Privacy-preserving Linear Computations in Spiking Neural P Systems

Mihail-Iulian Plesa

University of Bucharest
Bucharest, Romania

Department of Computer Science

mihail-iulian.plesa@s.unibuc.ro

Marian Gheorghe

University of Bradford
Bradford, UK

School of Electrical Engineering and Computer Science

M.Gheorghe@bradford.ac.uk

Florentin Ipate

University of Bucharest
Bucharest, Romania

Department of Computer Science

florentin.ipate@unibuc.ro

Spiking Neural P systems are a class of membrane computing models inspired directly by biological neurons. Besides the theoretical progress made in this new computational model, there are also numerous applications of P systems in fields like formal verification, artificial intelligence, or cryptography. Motivated by all the use cases of SN P systems, in this paper, we present a new privacy-preserving protocol that enables a client to compute a linear function using an SN P system hosted on a remote server. Our protocol allows the client to use the server to evaluate functions of the form $t_1 k + t_2$ without revealing t_1, t_2 or k and without the server knowing the result. We also present an SN P system to implement any linear function over natural numbers and some security considerations of our protocol in the honest-but-curious security model.

1 Introduction

Membrane computing (or P systems) is a new model of computation inspired by how membranes work and interact in living cells [17]. There are several variants of the model e.g. neural P systems, cell P systems, tissue P systems, etc., [29, 15, 12]. P systems have generated new perspectives on the P vs NP problem, being used to efficiently solve hard problems [27, 3, 7, 28]. There are also multiple applications of P systems in various fields like formal verification, artificial intelligence, or cryptography [30].

In this work we used a special type of P systems called Spiking Neural P systems (SN P systems for short) [12]. SN P systems are inspired by biological neurons. There are also numerous variants of SN P systems: SN P systems with astrocytes, SN P systems with communication on request, SN P systems with polarization, SN P systems with colored spikes, etc., [16, 14, 25, 23].

Although there are many theoretical aspects and simulations in the literature, to gain the maximum efficiency of these systems, they must be implemented on dedicated hardware [1]. If these systems are implemented at a large scale, they will have to be accessed remotely in the cloud. This raises privacy concerns about data uploaded to the server that hosts the P system. This paper approaches the problem of confidentiality in SN P systems by describing a protocol that allows a client to perform a simple linear computation using an SN P system that is served remotely without revealing private information.

1.1 Related work

Besides the theoretical work, there are also many applications of P systems. In [22] the authors propose a new key agreement protocol based on SN P systems. In [8, 10, 11] the authors describe how to implement the RSA algorithm in the framework of membrane computing. One ingenious way of applying P systems is shown in [24] which presents an algorithm to break the RSA encryption. There are also applications in artificial intelligence. In [2] the authors present a survey of the learning aspects in SN P systems. Clustering algorithms have also been developed in the framework of membrane computing [21, 20, 19]. Image processing is another common application of P systems [4, 26, 5].

1.2 Our contribution

In this paper, we present a protocol that allows a client to perform a linear computation using an SN P system hosted on a server without revealing any private data. The SN P system computes functions of the form $t_1 k + t_2$ over natural numbers. The client must retrieve from the server the result of the computation without the server knowing t_1, t_2 or k. Also, the server must not learn the value $t_1 k + t_2$. To enable privacy-preserving computations on the server side, we use the ElGamal cryptosystem and its homomorphic properties [6]. We also provide an SN P system that computes any linear function over natural numbers and some security considerations of our protocol. The paper is organized as follows: in Section 2 we present the background on the SN P system and homomorphic encryption. In Section 3, we show an SN P system that computes linear functions over the natural numbers. In Section 4, we introduce our protocol and some security considerations. Section 5 is left for the conclusions and further directions.

2 Preliminaries

In this section, we briefly present the Spiking Neural P systems (SN P systems) and the cryptographic algorithm used in our protocol. We stress some useful properties of the encryption scheme.

2.1 Spiking Neural P systems

A Spiking Neural P system (SN P system) of degree $m \geq 1$ is defined as the following construct:

Definition 2.1. $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, i_0)$ where:

- $O = \{a\}$ is the alphabet. The symbol a denotes a spike.

- σ_i, $1 \leq i \leq m$ represents a neuron. Each neuron is characterized by the initial number of spikes denoted by $n_i \geq 0$ and the finite set of rules denoted by R_i: $\sigma_i = (n_i, R_i)$.

- Each rule can be of the following two forms:
    1. $E/a^r \rightarrow a; t$ where E is a regular expression over the alphabet O, $r \in \mathbf{N}^*$ represents the current number of spikes in the neuron and $t \geq 0$ is the refractory period. This type of rule is called a firing rule.
    2. $a^s \rightarrow \lambda$ where $s \geq 1$ is the current number of spikes in the neuron and λ is a special symbol that denotes an empty set of spikes. This type of rule is called a forgetting rule.

- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ is the set of synapses between neurons. No neuron can have a synapse to it i.e. $(i, i) \notin syn \ \forall \ 1 \leq i \leq m$.

- i_0 represents the output neuron.

A neuron can fire using the firing rule $E/a^r \to a;t$ only if it contains n spikes such that $a^n \in L(E)$ and $n \geq r$ where $L(E)$ is a language defined in the following way:

- $L(\lambda) = \{\lambda\}$

- $L(a) = \{a\} \ \forall a \in O$

- $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$

- $L((E_1)(E_2)) = L(E_1)L(E_2)$

- $L((E_1)^+) = L(E_1)^+$

for all regular expressions over the alphabet O.

After firing, r spikes are consumed. A firing rule that is applied when the neuron contains exactly r spikes i.e. $L(E) = \{a^r\}$, is simply denoted as $a^r \to a;t$.

At the neuron level, all rules are applied sequentially, but the system as a whole evolves with maximum parallelism i.e. if a rule can be applied in a neuron then that rule will be applied.

At a certain point, a neuron can be firing, spiking, or closed. If a neuron applies the firing rule $E/a^r \to a;t$ at moment q then the neuron will send a spike to all the neurons to which it is connected by synapses at moment $q+t$. At times $q+1, q+2, \ldots q+t-1$ the neuron will be in the refractory period i.e. the neuron will not receive or send any spikes. When neuron σ_i is spiking, the spikes are replicated in such a way that each neuron σ_j with $(i,j) \in syn$ receives one spike although the number of spikes consumed by σ_i is exactly r.

When a forgetting rule $a^s \to \lambda$ is applied in a neuron, s spikes are removed from that neuron. A neuron can apply a forgetting rule only if the number of spikes is exactly s.

There are several ways in which we can record the output of an SN P system:

- The moments of time at which the output neuron i_0 sends a spike i.e. if the neuron i_0 releases spikes at the moments $q_1, q_2 \ldots$ then the output of Π is the sequence $q_1, q_2 \ldots$.

- The interval between the moments at which the output neuron i_0 sends a spike i.e. if the neuron i_0 releases spikes at the moments $q_1, q_2 \ldots$ then the output of Π is the sequence $q_2 - q_1, q_3 - q_2, \ldots$

An SN P system is constructed using the principle of minimal determinism i.e. at a certain moment in time, either a firing or a forgetting rule is applied without being able to choose which of the two types of rules is applied [12].

2.2   Homomorphic encryption

In this work, we use the ElGamal cryptosystem [6]. The security of the encryption scheme is based on the computational Diffie-Hellman assumption (CDH). Moreover, the scheme achieves semantic security based on the decisional Diffie-Hellman assumption (DDH) i.e. the scheme is randomized. Randomization implies that when encrypting the same message multiple times, each resulting ciphertext will be different. A consequence of this property is the fact that an attacker cannot distinguish two plaintexts by analyzing the corresponding ciphertext with non-negligible probability. The scheme works over a group G of order q with a generator g. We now proceed to the description of the cryptosystem:

- The key generation algorithm denoted by **KeyGen** generates a key pair i.e. a private key and the corresponding public key. The algorithm takes the following steps:

1. Generate a random integer $x \in \{1, 2, \ldots q-1\}$.
2. Compute $h := g^x$.
3. Output the public key h and the corresponding private key x.

- The encryption algorithm denoted by $\mathbf{Enc}_h^y$ encrypts a plaintext $m \in G$ using the public key h and a random number $y \in \{1, 2, \ldots, q-1\}$. The algorithm performs the following steps:

  1. Computes $s := h^y$.
  2. Computes $c_1 := g^y$ and $c_2 := m \cdot s$.
  3. Outputs the ciphertext $c := (c_1, c_2)$.

- The decryption algorithm denoted by $\mathbf{Dec}_x$ takes as input a ciphertext $c = (c_1, c_2)$ and decrypts it under the private key x. The algorithm is composed of the following steps:

  1. Computes $s := c_1^x$.
  2. Computes s^{-1}, the inverse of s in the group G.
  3. Computes the plaintext $m := c_2 \cdot s^{-1}$.
  4. Outputs the plaintext m.

The proof of correctness is straightforward:

$$c_2 \cdot s^{-1} = c_2 \cdot c_1^{-x} = m \cdot h^y \cdot g^{-xy} = m \cdot g^{-xy} \cdot g^{xy} = m \tag{1}$$

The encryption of a message $m \in G$ can be summarized by the following two equations:

$$c_1 = g^y \tag{2}$$

$$c_2 = m \cdot h^y \tag{3}$$

The scheme is homomorphic with respect to multiplication. Let $c = (c_1, c_2)$ and $c' = (c_1', c_2')$ be the encryptions of two plaintexts m and m' under the same public key h i.e. $c = \mathbf{Enc}_h^y(m)$ and $c' = \mathbf{Enc}_h^y(m')$. We define the following two operations:

1. Let $c \odot c' = (c_1 \cdot c_1', c_2 \cdot c_2')$ be the multiplication of two ciphertexts. The result of this operation is another ciphertext that encrypts the sum between m and m':

$$c_1 \cdot c_1' = g^y \cdot g^{y'} = g^{y+y'} \tag{4}$$

$$c_2 \cdot c_2' = m \cdot s \cdot m' \cdot s' = m \cdot m' \cdot h^y \cdot h^{y'} = m \cdot m' \cdot h^{y+y'} \tag{5}$$

From 4 and 5 we can see that $c \odot c' = (c_1 \cdot c_1', c_2 \cdot c_2')$ is a ciphertext that encrypts $m \cdot m'$ under the public key h.

2. Let $c \otimes k = (c_1, c_2 \cdot k)$ be the multiplication between a ciphertext and a constant k. The result of this operation is a ciphertext that encrypts the product between m and k as it can be seen from 6 and 7:

$$c_1 = g^y \tag{6}$$

$$c_2 \cdot k = m \cdot s \cdot k = (m \cdot k) \cdot s = (m \cdot k) h^y \tag{7}$$

3. When $c_1 = c'_1$ i.e. $y = y'$, we can also define the addition between two ciphertexts as follows: $c \oplus c' = (c_1, c_2 + c'_2)$. The result of this operation is a ciphertext that encrypts the sum between m and m':

$$c_1 = g^y \tag{8}$$

$$c_2 + c'_2 = m \cdot s + m' \cdot s = (m + m') \cdot s = (m + m') \cdot h^y \tag{9}$$

It is important to notice that when $y = y'$ the scheme is no longer semantically secure. Although an attacker who observes the two ciphertexts c and c' could not recover any of the plaintexts, it could determine additional information about them e.g. whether they are different. In many scenarios, this is not acceptable but in this work, we will use the $\oplus$ operation. All the messages encrypted with the same random y are not critical i.e. the impact of the lack of semantic security does not affect the security of the protocol in which the encryption scheme is used.

3   SN P system to compute linear functions

In this section, we describe an SN P system that computes functions of the form $t_1 k + t_2$ over natural numbers.
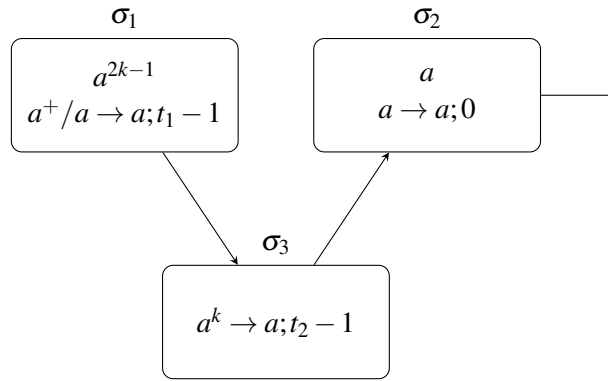

Figure 1: SN P system to compute linear functions

Let $\Pi_{Add}(t_1, t_2, k) = \{\{a\}, \sigma_1, \sigma_2, \sigma_3, syn_{add}, \sigma_2\}$ be the SN P system that computes the linear function $t_1 k + t_2$, $t_1, t_2 \in \mathbf{N}$ with the following components:

- The alphabet is made from a single symbol $\{a\}$ that denotes a spike.

- There are three neurons: σ_1, σ_2 and σ_3 with the following firing rules:
    1. For σ_1 the firing rule is $a^+/a \to a; t_1 - 1$.
    2. For σ_2 the firing rule is $a \to a; 0$.
    3. For σ_3 the firing rule is $a^k \to a; t_2 - 1$.

- The set of synapses syn_{add} is the set $\{(1, 3), (3, 2)\}$.

- The output neuron is σ_2.

Initially σ_1 has $2k - 1$ spikes, σ_2 has 1 spike and σ_3 has no spikes. Since σ_2 has no refractory time and one spike, it will release it in the first step of the computation. During the first step, σ_1 will be firing. Since its refractory period is $t_1 - 1$ during the time steps $2, 3, \ldots t_1 - 1$ the neuron will be closed i.e. it

will not receive or send any spikes. In step t_1 the neuron will send one spike to σ_3 and fire again. Thus σ_1 fires every t_1 steps consuming one spike and sending one spike to σ_3. σ_3 will fire when it acumulates k spikes from neuron σ_1. Since σ_1 fires one spikes every t_1 steps, at time step, $k \cdot t_1$ σ_3 will receive the kth spike. At the moment $t_1 \cdot k + 1$, σ_3 will fire. The refractory period of this neuron is $t_2 - 1$ thus at the steps $t_1 \cdot k + 2, t_1 \cdot k + 3, \ldots, t_1 \cdot k + t_2 - 1$ it will be closed and it will release one spike to σ_2 at $t_1 \cdot k + t_2$. Since σ_3 has no refractory period, it will release the spike at the moment $t_1 \cdot k + t_2 + 1$. At this point, the number of spikes left in σ_1 is $k - 1$ because it already sent k spikes to neuron σ_3 and the initial number of spikes was $2k - 1$. The neuron will continue to send spikes to σ_3 at the appropriate time steps until the spikes run out. The neuron σ_3 will never fire again since it can no longer accumulate k spikes. σ_2 will never fire again either since it will no longer receive the spike from σ_3. Thus, after σ_1 exhausts all the spikes, the computation will stop. There are two moments when the output neuron σ_2 fires: 1 and $t_1 \cdot k + t_2 + 1$. Thus, the result of the computation i.e. the difference between the time points at which the output neuron fires, is $t_1 \cdot k + t_2$. The $\Pi_{Add}(t_1, t_2, k)$ system is depicted in Figure 1.

4   Privacy-preserving computations in SN P systems

In this section, we describe our protocol which enables the running of an SN P system to compute linear functions in a privacy-preserving way. We also make some remarks regarding security.

4.1   The protocol

There are two actors in the protocol:

1. The Server: it can instantiate and run an SN P system of the form $\Pi_{Add}(t_1, t_2, k)$ for any integers t_1, t_2 and k.

2. The Client: it wants to evaluate the linear function $t_1 \cdot k + t_2$ using the system hosted by the Server without revealing any of inputs t_1, t_2 or k.

The protocol uses the holomorphic properties of the ElGamal cryptosystem to allow the client to use the server without revealing the inputs of the SN P system. There are 7 steps:

1. The Client will use the **KeyGen** algorithm to generate a key pair: h and x.

2. The Client will use the encryption algorithm $\mathbf{Enc}_h^y$ to encrypt t_1, t_2 and k. We denote by $c^{t_1} = \left(c_1^{t_1}, c_2^{t_1} \right), c^{t_2} = \left(c_1^{t_2}, c_2^{t_2} \right)$ and $c^k = \left(c_1^{t_k}, c_2^k \right)$ the encryptions of t_1, t_2 and k:
   - $c^{t_1} = \left(c_1^{t_1}, c_2^{t_1} \right) = \mathbf{Enc}_h^{y_1}(t_1)$
   - $c^k = \left(c_1^k, c_2^k \right) = \mathbf{Enc}_h^{y_2}(k)$
   - $c^{t_2} = \left(c_1^{t_2}, c_2^{t_2} \right) = \mathbf{Enc}_h^{y_1 + y_2}(t_2)$

3. The Client will store locally $c_1^{t_1}, c_1^{t_2}$ and c_1^k and send to the server $c_2^{t_1}, c_2^{t_2}$ and c_2^k.

4. The Server will instantiate and run an SN P system of the form $\Pi_{Add}\left(c_2^{t_1}, c_2^{t_2}, c_2^k \right)$.

5. After the computation stops, the Server will return to the client the result of the computation i.e. $c_2 = c_2^{t_1} \cdot c_2^k + c_2^{t_2}$.

6. The Client will compose a new ciphertext $c = (c_1, c_2)$ where $c_1 = c_1^{t_1} \cdot c_1^k$.

7. The Client will decrypt the ciphertext c using the algorithm $\mathbf{Dec}_x$ and retrived the result of the computation i.e. $t_1 \cdot k + t_2$.

We now prove that the ciphertext computed in step 6 of the protocol is a valid ElGamal ciphertext that correctly decrypts to the final result of the computation i.e. $t_1 \cdot k + t_2$. Let c' be the following ciphertext:

$$c' = \left(c_1, c_2^{t_1} \cdot c_2^k\right) \tag{10}$$

Since $c_1 = c_1^{t_1} \cdot c_1^k$ we can write c' as:

$$c' = \left(c_1^{t_1} \cdot c_1^k, c_2^{t_1} \cdot c_2^k\right) \tag{11}$$

The ciphertext c' represents the multiplication between the ciphertexts c^{t_1} and c^k:

$$c' = c^{t_1} \odot c^k \tag{12}$$

Since c and c' use the same randomess i.e. c_1, we can write c as the sum between the ciphertext c' and c^{t_2}:

$$c = c' \oplus c^{t_2} \tag{13}$$

In conclusion, we can express c as a composition of valid ElGamal ciphertexts which is also a valid ElGamal ciphertext:

$$c = \left(c^{t_1} \odot c^k\right) \oplus c^{t_2} \tag{14}$$

The ciphertext $\left(c^{t_1} \odot c^k\right)$ represents the encryption of $t_1 \cdot k$. When we add this ciphertext with c^{t_2} using the $\oplus$ operation, the resulting ciphertext will be the encryption of $t_1 \cdot k + t_2$ which is the result of the computation performed over plaintext data. Figure 2 depicts our protocol.

Since the SN P system works over natural numbers, we can use $G = \mathbf{Z}_q$ for a large prime q.

4.2 Security considerations

We analyze our protocol in the honest-but-curious security model [18]. In this model, we assume that the adversary is the Server. There are two properties of the adversary:

1. Curious: the Server will try to find information about the underlying plaintexts. In our case, these are the parameters of the SN P system: t_1, t_2, and k.

2. Honest: the Server will respect the protocol and it will complete every step. It will not modify in any way the messages received or sent to the Client.

The underlying encryption scheme i.e. the ElGamal cryptosystem is semantically secure given the DDH assumption as long as each ciphertext is generated using different randomness [9]. The security of the scheme can be illustrated using the following game. We suppose that the attacker runs in probabilistic polynomial time and has access to an encryption oracle that receives as input a plaintext and returns the corresponding ciphertext:

- The attacker chooses as many plaintexts as it wants and encrypts them using the oracle. For each ciphertext, the attacker knows the corresponding plaintext.

- The attacker chooses two plaintext m_0 and m_1 and send them to the oracle.

- The oracle will generate a random bit b and return the encryption of m_b.

- The attacker outputs the bit b.

The scheme is secure as long as the attacker cannot output the correct bit with non-negligible probability.

Given the fact that the Client encrypts each parameter of the SN P system using the ElGamal cryptosystem with different randomness, the Server cannot learn any information about them with non-negligible probability.
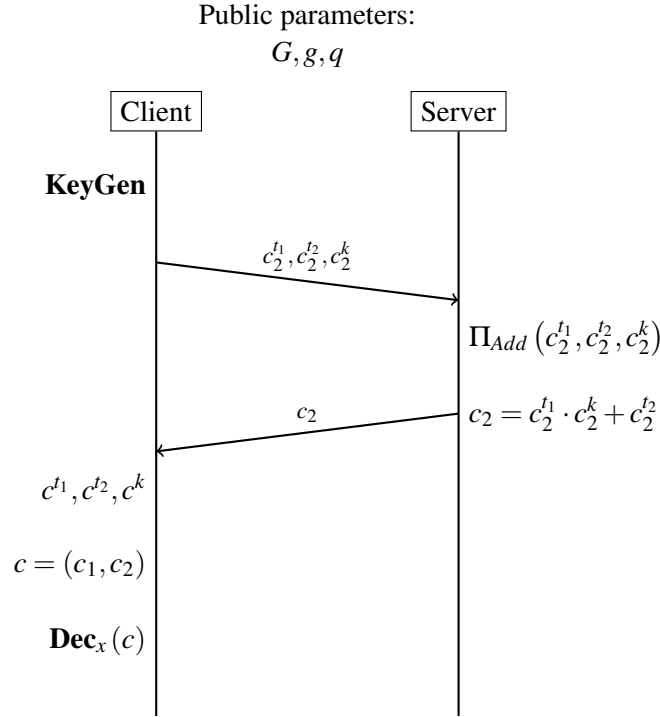
Public parameters:
$$G, g, q$$

Client	Server

KeyGen

$c_2^{t_1}, c_2^{t_2}, c_2^{k}$

$\Pi_{Add}\left(c_2^{t_1}, c_2^{t_2}, c_2^{k}\right)$

c_2

$c_2 = c_2^{t_1} \cdot c_2^{k} + c_2^{t_2}$

c^{t_1}, c^{t_2}, c^{k}

$c = (c_1, c_2)$

$\mathbf{Dec}_x(c)$

Figure 2: Privacy-preserving linear function computation using SN P systems

5 Conclusions and further directions of research

In this paper, we presented a protocol for performing privacy-preserving computation over SN P systems. There are two actors involved: the client and the server. The server hosts an SN P system that computes linear function over natural numbers i.e. $t_1 k + t_2$. The client uses the protocol to retrieve the result of the computation without revealing t_1, t_2 or k and without the server knowing the result of the calculation. We presented an SN P system that computes any linear function over natural numbers and also some security considerations about our protocol which is based on the ElGamal cryptosystem.

There are several directions of research. The first one is to give formal proof of the security of the protocol. Although the protocol is secure at first sight, we must prove it by reducing the security of it to the security of the underlying cryptosystem. The second direction of research is to enable complex computation on SN P systems in a privacy-preserving way. The third direction is to use dedicated cryptosystems e.g. fully homomorphic encryption schemes which are created to enable privacy-preserving computations [13]. The challenge here is to map the operations performed over encrypted data to the operations performed by the SN P system. We should also consider using other privacy-enhancing technologies e.g. secure multi-party computation or differential privacy to enable private computations over SN P systems. Another direction of research is to implement the protocol and perform the appropriate benchmarking to study its efficiency and communication overhead in practice.

5.0.1 Acknowledgements

References

[1] Alberto Arteta Albert, Ernesto Díaz-Flores, Luis Fernando de Mingo López & Nuria Gómez Blas (2021): *An in vivo proposal of cell computing inspired by membrane computing*. Processes 9(3), p. 511, doi:10.3390/pr9030511.

[2] Yunhui Chen, Ying Chen, Gexiang Zhang, Prithwineel Paul, Tianbao Wu, Xihai Zhang, Haina Rong & Xiaomin Ma (2021): *A Survey of Learning Spiking Neural P Systems and A Novel Instance*. International Journal of Unconventional Computing 16.

[3] Erzsébet Csuhaj-Varjú, Marian Gheorghe, Alberto Leporati, Miguel Ángel Martínez-del Amor, Linqiang Pan, Prithwineel Paul, Andrei Păun, Ignacio Pérez-Hurtado, Mario J Pérez-Jiménez, Bosheng Song et al. (2022): *Membrane computing concepts, theoretical developments and applications*. In: *Handbook of Unconventional Computing: VOLUME 1: Theory*, World Scientific, pp. 261–339, doi:10.1142/9789811235726_0008.

[4] Daniel Díaz-Pernil, Miguel A Gutiérrez-Naranjo & Hong Peng (2019): *Membrane computing and image processing: a short survey*. Journal of Membrane Computing 1, pp. 58–73, doi:10.1007/s41965-018-00002-x.

[5] Daniel Díaz-Pernil, Francisco Pena-Cantillana & Miguel A Gutiérrez-Naranjo (2013): *A parallel algorithm for skeletonizing images by using spiking neural P systems*. Neurocomputing 115, pp. 81–91, doi:10.1016/j.neucom.2012.12.032.

[6] Taher ElGamal (1985): *A public key cryptosystem and a signature scheme based on discrete logarithms*. IEEE transactions on information theory 31(4), pp. 469–472, doi:10.1109/TIT.1985.1057074.

[7] Songhai Fan, Yiyu Gong, Gexiang Zhang, Yun Xiao, Haina Rong, Prithwineel Paul, Xiaomin Ma, Han Huang & Marian Gheorghe (2021): *Implementation of Kernel P Systems in CUDA for Solving NP-hard Problems*. International Journal of Unconventional Computing 16.

[8] Ganbat Ganbaatar, Dugar Nyamdorj, Gordon Cichon & Tseren-Onolt Ishdorj (2021): *Implementation of RSA cryptographic algorithm using SN P systems based on HP/LP neurons*. Journal of Membrane Computing 3, pp. 22–34, doi:10.1007/s41965-021-00073-3.

[9] Oded Goldreich (2004): *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge, doi:10.1017/CBO9780511721656.

[10] Ping Guo & Wei Xu (2016): *A family P system of realizing RSA algorithm*. In: *Bio-inspired Computing–Theories and Applications: 11th International Conference, BIC-TA 2016, Xi'an, China, October 28-30, 2016, Revised Selected Papers, Part I 11*, Springer, pp. 155–167, doi:10.1007/978-981-10-3611-8_16.

[11] Ping Guo & Wei Xu (2017): *Implementation of RSA algorithm based on P system*. Journal of Computational and Theoretical Nanoscience 14(9), pp. 4227–4235, doi:10.1166/jctn.2017.6723.

[12] Mihai Ionescu, Gheorghe Păun & Takashi Yokomori (2006): *Spiking neural P systems*. Fundamenta informaticae 71(2-3), pp. 279–308.

[13] Paulo Martins, Leonel Sousa & Artur Mariano (2017): *A survey on fully homomorphic encryption: An engineering perspective*. ACM Computing Surveys (CSUR) 50(6), pp. 1–33, doi:10.1145/3124441.

[14] Linqiang Pan, Gheorghe Păun, Gexiang Zhang & Ferrante Neri (2017): *Spiking neural P systems with communication on request*. International journal of neural systems 27(08), p. 1750042, doi:10.1142/S0129065717500423.

[15] Linqiang Pan & Mario J Pérez-Jiménez (2010): *Computational complexity of tissue-like P systems*. Journal of Complexity 26(3), pp. 296–315, doi:10.1016/j.jco.2010.03.001.

[16] Linqiang Pan, Jun Wang & Hendrik Jan Hoogeboom (2012): *Spiking neural P systems with astrocytes*. Neural Computation 24(3), pp. 805–825, doi:10.1162/NECO_a_00238.

[17] Gheorghe Păun (2000): *Computing with membranes*. Journal of Computer and System Sciences 61(1), pp. 108–143, doi:10.1006/jcss.1999.1693.

[18] Andrew Paverd, Andrew Martin & Ian Brown (2014): *Modelling and automatically analysing privacy properties for honest-but-curious adversaries. Tech. Rep.*

[19] Hong Peng, Xiaohui Luo, Zhisheng Gao, Jun Wang, Zheng Pei et al. (2015): *A novel clustering algorithm inspired by membrane computing. The Scientific World Journal* 2015, doi:10.1155/2015/929471.

[20] Hong Peng, Jun Wang, Mario J Pérez-Jiménez & Agustín Riscos-Núñez (2015): *An unsupervised learning algorithm for membrane computing. Information Sciences* 304, pp. 80–91, doi:10.1016/j.ins.2015.01.019.

[21] Hong Peng, Jun Wang, Peng Shi, Agustín Riscos-Núñez & Mario J Pérez-Jiménez (2015): *An automatic clustering algorithm inspired by membrane computing. Pattern Recognition Letters* 68, pp. 34–40, doi:10.1016/j.patrec.2015.08.008.

[22] Mihail-Iulian Plesa, Marian Gheoghe, Florentin Ipate & Gexiang Zhang (2022): *A key agreement protocol based on spiking neural P systems with anti-spikes. Journal of Membrane Computing* 4(4), pp. 341–351, doi:10.1007/s41965-022-00110-9.

[23] Tao Song, Alfonso Rodríguez-Patón, Pan Zheng & Xiangxiang Zeng (2017): *Spiking neural P systems with colored spikes. IEEE Transactions on Cognitive and Developmental Systems* 10(4), pp. 1106–1115, doi:10.1109/TCDS.2017.2785332.

[24] Răzvan Vasile, Marian Gheorghe & Ionuț Mihai Niculescu (2023): *Breaking RSA Encryption Protocol with Kernel P Systems.* doi:10.21203/rs.3.rs-2684530/v1.

[25] Tingfang Wu, Andrei Păun, Zhiqiang Zhang & Linqiang Pan (2017): *Spiking neural P systems with polarizations. IEEE transactions on neural networks and learning systems* 29(8), pp. 3349–3360, doi:10.1109/TNNLS.2017.2726119.

[26] Rafaa I Yahya, Siti Mariyam Shamsuddin, Salah I Yahya, Shafatnnur Hasan, Bisan Al-Salibi & Ghada Al-Khafaji (2016): *Image segmentation using membrane computing: a literature survey.* In: *Bio-inspired Computing–Theories and Applications: 11th International Conference, BIC-TA 2016, Xi'an, China, October 28-30, 2016, Revised Selected Papers, Part I 11*, Springer, pp. 314–335, doi:10.1007/978-981-10-3611-8_26.

[27] Claudio Zandron, Claudio Ferretti & Giancarlo Mauri (2001): *Solving NP-complete problems using P systems with active membranes.* In: *Unconventional Models of Computation, UMC'2K: Proceedings of the Second International Conference on Unconventional Models of Computation,(UMC'2K)*, Springer, pp. 289–301, doi:10.1007/978-1-4471-0313-4_21.

[28] Ge-Xiang Zhang, Marian Gheorghe & Chao-Zhong Wu (2008): *A quantum-inspired evolutionary algorithm based on P systems for knapsack problem. Fundamenta Informaticae* 87(1), pp. 93–116.

[29] Ge-Xiang Zhang & Lin-Qiang Pan (2010): *A survey of membrane computing as a new branch of natural computing. Chinese journal of computers* 33(2), pp. 208–214, doi:10.3724/SP.J.1016.2010.00208.

[30] Gexiang Zhang, Mario J Pérez-Jiménez & Marian Gheorghe (2017): *Real-life applications with membrane computing.* 25, Springer, doi:10.1007/978-3-319-55989-6.