Guide for Managers*, NIST Special Publication 800-100, National Institute of Standards and Technology, Gaithersburg, `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-100.pdf`.

Butcher, J.C. (2008). *Numerical Methods for Ordinary Differential Equations*, 2nd Edn, Wiley, Hoboken

Chugunkov, I.V., Ivanov, M.A. and Kliuchnikova, B.V. (2019). Hash functions are based on three-dimensional stochastic transformations, *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg/Moscow, Russia*, pp. 202–205.

Table 8. Results of scheduling a randomly chosen batch of 10 jobs: 4×D(1), 4×D(2), 2×D(3) into VM1, VM2, VM3, VM4, and VM5, assuming an equal task number per VM. Results of the application of the scalable hashing functions: values of idle time of the VMs for a chosen batch of tasks. For simplicity, we assume the same hash type for all tasks scheduled at each single VM.

CC part	Jobs	$Time_{\min}$	T_{idle}	T_{idle} 64	T_{idle} 128	T_{idle} 512	T_{idle} 1024	T_{idle} 2048	T_{idle} 4096	T_{idle} 8192
VM1	D(1), D(3)	280 sec.	0	0	0	0	0	0	0	0
VM2	D(1), D(2)	130 sec.	150 sec.	121	145.4	126.4	110	52	–	–
VM3	D(1). D(2)	65 sec.	215 sec.	200	212.6	203	194.9	165.9	–	–
VM4	D(1), D(2)	32 sec.	248 sec.	240.7	246.8	242	237.9	223.4	138.9	36.2
VM5	D(2), D(3)	32 sec.	248 sec.	244.3	247.4	245	242.9	235.7	193.4	142.1

Cococcioni, M. (2021). Mackey–Glass time series generator, *MathWorks*, https://www.mathworks.com/matlabcentral/fileexchange/24390-mackey-glass-time-series-generator.

CSA (2011). *Security Guidance for Critical Areas of Focus in Cloud Computing V3.0*, Cloud Security Alliance, Bellingham, https://downloads.cloudsecurityalliance.org/assets/research/security-guidance/csaguide.v3.0.pdf.

Du, Y., He, G. and Yu, D. (2016). Efficient hashing technique based on bloom filter for high-speed network, *8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), Hangzhou, China*, Vol. 01, pp. 58–63.

Dworkin, M.J. (2015). *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, FIPS PUB 202, National Institute of Standards and Technology, Gaithersburg, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

ENISA (2009). *Cloud Computing: Benefits, Risks and Recommendations for Information Security*, European Network and Information Security Agency, Athens, https://resilience.enisa.europa.eu/cloud-security-and-resilience/publications/cloud-computing-benefits-risks-and-recommendations-for-information-security.

Fernandez-Cerero, D., Jakóbik, A., Grzonka, D., Kołodziej, J. and Fernandez-Montes, A. (2018). Security supportive energy-aware scheduling and energy policies for cloud environments, *Journal of Parallel and Distributed Computing* **119**: 191–202.

Fernandez-Cerero, D., Fernandez-Montes, A., Jakóbik, A., Kołodziej, J. and Toro, M. (2018). SCORE: Simulator for cloud optimization of resources and energy consumption, *Simulation Modelling Practice and Theory* **82**: 160–173.

Gholipour, A., Araabi, B.N. and Lucas, C. (2006). Predicting chaotic time series using neural and neurofuzzy models: A comparative study, *Neural Processing Letters* **24**(3): 217–239.

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

Google (2016). Encryption at rest in Google Cloud, *Google Cloud Documentation*, https://cloud.google.com/security/encryption/default-encryption.

Grzonka, D. (2018). *Intelligent Agent-based Monitoring Systems of Task Scheduling for Distributed High-Performance Environments*, PhD thesis, Polish Academy of Sciences, Warsaw.

Grzonka, D., Jakóbik, A., Kołodziej, J. and Pllana, S. (2018). Using a multi-agent system and artificial intelligence for monitoring and improving the cloud performance and security, *Future Generation Computer Systems* **86**: 1106–1117.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*, 2nd Edn, Prentice Hall, Upper Saddle River.

Huang, W. and Wang, L. (2019). A hash function based on sponge structure with chaotic map for spinal codes, *2019 International Conference on Computer, Information and Telecommunication Systems (CITS), Beijing, China*, pp. 1–5.

ISO/IEC (2005). *Information Technology—Security Techniques—Code of Practice for Information Security Management*, ISO/IEC 27002:2005, International Organization for Standardization/International Electrotechnical Commission, Geneva, https://www.iso.org/standard/50297.html.

ISO/IEC (2015). *Information Technology—Security Techniques—Code of Practice for Information Security Controls Based on ISO/IEC 27002 for Cloud Services*, ISO/IEC 27017:2015, International Organization for Standardization/International Electrotechnical Commission, Geneva, https://www.iso.org/standard/43757.html.

Jakóbik, A. (2016). Big data security, *in* F. Pop *et al.* (Eds), *Resource Management for Big Data Platforms: Algorithms, Modelling, and High-Performance Computing Techniques*, Springer, Cham, pp. 241–261, DOI: 10.1007/978-3-319-44881-7_12.

Jakóbik, A., Grzonka, D. and Kołodziej, J. (2017a). Security supportive energy aware scheduling and scaling for cloud environments, *European Conference on Modelling and Simulation, ECMS 2017, Budapest, Hungary*, pp. 583–590.

Jakóbik, A., Grzonka, D. and Palmieri, F. (2017b). Non-deterministic security driven meta scheduler for distributed cloud organizations, *Simulation Modelling Practice and Theory* **76**: 67–81.

Jakóbik, A., Grzonka, D., Kołodziej, J. and González-Vélez, H. (2016). Towards secure non-deterministic meta-scheduling for clouds, *30th European Conference on Modelling and Simulation, ECMS 2016, Regensburg, Germany,* pp. 596–602, DOI: 10.7148/2016-0596.

Jakóbik, A. and Wilczynski, A. (2017). Using polymatrix extensive Stackelberg games in security-aware resource allocation and task scheduling in computational clouds, *Journal of Telecommunications and Information Technology* **1**: 71–80.

Jankowski, N. and Linowiecki, R. (2019). A fast neural network learning algorithm with approximate singular value decomposition, *International Journal of Applied Mathematics and Computer Science* **29**(3): 581–594, DOI: 10.2478/amcs-2019-0043.

Kaur, N., Saini, H.S. and Singh, R. (2009). Design and analysis of secure scheduler for MLS distributed database systems, *2009 IEEE International Advance Computing Conference, Patiala, India*, pp. 1400–1404.

Kidon, M. and Dobai, R. (2017). Evolutionary design of hash functions for IP address hashing using genetic programming, *IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain*, pp. 1720–1727.

Kołodziej, J. (2012). *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*, Springer, Berlin.

Yee, L.P. and De Silva, L.C. (2002). Application of multilayer perceptron network as a one-way hash function, *International Joint Conference on Neural Networks. IJCNN'02, Honolulu, USA*, Vol. 2, pp. 1459–1462.

Mahjoob, M.J., Abdollahzade, M., Zarringhalam, R. and Kalhor, A. (2008). Chaotic time series forecasting using locally quadratic fuzzy neural models, *9th WSEAS International Conference on Fuzzy Systems, Sofia, Bulgaria*, pp. 26–32.

Menezes, A.J., Vanstone, S.A. and Oorschot, P.C.V. (1996). *Handbook of Applied Cryptography*, CRC Press, Boca Raton.

Merkle, R.C. (1979). *Secrecy, Authentication, and Public Key Systems*, Stanford University, Stanford, pp. 11–16.

Mohan, S., Yoon, M.K., Pellizzoni, R. and Bobba, R. (2014). Real-time systems security through scheduler constraints, *26th Euromicro Conference on Real-Time Systems, Madrid, Spain*, pp. 129–140.

NIST (2015). *Secure Hash Standard (SHS)*, FIPS PUB 180-4, National Institute of Standards and Technology, Gaithersburg, `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`.

NIST (2019). *Status Report on the First Round of the NIST Lightweight Cryptography Standardization Process*, NISTIR 8268, National Institute of Standards and Technology, Gaithersburg, `https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8268.pdf`.

Podolskiy, V., Jindal, A. and Gerndt, M. (2019). Multilayered autoscaling performance evaluation: Can virtual machines and containers co-scale?, *International Journal of Applied Mathematics and Computer Science* **29**(2): 227–244, DOI: 10.2478/amcs-2019-0017.

Rajeshwaran, K. and Anil Kumar, K. (2019). Cellular automata based hashing algorithm (CABHA) for strong cryptographic hash function, *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India*, pp. 1–6.

Saleh, M. and Dong, L. (2013). Real-time scheduling with security enhancement for packet switched networks, *IEEE Transactions on Network and Service Management* **10**(3): 271–285.

Schneier, B. (1995). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Wiley, New York.

Singh, M. and Garg, D. (2009). Choosing best hashing-strategies and hash functions, *2009 IEEE International Advance Computing Conference, Patiala, India*, pp. 50–55.

Tadokoro, H., Kourai, K. and Chiba, S. (2010). A secure system-wide process scheduler across virtual machines, *2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, Tokyo, Japan*, pp. 27–36.

Tchórzewski, J. and Jakóbik, A. (2019). Theoretical and experimental analysis of cryptographic hash functions, *Journal of Telecommunications and Information Technology* **1**: 125–133.

Tchórzewski, J., Jakóbik, A. and Grzonka, D. (2019). Towards ANN-based scalable hashing algorithm for secure task processing in computational clouds, *33rd International ECMS Conference on Modelling and Simulation, Caserta, Italy*, pp. 421–427.

Turcanik, M. (2017a). Hash function generation based on neural networks and chaotic maps, *Communication and Information Technologies (KIT), Vysoke Tatry, Slovakia*, pp. 1–5.

Turcanik, M. (2017b). Using recurrent neural network for hash function generation, *International Conference on Applied Electronics (AE), Pilsen, Chech Republic*, pp. 1–4.

Wang, M. and Li, Y. (2015). Hash function with variable output length, *International Conference on Network and Information Systems for Computers, Wuhan, China*, pp. 190–193.

Yang, Q., Gao, T., Fan, L. and Gu, Q. (2009). Analysis of one-way alterable length hash function based on cell neural network, *5th International Conference on Information Assurance and Security, Xi'an, China*, Vol. 1, pp. 391–395.

Jacek Tchórzewski received his BSc and MSc degrees 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 a PhD student at the AGH University of Science and Technology, Poland.

Agnieszka Jakóbik received her MSc in stochastic processes at the Jagiellonian University, Poland, and her PhD in artificial neural networks. Since 2009 she has been an assistant professor at the Cracow University of Technology, Poland.

Mauro Iacono received his MSc in degree computer science at the University of Naples "Federico II" and his PhD degree in electronics the Second University of Naples. He is an associate professor of computing systems at the University of "L. Vanvitelli", Caserta, Italy, where he leads the computer science section of the data and computer science research group. He is currently the president of the European Council for Modelling and Simulation.

Received: 31 January 2021
Revised: 28 May 2021
Re-revised: 30 July 2021
Accepted: 10 August 2021