Breaking RSA Encryption Protocol with Kernel P Systems

Răzvan Vasile[1*], Marian Gheorghe[1,2] and Ionuţ Mihai Niculescu[1]

[1*]Department of Computer Science, University of Bucharest, Bucharest, Bucharest, Romania.

[2]Department of Computer Science, University of Bradford, Bradford, West Yorkshire, UK.

*Corresponding author(s). E-mail(s): razvan.vasile@yahoo.com;

Contributing authors: m.gheorghe@bradford.ac.uk; ionutmihainiculescu@gmail.com;

Abstract

The *prime factorisation* problem is intractable, i.e., no efficient algorithm is known. In cryptography there are some well-known approaches based on the computational hardness of this problem, including Rivest-Shamir-Adleman (RSA) encryption protocol. Several attemps to break RSA have been investigated, some of them based on the massive parallelism of membrane systems. In this paper a new approach, based on kernel P system formalism, aimed at reassessing the space-time trade-off, usually involved in membrane computing solutions, is investigated. Two models are introduced and assessed in order to find the potential benefits of each of them for solving the above problem, revealing also their limitations and providing hints for further improvements.

Keywords: RSA encryption protocol, Breaking RSA, Integer Factorisation, Membrane Computing, kernel P Systems, kP-Lingua

1 Introduction

Rivest-Shamir-Adleman (RSA) encryption is *de facto* the standard encryption protocol for public key encryption systems. An in-depth study of RSA cryptosystem is provided in the comprehensive survey [1]. The challenge of breaking RSA is to solve the *integer factorisation problem*. There have been countless attempts to break RSA encryption, but the algorithm still stands. Some successes have been made on a few particular cases [2, 3], but the encryption, as a whole, still stands unbroken [4].

Membrane Computing was created by Gheorghe Păun [5] and the models are known as membrane systems or P systems. Membrane computing has been intensively studied. Some of the most recent papers refer to theoretical developments [6–11] and applications [12–16]. The possibility of generating exponential space in polynomial (more often linear) time, is a key feature of these models and represents the main element in solving **NP**-complete problems [17–19]. A few solutions have been proposed, based on this feature, for the RSA encryption problem as well, with various degrees of success.

The factorisation solution relies on arithmetic operations and data representation that may differ, depending on specific models and implementations. Arithmetic operations using membrane computing models have been investigated for different encodings (unary and binary represenations) [20–23]. Several solutions for arithmetic operations, including the *tinteger factorisation problem*, were proposed using variations of *spiking neural P systems* [24–28]. In article [29] the authors offer a solution to the integer factorisation problem, using *polarization P systems with active membranes*. In [30] it is provided a solution to the integer factorisation problem based on *polarizationless P systems*, to generate all pairs of numbers P and Q, using partial functions from $\mathbb{N}$ to $\mathbb{N}^2$ such that their product is equal to N. In [31] an *asynchronous P system* model is used in order to provide a solution to the factorisation problem. This allows to study both sequential and parallel solutions, and their complexity aspects. The paper [32] also uses a variation of spiking neural P systems, namely *SN P systems based on HP/LP neurons*. The authors use *HP/LP neurons* to build a Dual-rail type logic. This includes logical operators such as NOT, XOR, AND, OR. Based on this logic, the authors further build addition, multiplication and modulo modules. A conceptually different approach is found in the articles [33], [34], where pairs of random numbers (P, Q) are generated, and it is checked if their product is equal to N. In [35] a new formalism, called *Virus Machine*, is introduced to build arithmetic operations necessary in cryptography. The encoding is unary.

In this paper the space-time tradeoff is reassessed from a slightly different perspective. Most of the previous papers, in order to obtain tractable solutions, exploit maximally the use of space, ending up with a polynomial space complexity ($O(N^2)$, where N is the number to be decomposed) and the number of pairs of integers is N^2. In this paper it is studied the possibility of

decreasing the use of space. In this case, the time to find the factorisation will increase. Two models are investigated, whereby the encodings of integers are either unary or binary. The main contribution of the paper is a new solution to the factorisation problem for the above mentioned encodings, where $N * (N/2 + 1)/4$ pairs are generated instead of N^2, the space compexity is $O(\sqrt{N})$, but the execution time is linear in N. The size of the alphabet (number of objects in the model) and the number of rules are comparable to or better than those provided by previous solutions. The models provided are based on the kernel P system formalism, which is used for the first time in relation to the factorisation problem.

The papers is organised as follows: the next section introduces the kernel P system model; Section 3 introduces the two models, for unary and binary codification; in Section 4, complexity analysis is performed for both models, and are compared with previous results; several optimisations are presented in Section 5 and conclusions and future work are provided in Section 6.

2 Kernel P Systems

The solutions to the factorisation problem are based on the kernel P (kP) system model introduced in [36]. Kernel P systems use multisets of objects, rewriting and communication, and membrane division rules (all may have guards), various execution strategies, specific to each compartment, and types in order to specify more succinctly a problem.

Through membrane division and dynamic link creation and destruction, space is created. This approach has been a more versatile framework, with greater performances, as it can been seen in [37]. More details on various applications of kP systems may be found in [38]. These models are defined in a specific language, called kP-Lingua, which allows simulation and formal verification of the solution provided [39]. Only the necessary kP system concepts are introduced below. More details on kP systems can be found in [36] and for an introduction to multisets is recommended [40].

All the notions and definitions related to *kP systems* are from [36].

Definition 1 T is a *set of compartment types*, $T = \{t_1, \ldots, t_s\}$, where $t_i = (R_i, \sigma_i), 1 \leq i \leq s$, consists of a set of rules, R_i, and an execution strategy, σ_i, defined over $Lab(R_i)$, the labels of the rules of R_i.

Definition 2 *A kernel P (kP) system* of degree n is a tuple

$$k\Pi = (A, \mu, C_1, \ldots, C_n, i_0)$$

where A is a finite set of elements called objects; μ defines the membrane structure, which is a graph, (V, E), where V are vertices indicating components, and E edges; $C_i = (t_i, w_i), 1 \leq i \leq n$, is a compartment of the system consisting of a compartment

type from T and an initial multiset, w_i over A; i_0 is the output compartment where the result is obtained.

2.1 kP system rules

Definition 3 If g is the *abstract relational expression* $\gamma\ a^n$ and the current multiset is w, then the guard denotes the *relational expression* $\#_a(w)\gamma n$. The guard g is true for the multiset w if $\#_a(w)\gamma n$ is true.

Definition 4 If g is the *abstract Boolean expression* and the current multiset is w, then the guard denotes the *Boolean expression* for w, obtained by replacing abstract relational expressions with relational expressions for w. The guard g is true for the multiset w when the Boolean expression for w is true.

Definition 5 A guard is: (i) one of the Boolean constants true or false; (ii) an abstract relational expression; or (iii) an abstract Boolean expression.

Definition 6 A rule from a compartment $C_{l_i} = (t_{l_i}, w_{l_i})$ can have one of the following types:

- *(a)* **rewriting and communication** rule: $x \to y\{g\}$, where $x \in A^+$ and y has the form $y = (a_1, t_1), \ldots, (a_h, t_h), h \geq 0, a_j \in A$ and t_j indicates a compartment type from T; if t_j indicates the type of the current compartment, i.e. t_{l_i}, then it is ignored; if a link does not exist, then the rule is not applied; if a target, t_j, refers to a compartment type that has more than one instance connected to l_i, then one of them will be non-deterministically chosen;
- *(b)* **structure changing rules**; the following types are considered:
  - *(b1)* **membrane division rule**: $[x]_{t_{l_i}} \to [y_1]_{t_{i_1}} \ldots [y_p]_{t_{i_p}}\{g\}$, where $x \in A^+$ and y has the form $y_j = (a_{j,1}, t_{j,1}), \ldots, (a_{j,h_j}, t_{j,h_j})$, $1 \leq j \leq p$, like in rewriting and communication rules; the compartment l_i will be replaced by p compartments; the j-th compartment, instantiated from the compartment type t_j, contains the same objects as l_i, but x will be replaced by y_j; all the links of l_i are inherited by each of the newly created compartments;
  - *(b2)* **membrane dissolution** rule: $[]_{t_{l_i}} \to \lambda\ \{\ g\}$; the compartment l_i will be destroyed together with its links;
  - *(b3)* **link creation** rule: $[x]_{t_{l_i}};\ []_{t_{l_j}} \to [y]_{t_{l_i}} - []_{t_{l_j}}\{g\}$; the current compartment is linked to a compartment of type t_{l_j} and x is transformed into y; ; if more than one instance of the compartment type t_{l_j} exists then one of them will be non-deterministically picked up; g is a guard that refers to the compartment instantiated from the compartment type t_{l_1};

– *(b4)* **link destruction** rule: $[x]_{t_{l_i}} - []_{t_{l_j}} \rightarrow [y]_{t_{l_i}}$; $[]_{t_{l_j}}\{g\}$;
is the opposite of link creation and means that the compartments are disconnected.

In this paper, only rules of types (a), (b1) and (b2) are used.

2.2 kP system execution strategy

Definition 7 For a compartment type $t = (R, \sigma)$ from T and $r \in Lab(R)$, $r_1, \ldots, r_s \in Lab(R)$, the execution strategy, σ, is defined by the following

1. $\sigma = \lambda$, means no rule from the current compartment will be executed;
2. $\sigma = r$ – the rule r is executed;
3. $\sigma = \{r_1, \ldots, r_s\}$ – one of the rules labelled $r_1, \ldots, r_s$ will be chosen nondeterministically and executed; if none is applicable then none is executed; this is called *alternative* or *choice*;
4. $\sigma = \{r_1, \ldots, r_s\}^T$ - the rules are executed according to the *maximal parallelism* strategy.
5. $\sigma = \sigma_1 \& \ldots \& \sigma_s$, means executing sequentially $\sigma_1, \ldots, \sigma_s$, where $\sigma_i, 1 \leq i \leq s$, describes any of the above cases, namely λ, one rule, a choice, or maximal parallelism; if one of σ_i fails to be executed then the rest is no longer executed;
6. for any of the above σ strategies only one single structure changing rule is allowed.

3 Solutions to Integer Factorisation Problem

In the context of membrane computing, any solution to an *NP-Hard* problem has to have an underlying exponential space support. As exemplified by articles [30] and [34] there are two different approaches/philosophies:

1. The solution creates itself an exponential space in polynomial time, by cell division.
2. The solution is a module that represents a verifier to the *NP-Hard* problem. For every integer N, the verifier is acting upon an exponential space of compartments.

Subsequently are presented two linear time solutions to the *integer factorisation problem* utilised to break Rivest-Shamir-Adleman (RSA) encryption protocol. The models are based on kP systems formalism, each one mapping the first philosophy described above. The first solution uses unary encoding and the second one binary encoding. The solutions are implementated in kP-Lingua and can be found in *GitHub* [41].

3.1 Unary encoding

One of the first papers on arithmetic operations via membrane computing was [20], in which the numbers are stored in base 1. The same approach can be found in [42]. In this case, the addition can be implemented in one step.

In the solution presented below, integer numbers are codified in base 1, i.e., an integer number n is represented as a multiset l^n, where l is an object.

First, the algorithm for finding X and Y such that $N = X * Y$ is the following: generate all values x, y, where $x = 3, 5, 7\ldots$ and $y = 3, 5, 7\ldots$, such that $x \leq y$ and $x * y \leq N$; if there are x, y such that $x * y = N$ then the solution is $X = x$ and $Y = y$.

Remark 1 One can notice that the number of pairs generated above is $N*(N/2+1)/4$ which is less than the number of pairs, N^2, used in previous solutions [29–31].

In order to generate all x, y values as above one proceeds as follows:

Algorithm 1:

1. Start with $x = y = 1$.
2. A new pair of equal values x, y is obtained by increasing the previous values with 2 and the product $p = x * y$ is computed and compared to N; if $p < N$ then continue with the next step, otherwise a solution is found $(X = x, Y = y)$.
3. y is increased by 2 and $p = x * (y + 2)$ is computed and compared to N; if $p < N$ then this step is iterated; if $p = N$ then a solution is found; otherwise return to step 2.

One can parallelise steps 2 and 3, such that whenever a pair of equal values x, y is obtained and $x * y < N$ then step 3 will start in the same time with a new step 2, where $x + 2$ and $y + 2$ are calculated. Table 1 shows how these values are calculated in parallel.

(x^1, y^1)	———	———	———	———	—
—	$(x^3, y^3), (x^3, y^3)$	———	———	———	—
—	(x^3, y^5)	$(x^5, y^5), (x^5, y^5)$	———	———	—
—	(x^3, y^7)	(x^5, y^7)	$(x^7, y^7), (x^7, y^7)$	———	—
—	(x^3, y^9)	(x^5, y^9)	(x^7, y^9)	$(x^9, y^9), (x^9, y^9)$	—
...	...	...	...	...	...

Table 1: All (x, y) pairs of odd numbers considered by the solution.

One can notice that in step 2, the new product, $p = (y+2)^2$, can be written as $p = p' + 4y + 4$, where $p' = x * y$, is the product computed in the previous iteration. Also, in step 3, when a new value for y is computed, the product, $p = x * (y+2)$, can be written as $p = p' + 2x$, where $p' = x * y$. Consequently, apart from the first product $p = 1$, when $x = y = 1$, any other one is computed by adding to an existing one, $4y+4$ (step 2) or $2x$ (step 3). Hence, the algorithm uses only additions and multiplications with constants 4 or 2, respectively.

The kP system described below will implement the above mentioned algorithm in a parallel manner and operating on numbers represented in unary notation.

Definition 8 $k\Pi_{fact_1}$ denotes the *kP system* that resolves the *integer factorisation problem*:

$$k\Pi_{fact_1} = \{ A, \mu, C_1, C_2 \}$$

where: $A = \{ x, y, X, Y, p, n, z, x\prime \}$ is the finite set of objects; $\mu = (V, E)$ is the graph that defines the membrane structure, $V = \{ C_1, C_2 \}, E = (\{ C_1, C_2 \})$. Two compartment types are used, $T = \{ t_1, t_2 \}$; t_1 - root, where the solution is provided; and t_2 - the type solving the problem: pairs are generated, their product is computed and compared to N.

The meaning of the objects of A: x and y will count the values that give the product, defined by the number of objects p; the number of objects x' is always two times the number of objects x; z is used to introduce two objects y; n is a flag that appears only in a newly created compartment and disappears after it is divided.

$k\Pi_{fact_1}$ has the following compartments: $C_1 = (t_1, \lambda)$, $C_2 = (t_2, w_2)$, with the initial multiset $w_2 = ypn$.

$t_1 = (R_1, \sigma_1), \quad R_1 = \emptyset, \quad \sigma_1 = \emptyset;$
$t_2 = (R_2, \sigma_2), \quad R_2 = \{ r_{2,1}, \ldots, r_{2,9} \}, \qquad \sigma_2 = \{ r_{2,1}, \ldots, r_{2,8} \}^T \{ r_{2,9} \}.$

The execution strategy for σ_2 is to execute all the rules from $r_{2,1}, \ldots, r_{2,8}$ using *maximal parallelism*, followed by a *choice* strategy for the rule $r_{2,9}$.

$\mathtt{R_2}$:

$r_{2,\ 1}: \quad x \to (X, t_1) \quad \{ = p^N \ \wedge \ < n \}$
$r_{2,\ 2}: \quad y \to (Y, t_1) \quad \{ = p^N \ \wedge \ < n \}$
$r_{2,\ 3}: \quad n \to \lambda \quad \{ > p^N \}$
$r_{2,\ 4}: \quad x \ \to \lambda \quad \{ < p^N \ \wedge \ = n \}$
$r_{2,\ 5}: \quad x' \to \lambda \quad \{ < p^N \ \wedge \ = n \}$
$r_{2,\ 6}: \quad y \ \to xyx\prime^2 p^4 \quad \{ < p^N \ \wedge \ = n \}$
$r_{2,\ 7}: \quad x\prime \to px\prime \quad \{ < p^N \ \wedge \ < n \}$
$r_{2,\ 8}: \quad z \ \to zy^2 \quad \{ < p^N \}$
$r_{2,\ 9}: \quad [n]_{t_2} \to [p^4 y^2 x^2 x'^4 z]_{t_2} [np^4 y^2]_{t_2} \quad \{ < p^N \}$

The main part of algorithm described above, that finds X, Y such that $N = X * Y$, is defined in compartment C_2. When the solution is found, the rules $r_{2,1}, r_{2,2}$ will send x, y to C_1 as X, Y.

In order to describe how the rules R_2 compute values x, y and compare their product with N we introduce the configuration of a compartment C_2 as being given by the multiset $x^{h_1}x'^{h_2}y^{h_3}p^{h_4}n^{h_5}z^{h_6}$, where $h_i \geq 0$, $1 \leq i \leq 6$. As this compartment will be repetitively divided, according to step 2 of the algorithm, the initial one will be considered instance number 1 and denoted $C_{2,1}$. Its initial configuration is $x^0x'^0ypnz^0$, corresponding to initial multiset w_2, and is denoted by $M_{1,0}$. The compartments and configurations are illustrated in Table 2.

One can notice that the execution strategy σ_2 is a sequence with rules $r_{2,1}, \ldots, r_{2,8}$ executed with maximal parallelism and followed by $r_{2,9}$ (when applicable), which is a membrane division rule.

Starting from $M_{1,0}$ the execution strategy σ_2 is applied, i.e., $r_{2,6}$ is used in a maximal way and is followed by $r_{2,9}$, when this is applicable (choice). This latest rule will divide $C_{2,1}$ into a compartment denoted also by $C_{2,1}$, as this will continue step 3 of the algorithm, and a new instance of C_2, denoted by $C_{2,2}$, will be created.

The new $C_{2,1}$ will have the configuration $M_{1,1} = x^3x'^6y^3p^9n^0z^1$ and a new instance, $C_{2,2}$, will start with the configuration $M_{2,0} = x_1x'_1y^3p^9n^1z^0$. One can observe that in both configurations there are 3 objects y and 9 objects p. In $C_{2,1}$ there are 3 objects x and 6 of x' (as number of objects x' is two times those of x). Also, in $C_{2,1}$ n has disappeared and z has appeared; z will be used to increase the number of objects y by 2. In $C_{2,2}$ n is present and this means that in the next step this compartment will be divided, unless the solution has been found. The number of objects x, x' is irrelevant (hence, denoted x_1, x'_1, respectively), as in the next step they will be erased (rules $r_{2,4}, r_{2,5}$).

The compartment $C_{2,1}$ is no longer divided, as there is no n and in each step two more y objects will be added and p will be increased by the number of objects x'. The next configuration of $C_{2,1}$ will be $M_{1,2} = x^3x'^6y^5p^{15}n^0z^1$ and then $M_{1,3} = x^3x'^6y^7p^{21}n^0z^1$. In each case $r_{2,7}, r_{2,8}$ will be executed in maximal parallel way, increasing p and y, respectively.

The compartment $C_{2,2}$ is instead divided and after using σ_2 the compartment $C_{2,2}$ is divided, obtaining, similarly to the case of dividing $C_{2,1}$, a compartment denoted also with $C_{2,2}$ and another one denoted $C_{2,3}$. The corresponding configurations are $M_{2,1} = x^5x'^{10}y^5p^{25}n^0z^1$ and $M_{3,0} = x_2x'_2y^5p^{25}n^1z^0$, respectively.

The model written in kP-Lingua is available in GitHub [41].

Config./ Step	$C_{2,1}$	$C_{2,2}$	$C_{2,3}$	$C_{2,4}$	...
0	$M_{1,0} = x^0x'^0ypnz^0$	———	———	———	—
1	$M_{1,1} = x^3x'^6y^3p^9n^0z^1$	$M_{2,0} = x_1x'_1y^3p^9n^1z^0$	———	———	—
2	$M_{1,2} = x^3x'^6y^5p^{15}n^0z^1$	$M_{2,1} = x^5x'^{10}y^5p^{25}n^0z^1$	$M_{3,0} = x_2x'_2y^5p^{25}n^1z^0$	———	—
3	$M_{1,3} = x^3x'^6y^7p^{21}n^0z^1$	$M_{2,2} = x^5x'^{10}y^7p^{35}n^0z^1$	$M_{3,1} = x^7x'^{14}y^7p^{49}n^0z^1$	$M_{4,0} = x_3x'_3y^7p^{49}n^1z^0$	—
...	...	...	...	...	...

Table 2: Compartments and configurations.

3.2 Binary encoding

In the case of binary encoding, the underlying idea for the factorisation algorithm is the same as in unary encoding. There are some changes that reflect the new encoding. As such, the binary enconding is discussed in the context of *kP systems*:

Notation 1. For any given positive number $N \in \mathbb{N}$, let $k \in \mathbb{N}$ denote the number of bits that are needed to encode N in a binary format. The most significant bit (MSB) is on position $k-1$, and the least significant bit (LSB) is on position 0.

For example, for $N = 16$ then $k = 5$, with the following binary representation, 10000; where MSB=1 is on position 4 and LSB=0 on 0.

Notation 2. For a given positive number $N \in \mathbb{N}$, let $k \in \mathbb{N}$ denote the number of bits that are needed to encode N in a binary format, with the MSB on position $k-1$. One can encode N, in a binary format, as a sequence of the bits that have only the value 1, and the position of those bits. The bits that have the value 0 are encoded by the absence of those bits.

In case of $N = 17, k = 5$, the binary representation would be $1_4 1_0$.

Definition 9 Let $N \in \mathbb{N}$ be a positve integer, and let $k \in \mathbb{N}$ denote the number of bits that are needed to encode N in a binary format. Let $\mathcal{N} = \{ N_0, \ldots, N_{k-1} \mid k \in \mathbb{N}\}$ be a set of objects, where N_i represents the value of 1 on position i, $0 \leq i < k-1$, and $\mathcal{N}_\star$ be a multiset over $\mathcal{N}$. The notation $\mathcal{N}_\star$ is the *kP system binary encoding* of N.

For example, let $N = 17$, $k = 5$, $\mathcal{N} = \{ N_0, \ldots N_4\}$, then the *kP system binary encoding* of N is the multiset $\mathcal{N}_\star = N_4^1 N_0^1$.

Definition 10 Let $k\Pi_{fact_2}$ the *kernel P system* that resolves the *integer factorisation problem* for binary encoding:

$$k\Pi_{fact_2} = \{ A, \mu, C_1, C_2 \}$$

where: $A = A_1 \cup A_2$ is the finite set of objects; $\mu = (V, E)$ is the graph that defines the membrane structure, $V = \{ C_1, C_2 \}, E = (\{ C_1, C_2 \})$. Similar to $k\Pi_{fact_1}$, two compartment types are used $T = \{ t_1, t_2 \}$, with the same meaning.

The set A_1 is a set composed only of objects that are meant to be used as flags: $A_1 = \{ B,\ D,\ P_?,\ C_{sol},\ Y_{+2},\ Y\prime_{+2},\ P_s,\ P,\ P_n,\ P_1,\ P_2,\ P_G,\ P_L,\ S,\ S_d,\ L_{<<},\ L_{2<<},\ s,\ d,\ r \}$.

The set A_2 is a set composed of objects required for encoding numbers, as shown in notations 1 and 2, where k is the number of bits used to encode N:

$$A_2 = \mathcal{X} \cup \mathcal{Y} \cup \mathcal{P} \cup \mathcal{Z} \cup \mathcal{X}\prime \cup \mathcal{Y}\prime \cup \mathcal{N} \quad \text{where,}$$
$$\mathcal{X} = \{ x_0, \ldots, x_{k-1} \mid k \in \mathbb{N} \},\ \mathcal{Y} = \{ y_0, \ldots, y_{k-1} \mid k \in \mathbb{N} \},$$
$$\mathcal{P} = \{ p_0, \ldots, p_{k-1} \mid k \in \mathbb{N} \}\ \mathcal{Z} = \{ z_0, \ldots, z_{k-1} \mid k \in \mathbb{N} \},$$
$$\mathcal{X}\prime = \{ X_0, \ldots, X_{k-1} \mid k \in \mathbb{N} \},\ \mathcal{Y}\prime = \{ Y_0, \ldots, Y_{k-1} \mid k \in \mathbb{N} \},$$
$$\mathcal{N} = \{ N_0, \ldots, N_{k-1} \mid k \in \mathbb{N} \}.$$

$k\Pi_{fact_2}$ has the following compartments: $C_1 = (t_1, \lambda)$, $C_2 = (t_2, w_2)$, with the initial multiset $w_2 = x_1 y_1 p_1 r P_n \mathcal{N}_\star$, where $\mathcal{N}_\star$ is the *kP system binary encoding* of N, as multiset over the set $\mathcal{N} \subset A$ (see defitinion 9 and notations 1, 2), where:

$$t_1 = (R_1, \sigma_1), \quad R_1 = \emptyset, \quad \sigma_1 = \emptyset;$$
$$t_2 = (R_2, \sigma_2),$$
$$R_2 = \{ r_{2,\ 1},\ r_{2,\ 2,\ i},\ \ldots, r_{2,\ 4,\ i},\ r_{2,\ 5},\ \ldots,\ r_{2,\ 10},\ r_{2,\ 11,\ i},\ r_{2,\ 12},\ \ldots,\ r_{2,\ 19},$$
$$r_{2,\ 20,\ i},\ \ldots,\ r_{2,\ 27,\ i},\ r_{2,\ 28},\ r_{2,\ 29},\ r_{2,\ 30,\ \tau},\ r_{2,\ 31,\ \tau},\ r_{2,\ 32},\ r_{2,\ 33} \mid 0 \leq i, \tau \leq k-1 \},$$
$$\sigma_2 = \sigma_{2,\ 1}\ \sigma^\star_{2,\ 2}\ \sigma_{2,\ 3}\ \sigma_{2,\ 4},$$
$$\sigma_{2,\ 1} = \{ r_{2,\ 1},\ r_{2,\ 2,\ i},\ \ldots, r_{2,\ 4,\ i},\ r_{2,\ 5},\ \ldots,\ r_{2,\ 10},\ r_{2,\ 11,\ i},\ r_{2,\ 12},\ \ldots,\ r_{2,\ 19},$$
$$r_{2,\ 20,\ i},\ \ldots,\ r_{2,\ 27,\ i},\ r_{2,\ 28},\ r_{2,\ 29} \mid 0 \leq i \leq k-1 \}^T,$$
$$\sigma^\star_{2,\ 2} = \{ r_{2,\ 30,\ k-1},\ r_{2,\ 31,\ k-1} \}^T \{ r_{2,\ 30,\ k-2},\ r_{2,\ 31,\ k-2} \}^T \cdots$$
$$\cdots \{ r_{2,\ 30,\ 0},\ r_{2,\ 31,\ 0} \}^T,$$
$$\sigma_{2,\ 3} = \{ r_{2,\ 32} \}^T,\ \sigma_{2,\ 4} = \{ r_{2,\ 33} \}^T.$$

One could remember that, $\mathcal{N}_\star$ is the *kP system binary encoding* of $N = X * Y$. The pair (X_i, Y_i) represents the *kP system binary encoding* of the solution (X, Y). The pair (x_i, y_i) denote two additional variables in *kP sytem binary encoding*. This pair is helpful for searching all potential solutions, as stated in the Algorithm 3.1. The product $p = x * y$ is denoted as p_i in *kP system binary encoding*. At last, z_i is an auxiliary variable in which the sum of two numbers is computed. All the rules of the model are listed below:

$\mathtt{R}_2$:

$r_{2,\ 1}$: $\mathtt{r} \rightarrow \lambda\ \{=P_G\}$

$r_{2,\ 2,i}$: $\mathtt{x}_i \rightarrow (X_i, t_1)\quad \{>=\mathtt{s}\}$

$r_{2,\ 3,i}$: $\mathtt{y}_i \rightarrow (Y_i, t_1)\quad \{>=\mathtt{s}\}$

$r_{2,\ 4,i}$: $\mathtt{p}_i \rightarrow \mathtt{z}_i\ \{=P_n\ \ =P\}$

$r_{2,\ 5}$: $P_n \rightarrow P_1 L_{2<<}$

$r_{2,\ 6}$: $P \rightarrow P_s L_{<<}$

$r_{2,\ 7}$: $\mathtt{r} \rightarrow s\ \{< P_G\ \wedge < P_L \wedge = C^2_{sol}\}$

$r_{2,\ 8}$: $dP_L \rightarrow D\quad \{= C_{sol}\}$

$r_{2,\ 9}$: $P_L \rightarrow P\quad \{= C_{sol}\ \wedge\ < d\}$

$r_{2,\ 10}$: $C_{sol} \rightarrow \lambda\ \{(= C_{sol} \mid\ = C^2_{sol}) \wedge < P_L\}$

$r_{2,\ 11,i}$: $z_i^2 \rightarrow \mathtt{z}_{i+1}\ \{=\mathtt{S}\}$

$r_{2,\ 12}$: $\mathtt{S} \rightarrow S_d\ \{<2z_0 \wedge \cdots \wedge < 2z_{k-1}\}$

$r_{2,\ 13}$: $\mathtt{r} \rightarrow \mathtt{rz}_2\ \{=S_d\ \wedge\ =P_1\}$

$r_{2,\ 14}$: $\mathtt{r} \rightarrow \mathtt{rz}_1\ \{=S_d\ \wedge\ (=P_2\ \ =P_s)\}$

$r_{2,\ 15}$: $P_1 S_d \rightarrow \mathtt{S}P_2$

$r_{2,\ 16}$: $P_2 S_d \rightarrow \mathtt{S}Y_{+2}$

$r_{2,\ 17}$: $P_s S_d \rightarrow \mathtt{S}Y\prime_{+2}$

$r_{2,\ 18}$: $Y_{+2} S_d \rightarrow dP_?$

$r_{2,\ 19}$: $Y\prime_{+2} S_d \rightarrow P_?$

$r_{2,\ 20,i}$: $\mathtt{z}_i \rightarrow \mathtt{p}_i\ \{=S_d\ \wedge\ (=P_2\ \ =P_s)\}$

$r_{2,\ 21,i}$: $\mathtt{y}_i \rightarrow \mathtt{z}_i\ \{=S_d\ \wedge\ =P_2\}$

$r_{2,\ 22,i}$: $\mathtt{x}_i \rightarrow \lambda\ \{=Y_{+2}\}$

$r_{2,\ 23,i}$: $\mathtt{z}_i \rightarrow \mathtt{y}_i x_i\ \{=S_d\ \wedge\ =Y_{+2}\}$

$r_{2,\ 24,i}$: $\mathtt{y}_i \rightarrow \mathtt{z}_i\ \{=S_d\ \wedge\ =P_s\}$

$r_{2,\ 25,i}$: $\mathtt{z}_i \rightarrow \mathtt{y}_i\ \{=S_d\ \wedge\ =Y\prime_{+2}\}$

$r_{2,\ 26,i}$: $\mathtt{x}_i \rightarrow \mathtt{x}_i z_{i+1}\ \{=L_{<<}\}$

$r_{2,\ 27,i}$: $\mathtt{y}_i \rightarrow \mathtt{y}_i z_{i+2}\ \{=L_{2<<}\}$

$r_{2,\ 28}$: $L_{<<} \rightarrow \mathtt{S}$

$r_{2,\ 29}$: $L_{2<<} \rightarrow \mathtt{S}$

$r_{2,\ 30,\tau}$: $P_? \rightarrow C_{sol} P_G\ \{=P_?\ \wedge\ =\mathtt{p}_\tau\ \wedge\ <\mathtt{N}_\tau\}$,

$r_{2,\ 31,\tau}$: $P_? \rightarrow C_{sol} P_L\ \{=P_?\ \wedge\ <\mathtt{p}_\tau\ \wedge\ =\mathtt{N}_\tau\}$,

$r_{2,\ 32}$: $P_? \rightarrow C^2_{sol}$

$r_{2,\ 33}$: $[D]_{t_2} \rightarrow [P]_{t_2} [P_n]_{t_2}$

where $0 \leq i, \tau < k$.

The algoritm for resolving *integer factorisation problem* is the same as in unary encoding (see Algorithm 3.1) . The main differences comes from the particularities of the binary encoding. Because in binary case the order matters; one must emulate the folowing operations: a) sum of two numbers; b)left shift and two times left shift of a number; c) a comparison method for p_i with respect to $\mathcal{N}_\star$.

The sum of two numbers is quite easy, and it is composed of only two rules:

$$\begin{aligned} r_{2,\ 11,\ i}: &\quad z_i^2 \to z_{i+1} \\ r_{2,\ 12} &: \quad S \to S_d \quad \{< 2z_0 \wedge \cdots \wedge < 2z_{k-1}\} \end{aligned}$$

This works as follows: in order to compute the sum of x_i and y_i, all the bits of x_i and y_i are copied into z_i. The rule $r_{2,\ 11,\ i}$ computes the sum into z_i with respect to *carry*. The rule $r_{2,\ 12}$ marks the end of the computations for the sum.

The rules $r_{2,\ 26,\ i}$ and $r_{2,\ 27,\ i}$ computes the results of $2 * x_i$ and $4 * y_i$, without modifying the values of x_i, y_i. The sensitive part is the timing. The flags $L_{<<}$ and $L_{2<<}$ marks the signal to start computing left shift, on x_i and y_i, at certain points in time.

$$\begin{aligned} r_{2,\ 26,\ i}: &\quad x_i \to x_i z_{i+1} \quad \{= L_{<<}\} \\ r_{2,\ 27,\ i}: &\quad y_i \to y_i z_{i+2} \quad \{= L_{2<<}\} \end{aligned}$$

As for the comparison of $p = x * y$ with N, a bit by bit comparisons is done, starting from MBS down to LSB. In this case, the value of i starts from $k-1$ down to 0. As such, if exists a bit p_τ and NOT exists a bit N_τ then the value of p is greater than N, marked here by the symbol P_G. In reverse, if exists a bit N_τ and NOT exists a bit p_τ then the value of p is less than N, marked by the symbol P_L. The symbol $P_?$ is a flag - the meaning of it is to mark the time when to make the comparison. C_{sol} is another flag, which comes from *check solution*.

$$\begin{aligned} r_{2,\ 30,\ \tau}: &\quad P_? \to C_{sol} P_G \quad \{= P_? \wedge = p_i \wedge < N_i\} \\ r_{2,\ 31,\ \tau}: &\quad P_? \to C_{sol} P_L \quad \{= P_? \wedge < p_i \wedge = N_i\} \end{aligned}$$

If p is not greater or less than N, then it must be equal with it. As such, the next step is to check for solution. In this case, one must differentiate from the previous step (i.e., rules $r_{2,\ 30}$, $r_{2,\ 31}$). Thus, two objects C_{sol} are sent to the systems - $r_{2,\ 32}$. This will be important later on, when the model checks for solution.

$$r_{2,\ 32}: \quad P_? \to C_{sol}^2$$

In order to describe how the rules R_2 computes values x, y and compare their product with N, we introduce the configuration of a compartment C_2 as being given by the multiset $w_\star = x_{\star i,j} y_{\star i,j} p_{\star i,j} z_{\star i,j} \delta_{i,j}, \quad i > 1, j \geq 0$, such that, $x_{\star i,j}$, $y_{\star i,j}$, $p_{\star i,j}$, $z_{\star i,j}$ are multisets over the sets $\mathcal{X}, \mathcal{Y}, \mathcal{P}, \mathcal{Z}$, and $\delta_{i,j}$ is a multiset over the set $\Delta = A - (\mathcal{X} \cup \mathcal{Y} \cup \mathcal{P} \cup \mathcal{Z})$.

As C_2 compartment will be repetitively divided, according to step 2 of the algorithm 3.1, the initial one will be considered instance number 1 and denoted $C_{2,1}$. The initial configuration $C_{2,1}$ is $M_{1,0} = x_{\star 1,0}y_{\star 1,0}p_{\star 1,0}z_{\star 1,0}\delta_{1,0}$, where $x_{\star 1,0} = x_0,\ y_{\star 1,0} = y_0,\ p_{\star 1,0} = p_0,\ z_{\star 1,0} = \lambda,\ \delta_{1,0} = rP_n\mathcal{N}_\star,\ \ \mathcal{N}_\star$ is the *kP system binary encoding* of N, as multiset over the set $\mathcal{N} \subset A$ (see definitions 10, 9 and notations 1, 2).

After the rule $r_{2,33}$ has been applied, the compartment $C_{2,1}$ is divided, thus resulting in two compartments: one denoted also by $C_{2,1}$, and a new instance of C_2, denoted by $C_{2,2}$.

The new $C_{2,1}$ will have the configuration $M_{1,29} = x_{\star 1,29}y_{\star 1,29}p_{\star 1,29}z_{\star 1,29}\delta_{1,29}$, where $x_{\star 1,29} = x_1x_0,\ y_{\star 1,29} = y_1y_0,\ p_{\star 1,29} = p_3p_0,\ z_{\star 1,29} = \lambda,\ \delta_{1,29} = rP\mathcal{N}_\star$.

The new $C_{2,2}$ will have the initial configuration $M_{2,0} = x_{\star 2,0}y_{\star 2,0}p_{\star 2,0}z_{\star 2,0}\delta_{2,0}$, where $x_{\star 2,0} = x_1x_0,\ y_{\star 2,0} = y_1y_0,\ p_{\star 2,0} = p_3p_0,\ z_{\star 2,0} = \lambda,\ \delta_{2,0} = rP_n\mathcal{N}_\star$. One could remark that, there exists a P_n in $\delta_{2,0}$, thus the step 2 from the algorithm will be repeated.

Tables 3 and 4 show the configurations of $C_{2,1}$, starting from the initial configuration $M_{1,0}$ up until the final configuration $M_{1,5k+9}$. In those tables are illustrated all the steps taken and rules applied by $k\Pi_{fact_2}$ in order to factorise $N = 15$, that has the solution $X = 3, Y = 5$. In this case, the *kP systems binary encoding* of N is $\mathcal{N}_\star = N_3N_2N_1N_0$, and the result will be the pair of multisets (x_1x_0, y_2y_0).

In case there is a P_n in the compartment, the rules from Table 3 are applied; otherwise, in case there is a P in the compartment, are applied the rules from Table 4.

Although the custom is to represent each object, within a multiset, without any separator, in case of the tables mentioned above, for a more accurate overview, a comma has been chosen to separate them.

4 Complexity

In the first model, $k\Pi_{fact_1}$, integer numbers are codified in base 1, i.e., an integer number n is represented as a multiset l^n, where l is an object. This solution improves the previous solutions in terms of steps required for solving the *integer factorisation problem*, since the addition is done in $O(1)$. Hence, the maximum number of steps is $\lfloor N/6 \rfloor + 1$, so the solution is linear in N and the number of compartments is bounded by $\lceil \sqrt{N} \rceil$.

In the second model, $k\Pi_{fact_2}$, integer numbers are in binary encoding, and codified as sequence, using only the bits with the value 1. From Tables 3 and 4 it turns out that the time complexity of this solution is slightly bigger than in the unary case, $O(k * N)$; whereas the number of compartments is the same. So, the space complexity of both solutions is $O(\sqrt{N})$. The time complexity

Config.	$x_{\star 1}$	$y_{\star 1}$	$p_{\star 1}$	$z_{\star 1}$	δ_1	Rules
$M_{1,0}$	x_0	y_0	p_0		$\mathcal{N}_\star$, r, P_n	$r_{2,4,\ i}$, $r_{2,5}$
$M_{1,1}$	x_0	y_0		z_0	$\mathcal{N}_\star$, r, P_1, $L_{2<<}$	$r_{2,27,\ i}$, $r_{2,29}$
$M_{1,2}$	x_0	y_0		z_2, z_0	$\mathcal{N}_\star$, r, P_1, S	$r_{2,11,\ i}$
...	...	...	...	...	...	...
$M_{1,k}$	x_0	y_0		z_2, z_0	$\mathcal{N}_\star$, r, P_1, S	$r_{2,11,\ i}$, $r_{2,12}$
$M_{1,k+1}$	x_0	y_0		z_2, z_0	$\mathcal{N}_\star$, r, P_1, S_d	$r_{2,13}$, $r_{2,15}$
$M_{1,k+2}$	x_0	y_0		z_2^2, z_0	$\mathcal{N}_\star$, r, S, P_2	$r_{2,11,\ i}$
$M_{1,k+3}$	x_0	y_0		z_3, z_0	$\mathcal{N}_\star$, r, S, P_2	$r_{2,11,\ i}$
...	...	...	...	...	...	...
$M_{1,2k}$	x_0	y_0		z_3, z_0	$\mathcal{N}_\star$, r, S, P_2	$r_{2,11,\ i}$, $r_{2,12}$
$M_{1,2k+1}$	x_0	y_0		z_3, z_0	$\mathcal{N}_\star$, r, S_d, P_2	$r_{2,20,\ i}$, $r_{2,21,\ i}$, $r_{2,14}$, $r_{2,16}$
$M_{1,2k+2}$	x_0		p_0, p_3	z_1, z_0	$\mathcal{N}_\star$, r, S, Y_{+2}	$r_{2,22,\ i}$, $r_{2,11,\ i}$
...	...	...	...	...	...	...
$M_{1,3k}$			p_0, p_3	z_1, z_0	$\mathcal{N}_\star$, r, S, Y_{+2}	$r_{2,11,\ i}$, $r_{2,12}$
$M_{1,3k+1}$			p_0, p_3	z_1, z_0	$\mathcal{N}_\star$, r, S_d, Y_{+2}	$r_{2,23,\ i}$, $r_{2,18}$
$M_{1,3k+2}$	x_0, x_1	y_1, y_0	p_0, p_3		$\mathcal{N}_\star$, r, $P_?$, d	$r_{2,\ 31,\ 3}$
$M_{1,3k+3}$	x_0, x_1	y_1, y_0	p_0, p_3		$\mathcal{N}_\star$, r, d, P_L, C_{sol}	$r_{2,8}$
$M_{1,3k+4}$	x_0, x_1	y_1, y_0	p_0, p_3		$\mathcal{N}_\star$, r, C_{sol}, D	$r_{2,10}$ $r_{2,33}$

Table 3: The compartment $C_{2,1}$ with configuration $M_{1,0} = x_{\star 1,0} y_{\star 1,0} p_{\star 1,0} z_{\star 1,0} \delta_{1,0}$, where $x_{\star 1,0} = x_0,\ y_{\star 1,0} = y_0,\ p_{\star 1,0} = p_0,\ z_{\star 1,0} = \lambda,\ \delta_{1,0} = r P_n \mathcal{N}_\star$

reported in [29] is $O(k * logk)$ and in [30] is $O(k)$, whereas the space complexity is $O(N^2)$ for both of them.

One can assess the size of the models used in these solutions by counting the number of objetcs and rules. In the case of $k\Pi_{fact_1}$, the number of objects is 8 and number of rules is 9; so, both have $O(1)$ complexity. The binary encoding kP system model has $7k+30$ objects and $15k+18$ rules; so, linear, in k, number of objects and rules. In [29] and [30], both the number of objects and rules are polynomial in k, $O(k^2)$.

These comparisons clearly show that the two solutions presented in the paper have a better usage of the space, and also model sizes, as expressed by the number of objects and rules, but less efficient than those provided in [29, 30] with respect to time complexity.

Config.	$x_{\star 1}$	$y_{\star 1}$	$p_{\star 1}$	$z_{\star 1}$	δ_1	Rules to be applied
$M_{1,3k+5}$	x_0, x_1	y_1, y_0	p_0, p_3		$\mathcal{N}_\star$, r, P	$r_{2,4,\ i}$, $r_{2,6}$
$M_{1,3k+6}$	x_0, x_1	y_1, y_0		z_0, z_3	$\mathcal{N}_\star$, r, $L_{<<}$, P_s	$r_{2,26,\ i}$, $r_{2,28}$
$M_{1,3k+7}$	x_1, x_0	y_1, y_0		z_2, z_0, z_3, z_1	$\mathcal{N}_\star$, r, S, P_s	$r_{2,11,\ i}$
...	...	...	...	...	...	...
$M_{1,4k+5}$	x_1, x_0	y_1, y_0		z_2, z_0, z_3, z_1	$\mathcal{N}_\star$, r, S, P_s	$r_{2,11,\ i}$, $r_{2,12}$
$M_{1,4k+6}$	x_1, x_0	y_1, y_0		z_2, z_0, z_3, z_1	$\mathcal{N}_\star$, r, S_d, P_s	$r_{2,20,\ i}$, $r_{2,24,\ i}$, $r_{2,14}$, $r_{2,17}$
$M_{1,4k+7}$	x_1, x_0		p_3, p_0, p_2, p_1	z_0, z_1^2	$\mathcal{N}_\star$, r, S, $Y\prime_{+2}$	$r_{2,11,\ i}$
...	...	...	...	...	...	...
$M_{1,5k+5}$	x_1, x_0		p_3, p_0, p_2, p_1	z_0, z_2	$\mathcal{N}_\star$, r, S, $Y\prime_{+2}$	$r_{2,11,\ i}$, $r_{2,12}$
$M_{1,5k+6}$	x_1, x_0		p_3, p_0, p_2, p_1	z_0, z_2	$\mathcal{N}_\star$, r, S_d, $Y\prime_{+2}$	$r_{2,25,\ i}$ $r_{2,19}$
$M_{1,5k+7}$	x_1, x_0	y_0, y_2	p_3, p_0, p_2, p_1		$\mathcal{N}_\star$, r, $P_?$	$r_{2,32}$
$M_{1,5k+8}$	x_1, x_0	y_0, y_2	p_3, p_0, p_2, p_1		$\mathcal{N}_\star$, r, C^2_{sol}	$r_{2,7}$, $r_{2,10}$
$M_{1,5k+9}$	x_1, x_0	y_0, y_2	p_3, p_0, p_2, p_1		$\mathcal{N}_\star$, s	$r_{2,2,\ i}$, $r_{2,3,\ i}$

Table 4: The compartment $C_{2,1}$ starting with configuration $M_{1,3k+5} = x_{\star 1,3k+5} y_{\star 1,3k+5} p_{\star 1,3k+5} z_{\star 1,3k+5} \delta_{1,3k+5}$, where $x_{\star 1,3k+5} = x_1 x_0,\ y_{\star 1,3k+5} = y_1 y_0,\ p_{\star 1,3k+5} = p_3 p_0,\ z_{\star 1,3k+5} = \lambda,\ \delta_{1,3k+5} = rP\mathcal{N}_\star$

5 Optimisations

One type of optimisation that can be done is to stop the entire kP system, once a solution has been found. This means that the current compartments will be dissolved and no other compartment will be created. An illustration for the unary encoding model, $k\Pi_{fact_1}$, will be given below, but the same is true, as well, for binary encoding.

With this is mind, one more object, m, is added to the set A, and this becomes $A = \{x, y, X, Y, p, n, z, x\prime, m\}$. The compartment C_1 is now, $C_1 = (t_1, w_1)$, where $w_1 = \lambda$ and $t_1 = (R_1, \sigma_1)$, $\quad R_1 = \{r_{1,1},\ r_{1,2}\}$, $\quad \sigma_1 = \{r_{1,1},\ r_{1,2}\}^T$. In case of C_2, w_2 is changed into $w_2 = ypnm$, the rule $r_{2,0}$ is added to the set of rules R_2, such that $\sigma_2 = r_{2,0}\{r_{2,1}, \ldots, r_{2,8}\}^T\{r_{2,9}\}$, and $r_{2,2}$ will be slightly modified, as can be seen below:

$r_{1,\ 1}:\quad m \to \lambda$
$r_{1,\ 2}:\quad []_{t_1} \to \lambda \quad \{= S\}$

$r_{2,\ 0}:\quad m \to (m, t_1)(m, t_2)$
$r_{2,\ 2}:\quad y \to (YS, t_1) \quad \{= p^N \wedge\ < n\}$

These modifications will enhance $k\Pi_{fact_1}$ to be able to stop the entire system when a solution is found. Indeed, when the solution is found in an instance of

C_2 then an S is sent to C_1 through $r_{2,2}$. When S is present in compartment C_1 then this is dissolved using the rule $r_{1,2}$. When the rule $r_{2,0}$ is meant to be executed (it is the first rule of σ_2 execution strategy) then this cannot be used as the target compartment is no longer available, as it was dissolved. Hence, the entire sequence defining σ_2 stops. For a more in depth look into why the entire system stops, please see the paper [36].

6 Conclusions and Future Work

In this paper, two models in *kP systems* have been successfully created, factorising any integer N that can be written as $N = X \times Y$ and $9 \leq N$, where X, Y are two large prime numbers. Both models lead to algorithms that run in linear time with respect to input N, using only $O(\sqrt{N})$ space. This is achieved by looking in a parallel way for all the pairs (x, y) and verifying if their product is equal to N. The number of (x, y) pairs considered in the paper is smaller than those used in previous research and also the representation of each binary number is more condensed than previous representations as only bits equal to 1 are kept. Also, the number of objects and rules is smaller. However, the solutions provided are less efficient than previous ones and this is an aspect to be considered in future developments.

The assistant tool *kPWorkbench*[43, 44] has been very helpful in all the design process, providing very useful insights. With the help of it, various simulations have been tested in *kP-Lingua*.

Moreover, the solutions proposed can be a blueprint for a more refined version, since these are the first attempts for the *integer factorisation problem* using kP systems.

Acknowledgments. This research was supported by the European Regional Development Fund, Competitiveness Operational Program 2014-2020 through project IDBC (code SMIS 2014+: 121512). Marian Gheorghe has been partially supported by the Royal Society grant IES\R3\213176, 2022-2024.

References

[1] Schneier, B.: Applied cryptography: protocols, algorithms, and source code in C. John Wiley & sons (2007)

[2] Boneh, D., Durfee, G.: Cryptanalysis of RSA with private key d less than $N^{0.292}$. IEEE Transactions on Information Theory **46**(4), 1339–1349 (2000)

[3] Wiener, M.J.: Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information theory **36**(3), 553–558 (1990)

[4] Mumtaz, M., Ping, L.: Forty years of attacks on the RSA cryptosystem: A brief survey. Journal of Discrete Mathematical Sciences and Cryptography **22**(1), 9–29 (2019)

[5] Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000)

[6] Alhazov, A., Freund, R., Ivanov, S.: When catalytic P systems with one catalyst can be computationally complete. Journal of Membrane Computing **3**(3), 170–181 (2021)

[7] Wu, T., Jiang, S.: Spiking neural P systems with a flat maximally parallel use of rules. Journal of Membrane Computing **3**(3), 221–231 (2021)

[8] Liu, Y., Zhao, Y.: Weighted spiking neural P systems with polarizations and anti-spikes. Journal of Membrane Computing **4**(4), 269–283 (2022)

[9] Liu, L., Jiang, K.: Turing universality of sequential spiking neural P systems with polarizations as number accepting devices. Journal of Membrane Computing **4**(3), 232–242 (2022)

[10] Ning, G., Valencia-Cabrera, L., Song, X.: Small universal improved spiking neural P systems with multiple channels and autapses. Journal of Membrane Computing **4**(2), 153–165 (2022)

[11] Lazo, P.P.L., De La Cruz, R.T.A., Macababayao, I.C.H., Cabarle, F.G.C.: Universality of SN P systems with stochastic application of rules. Journal of Membrane Computing **4**(2), 166–176 (2022)

[12] Zhang, H., Liu, X., Shao, Y.: Chinese dialect tone's recognition using gated universal spiking neural P systems. Journal of Membrane Computing **4**(4), 284–292 (2022)

[13] Dong, J., Zhang, G., Xiao, D., Luo, B., Rong, H.: Migration strategy in distributed adaptive optimization spiking neural P systems. Journal of Membrane Computing **4**(4), 314–328 (2022)

[14] Qiu, C., Xue, J., Liu, X., Li, Q.: Deep dynamic spiking neural P systems with applications in organ segmentation. Journal of Membrane Computing **4**(4), 329–340 (2022)

[15] Dong, J., Zhang, G., Luo, B., Xiao, D.: Multi-learning rate optimization spiking neural P systems for solving the discrete optimisation problems. Journal of Membrane Computing **4**(3), 209–221 (2022)

[16] Yang, X., Qian, L., Liu, X., Xue, J.: Am improved deep echo state network inspired by tissue-like P systems forecasting for non-stationary time series. Journal of Membrane Computing **4**(3), 222–231 (2022)

[17] Orellana-Martín, D., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: Membrane creation and symport/antiport rules solving QSAT. Journal of Membrane Computing **4**(3), 261–267 (2022)

[18] Riscos-Núñez, A., Valencia-Cabrera, L.: From SAT to SAT-UNSAT using P systems with dissolution rules. Journal of Membrane Computing **4**(2), 97–106 (2022)

[19] Aman, B.: On the efficiency of synchronized P systems. Journal of Membrane Computing **4**(1), 1–10 (2022)

[20] Atanasiu, A., Martín-Vide, C.: Arithmetic with membranes. In: Proc of the Workshop on Mutiset Processing, pp. 1–17 (2000)

[21] Bonchiş, C., Ciobanu, G., Izbaşa, C.: Encodings and arithmetic operations in membrane computing. In: International Conference on Theory and Applications of Models of Computation, pp. 621–630 (2006). Springer

[22] Gutiérrez-Naranjo, M.A., Leporati, A.: First steps towards a CPU made of spiking neural P systems. International Journal of Computers Communications & Control **4**(3), 244–252 (2009)

[23] Gutiérrez-Naranjo, M.A., Leporati, A.: Performing arithmetic operations with spiking neural P systems. Proc. of the Seventh Brainstorming Week on Membrane Computing **1**, 181–198

[24] Zhang, X., Zeng, X., Pan, L., Luo, B.: A spiking neural P system for performing multiplication of two arbitrary natural numbers. Chinese journal of computers **32**(12), 2362–2372 (2009)

[25] Zhang, X., Zeng, X., Pan, L.: Weighted spiking neural P systems with rules on synapses. Fundamenta Informaticae **134**(1-2), 201–218 (2014)

[26] Wang, H., Zhou, K., Zhang, G.: Arithmetic operations with spiking neural P systems with rules and weights on synapses. International Journal of Computers Communications & Control **13**(4), 574–589 (2018)

[27] Xu, Z., Cavaliere, M., An, P., Vrudhula, S., Cao, Y.: The stochastic loss of spikes in spiking neural P systems: Design and implementation of reliable arithmetic circuits. Fundamenta Informaticae **134**(1-2), 183–200 (2014)

[28] Zhang, G., Rong, H., Paul, P., He, Y., Neri, F., Pérez-Jiménez, M.J.: A complete arithmetic calculator constructed from spiking neural P systems and its application to information fusion. International Journal of Neural Systems **31**(01), 2050055 (2021)

[29] Leporati, A., Zandron, C., Mauri, G.: Solving the factorization problem with P systems. Progress in Natural Science **17**(4), 471–478 (2007)

[30] Orellana-Martín, D., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: The factorization problem: a new approach through membrane systems. In: UCNC 2018: 17th International Conference on Unconventional Computation and Natural Computation (2018), Pp. 39-56. (2018)

[31] Murakawa, T., Fujiwara, A.: Arithmetic operations and factorization using asynchronous P systems. International Journal of Networking and Computing **2**(2), 217–233 (2022)

[32] Ganbaatar, G., Nyamdorj, D., Cichon, G., Ishdorj, T.-O.: Implementation of RSA cryptographic algorithm using SN P systems based on HP/LP neurons. Journal of Membrane Computing **3**(1), 22–34 (2021)

[33] Obtułowicz, A.: On P systems with active membranes solving the integer factorization problem in a polynomial time. In: Workshop on Membrane Computing, pp. 267–285 (2000). Springer

[34] Wang, H., Zhou, K., Zhang, G., Paul, P., Duan, Y., Qi, H., Rong, H.: Application of weighted spiking neural P systems with rules on synapses for breaking RSA encryption. Int. J. Unconv. Comput. **15**(1-2), 37–58 (2020)

[35] Ramirez-de-Arellano, A., Orellana-Martin, D., Pérez-Jiménez, M.J.: Using virus machines to compute pairing functions [to appear]. International Journal of Neural Systems (2023)

[36] Gheorghe, M., Ipate, F., Dragomir, C., Mierlă, L., Valencia-Cabrera, L., García-Quismondo, M., Pérez-Jiménez, M.J.: Kernel P systems - Version I. Eleventh Brainstorming Week on Membrane Computing (11BWMC), 97–124 (2013)

[37] Gheorghe, M., Ipate, F., Lefticaru, R., Pérez-Jiménez, M.J., Ţurcanu, A., Valencia-Cabrera, L., García-Quismondo, M., Mierlă, L.: 3-Col problem modelling using simple kernel P systems. International Journal of Computer Mathematics **90**(4), 816–830 (2013)

[38] Ipate, F., Lefticaru, R., Mierlă, L., Valencia-Cabrera, L., Han, H., Zhang, G., Dragomir, C., Pérez-Jiménez, M.J., Gheorghe, M.: Kernel P systems: Applications and implementations. In: Proceedings of The Eighth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2013, pp. 1081–1089 (2013). Springer

[39] Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S., Lefticaru, R.: Kernel P systems: from modelling to verification and testing. Theoretical Computer Science **724**(9), 45–60 (2018)

[40] Syropoulos, A.: Mathematics of multisets. In: Multiset Processing: Mathematical, Computer Science, and Molecular Computing Points of View 1, pp. 347–358 (2001). Springer

[41] Vasile, R. https://github.com/Razvan-V/kPL

[42] Nicolescu, R., Ipate, F., Wu, H.: Towards high-level P systems programming using complex objects. In: International Conference on Membrane Computing, pp. 255–276 (2013)

[43] Konur, S., Mierlă, L., Ipate, F., Gheorghe, M.: kPWorkbench: A software suit for membrane systems. SoftwareX **11**, 100407 (2020). https://doi.org/10.1016/j.softx.2020.100407

[44] kPWorkbench. https://github.com/Kernel-P-Systems/kPWorkbench