Int. J. Communications, Network and System Sciences, **2012, 5, 154-164**
http://dx.doi.org/10.4236/ijcns.2012.53020 Published Online March 2012 (http://www.SciRP.org/journal/ijcns)

Cryptanalysis of a Substitution-Permutation Network Using Gene Assembly in Ciliates

Arash Karimi, Hadi Shahriar Shahhoseini*
Electrical Engineering Department, Iran University of Science and Technology, Tehran, Iran
Email: ar_karimi@elec.iust.ac.ir, *bhshsh@iust.ac.ir

Received November 26, 2011; revised January 12, 2012; accepted January 27, 2012

ABSTRACT

In this paper we provide a novel approach for breaking a significant class of block ciphers, the so-called SPN ciphers, using the process of gene assembly in ciliates. Our proposed scheme utilizes, for the first time, the Turing-powerful potential of gene assembly procedure of ciliated protozoa into the real world computations and has a fewer number of steps than the other proposed schemes to break a cipher. We elaborate notions of formal language theory based on AIR systems, which can be thought of as a modified version of intramolecular scheme to model the ciliate bio-operations, for construction of building blocks necessary for breaking the cipher, and based on these nature-inspired constructions which are as powerful as Turing machines, we propose a theoretical approach for breaking SPN ciphers. Then, we simulate our proposed plan for breaking these ciphers on a sample block cipher based on this structure. Our results show that the proposed scheme has 51.5 percent improvement over the best previously proposed nature-inspired scheme for breaking a cipher.

Keywords: Nature-Inspired Computation; Accepting Intramolecular Recombination (AIR) Systems; Cryptanalysis; Gene Assembly; Block Ciphers

1. Introduction

L. Adleman, during a laboratory experiment, for the first time discovered the potential of DNA molecules to solve computationally hard problems [1]. His revolutionary paper started the interdisciplinary area of DNA computing. Since then, a bulk of research has tried to concentrate on the theoretical ability of DNA strands to solve hard problems. Specifically, language theory has helped researchers find mathematical constructions to build computing machines based on ability of biomolecules represented in form of words. Natural computing which utilizes the potential of biomolecules in their living environments (*i.e.* cells) is of special interest. In this respect, Kari *et al.* in [2,3] considered the gene assembly process in ciliates and demonstrated that it has computational capability just like Turing machines. Their findings aroused a hot line of research in cellular computing.

Ciliates are single-celled eukaryotes that have special features which make them appealing and distinctive. They possess cilia which are used for their motion and also for making a current of water to sweep bacteria and other nutrients into their oral cavities. In addition, they have two different sorts of nuclei: A diploid micronucleus and a polyploid macronucleus. The former is germline nucleus which is activated only during the sexual process of conjugation and remains dormant in the vegetative cycle. And the latter is the somatic nucleus which is the housekeeping nucleus responsible for production of RNA transcripts which is a must for cell development during its life cycle. A species of ciliated protozoa Oxytricha trifallax is shown in **Figure 1**.

Ciliates do sorting, inversion and excision of their DNA sequences. We adopt the strategy of encoding all solution candidates into a micronuclear gene, then assembling the gene using intramolecular model [4] and ultimately filtering the results and checking through the cipher to find the right key.

In [5], Adleman *et al.* proposed a scheme to break the Data Encryption Standard using DNA molecules, by molecular biology tools.

In this paper we want to replace formal biological operations by ciliate bio-operations for cryptanalysis of SPN ciphers. For this reason, we use language-theoretic notions to describe the process of cryptanalysis and by utilizing an encoding scheme of the words of our constructed notion to the MIC genes of a hypothetical ciliated protozoa from the Stichotrichous family as shown in **Figure 2**, we design AIR systems which simulate different blocks necessary to do the cryptanalysis of a large class of block ciphers, called substitution-permutation

*Corresponding author.

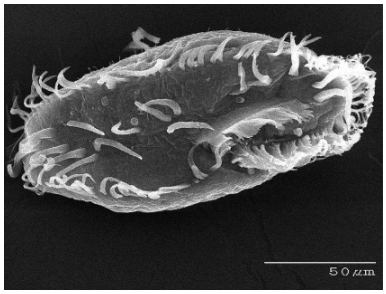

Figure 1. Ciliated protozoa Oxytrichia trifallax.

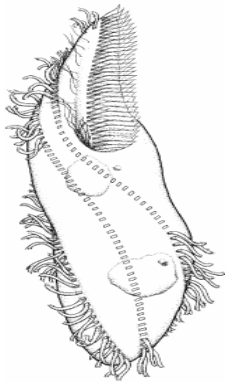

Figure 2. Ciliated protozoa Stichotricha.

networks and then using these Turing machine constructions, we simulate our theoretical attack on this structure defined by specific and predefined blocks.

The rest of this paper is organized as follows. In Section 2, the substitution-permutation networks are introduced which constitute a large class of block ciphers, in Section 3.1, the concept of splicing schemes that is necessary to understand AIR systems which is a variant of intramolecular operations for modeling ciliate bio-operations is briefly introduced. In Section 3.2, we define the accepting intramolecular recombination systems (or AIR systems) on which our proposed scheme to break the cipher is based. In Section 4, we propose and build the necessary blocks which we need for cryptanalysis of the cipher. In Section 5, based on our previously designed AIR systems, we devise a theoretical approach to attack the cipher. In Section 6, we evaluate the performance of our proposed scheme and derive the total bio-steps necessary to mount the attack. In Section 7, our simulations are discussed and the results are reported. Finally, in Section 8, we summarize our paper and conclusions are drawn, and the plans for the future research based on this work are presented.

2. Substitution-Permutation Networks

A variety of modern block ciphers are built using an iterative structure of Substitution-Permutation Networks or SPN for short. AES (Rijndael), Shark, Khazad and Anubis are good examples for SPN ciphers [6]. The selected example SPN is as shown in **Figure 3** and we will focus our discussion on this network. As demonstrated in **Figure 3**, the block size of our cipher is 16-bits and each block of the plaintext is processed by repeating basic operations of a round which are substitution, permutation and key mixing. Indeed, our considered scheme is similar to what is found in many modern block ciphers including Rijndael from basic operations viewpoint and provides us with an insight into cryptanalysis of the real-world block ciphers using natural computing methods.

3. Preliminary Definitions

Our proposed scheme to break the cipher is based on Accepting intramolecular recombination systems (AIR systems) which is a variant of intramolecular models of gene assembly in ciliated protozoa. In this section we bring some basic notions and notations that are necessary to conceive the attack procedure.

3.1. Splicing Schemes

A splicing scheme [7] is defined as a pair $R=(\Sigma,\sim)$ in which Σ is an alphabet and ~ is a binary relation over $(\Sigma^*\Sigma\Sigma^*)^2$. Assuming that this relation exists between two triples of strings as follows $(\alpha,p,\beta)\sim(\alpha',p,\beta')$ we say that given the abovementioned binary relation, strings $z_1=x'\alpha p\beta' y''$ and $z_2=y'\alpha' p\beta x''$ can be obtained by recombination from $x=x'\alpha p\beta x''$ and $y=y'\alpha' p\beta' y''$.

3.2. AIR Systems

An accepting intramolecular recombination system is defined as a quadruple $G=(\Sigma,\sim,\alpha_{in},\beta)$ in which $R=(\Sigma,\sim)$ is the splicing scheme and α_{in} and β are input and the target words, respectively. Considering a splicing scheme, $R=(\Sigma,\sim)$, we define the contextual intramolecular operations of translocation, *trl*, and deletion, *del*, which are generalizations of dlad and ld intramolecular operations, respectively, as follows [8].

Assuming $x',u',u'',v',x'',y',y'',z'\in\Sigma^*$ and $x=x'\alpha$, $uqy=\beta u'=u''\alpha'$, $vqz=\beta' v'$, $xpu=x''\gamma$, $ypv=\delta y'=y''\gamma'$, $z=\delta' z'$ The *trl* operation with respect to *R* is defined as

$$(trl_{p,q})\,xpuqypvqz \Rightarrow_{trl_{p,q}} xpvqypuqz \qquad (1)$$

Therefore, in the $trl_{p,q}$ operation the strings of *u* and *v* which are flanked by pointers *p* and *q* are swapped. The *del* operation with respect to *R* is also defined as:

$$(del_p)\,xpupy \Rightarrow_{del_p} xpy \qquad (2)$$

where $(\alpha,p,\beta)\sim(\alpha',p,\beta')$ and for $x',u',u'',\beta'\in\Sigma^*$, $x=x'\alpha$, $u=\beta u'=u''\alpha'$, $y=\beta' y'$. Hence, intuitively,

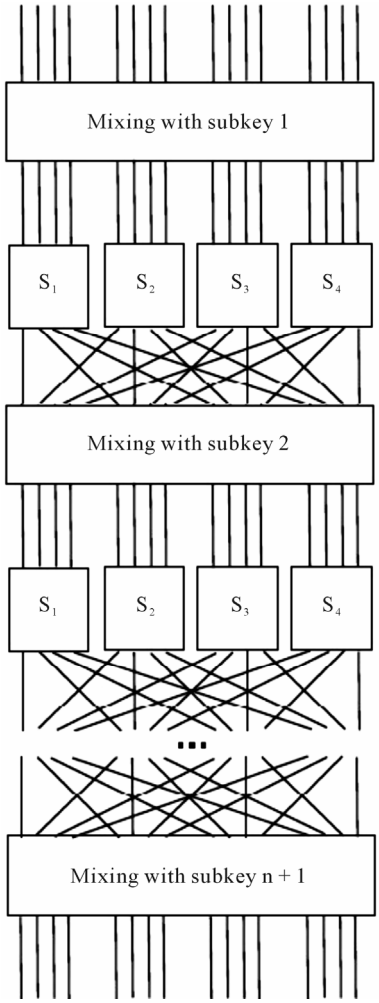

Figure 3. The basic substitution-permutation network.

if the contexts of occurrences p are in the relation ~, the del_p operation removes the string u that is flanked by two occurrences of p. We can now define the set $\tilde{R}$ of all contextual intramolecular operations under guidance of ~ as:

$$\tilde{R} = \left\{ trl_{p,q}, del_m \middle| \, p,q,m \in \Sigma \right\} \tag{3}$$

Accordingly, we can define the language that is accepted by the AIR system, G, as all the words $w \in \Sigma^*$ for which by consecutive application of any number of $op \in \tilde{R}$ operations from $\alpha_{in} w$, we can get to the target word β.

4. Constructing Necessary Building Blocks for Attacking the Cipher

Our proposed scheme to break the SPN family of block ciphers considers the modified approach of intramolecular recombination for modeling gene assembly process in ciliates, and then applies this approach in constructing necessary building blocks for breaking the cipher. Our attack approach is brute force in which we assume that we possess a (plaintext, ciphertext) pair and by exhaustive search over all possible keys, we aim to find the correct key of the cipher. Therefore, in the first step, we should produce all genes of a hypothetical ciliate each of which codes for an individual key of the cipher and then using gene assembly process that naturally happens in ciliated protozoa, we construct Turing machines that imitate main operations that we need in the procedure of cryptanalysis such as the substitution, permutation and logical XOR, and ultimately, we can find the key that when mixed with the plaintext, gives the ciphertext if in each step of the computation the micronuclear (MIC) genes are assembled to the expected macronuclear (MAC) genes. In the next section we introduce the main operations needed for cryptanalysis and then, build Turing machines that imitate these operations.

4.1. Generation of All Possible Keys

In order for generation of all genes that code for all possible combinations of the key, we utilize the graph of **Figure 4** in which all possible paths that start from a_b and terminate at a_e code for a different n-bit key. We use intramolecular model of gene assembly in ciliates. Therefore, beginning with a single MIC gene pattern for which there exist more than two occurrences of a pointer, we can assemble different MAC gene patterns that code for different keys of the cipher. Now, assuming that graph of **Figure 4** is demonstrated with $G = (V, E)$ for which V and E denote sets of vertices and edges, respectively, we define an encoding of G in the MIC gene pattern in terms of MDS descriptors as follows: we associate a pointer p to each vertex of G and to each directed edge (p,q) we associate MDS of (p,q). Therefore, a path $pp_1p_2\cdots p_nq$ of G can be encoded by an intermediate MDS $(p, p_1p_2\cdots p_k, q)$ in which p and q are the incoming and outgoing pointers, respectively, and $p_i's$, $1 \le i \le k$ are those pointers for which MIC MDSs that correspond with edges $(p,p_1),(p_1,p_2),\cdots,(p_k,q)$ belonging to set of edges of G, have been spliced. Note that MIC MDSs are spliced on their common pointers. For our graph of representation of all possible keys (graph of **Figure 4**) we have

$$V = \left\{ a_b, a_1, b_1, b_1', \cdots, a_n, b_n, b_n', a_{n+1}, a_e \right\}$$

and

$$E = \left\{ (a_b, a_1), (a_1, b_1), (a_1, b_1'), (b_1, a_2), (b_1', a_2), \cdots, (a_{n+1}, a_e) \right\}$$

hence, for instance, we associate an MDS $(a_b, a_1b_1a_2b_2'\cdots a_{n+1}, a_e)$ to a path $a_ba_1b_1a_2b_2'\cdots a_{n+1}a_e$ and with different strategies to the same MIC gene pattern, we can obtain different MAC genes which represent all

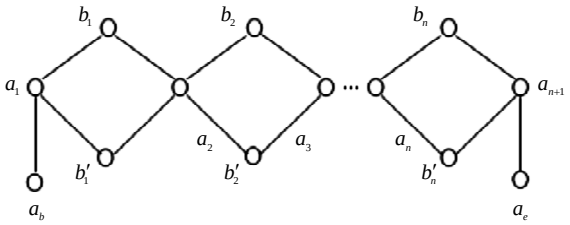

Figure 4. The graph for generation of all possible keys.

possible keys. According to the universality result of [9] for intramolecular operations, each path of *G* can be assembled using intramolecular ciliate operations *ld*, *hi* and *dlad*. Since we are looking for all those paths that start from a_b and end at a_e, our desired assembled MAC gene would be of the form, (a_b, u, a_e), in which *u* is any path that contains all a_i's $(1 \le i \le n+1)$ and all different b_i's or b_i''s $(1 \le i \le n)$. Therefore, if each edge of *G* is encoded as a MIC MDS then, we show that one can produce all possible keys of the cipher using *ld* operation only from a single string of MDS descriptors that demonstrate all edges of *G* as shown in Theorem 1. For this reason, we say that a descriptor φ_G is associated to *G* if it is of the form $\varphi_G^{ld} = (a_b, \Lambda, a_1)\alpha_G^{2n}(a_{n+1}, \Lambda, a_e)$ where $\alpha_G = \prod_{(p,q)\in E}(p, \Lambda, q)$ is defined as a descriptor that encodes all edges of *G*.

Theorem 1. Any successfully assembled $MDS \in \ell_{ld}(\varphi_G^{ld})$ for which all $n+1$ a_i's and a total number of *n* of either b_i's or b_i''s appear and $|MDS| = |V| + 2$, produces any possible path that is representative for every one of possible keys of the cipher.

Implementation of logical XOR using ciliate bio operations is explained in Section 4.2. $|MDS| = |V| + 2$, produces any possible path that is representative for every one of possible keys of the cipher.

Implementation of logical XOR using ciliate bio operations is explained in Section 4.2.

4.2. An Intramolecular Model for Computing Logical XOR

We define logical XOR as satisfiability of a Boolean relationship that is depicted in Equation (4).

$$C = (x \wedge \bar{y}) \vee (\bar{x} \wedge y) \tag{4}$$

And here we provide a ciliate solution for Equation (4) using intramolecular model which computes the result of XOR in polynomial time. For this reason, we construct an AIR system $G = (\Sigma, \sim, \alpha_0, YES)$ [9] that evaluates the output of Equation (4). The circuit that implements XOR is depicted in **Figure 5**.

As can be seen in **Figure 5**, the circuit is composed of three layers and we show its output with the following notation demonstrated in Equation (5).

$$c = \left({}_{c_{31}}\langle {}_{c_{21}} x[{}_{c_{11}} y]_{c_{11}} \rangle_{c_{21}} \langle {}_{c_{22}}[{}_{c_{12}} x]_{c_{12}} y \rangle_{c_{22}} \right)_{c_{31}} \tag{5}$$

In the above notation, each gate is shown with c_{ij} in which *i* denotes number of layer of the circuit and *j* denotes number of that gate in the i^{th} layer and $\langle\ \rangle$ and $(\ )$ and $[\ ]$ denote logical AND, OR and NOT gates, respectively. In this circuit we have $C = (V, E)$ where $V = \{c_{11}, c_{12}, c_{21}, c_{22}, c_{31}\}$ and $E = \{(c_{11}, c_{21}), (c_{12}, c_{22}), (c_{21}, c_{31}), (c_{22}, c_{31})\}$. Now we construct an AIR system for the Boolean circuit of **Figure 5** as $G_c = (V', \sim, YES)$ where

$$V' = V \cup \{X, Y\} \cup \{T, F, Y, E, S, \Xi, \$\} \cup \{[_{c_{11}},]_{c_{11}}, [_{c_{12}},]_{c_{12}}, \langle_{c_{21}}, \rangle_{c_{21}}, \langle_{c_{22}}, \rangle_{c_{22}}, (_{c_{31}},)_{c_{31}}, \Xi_{11}, \Xi_{12}, \Xi_{21}, \Xi_{22}, \Xi_{31}\}$$

YES is the axiom and $\$$ and Ξ are not included in alphabet of the Turing machine. Furthermore, $(V', \sim)$ is the splicing scheme. Assuming that F is used instead of logical *zero* and T is used instead of logical *one*, *x* and *y* are encoded into inputs for G_c by utilizing the mapping $\Phi(i) \to \{T, F\}$ in which $i = \{x, y\}$ is the set of inputs of the circuit. Now, if we consider Equations (6) and (7):

$$\vartheta_1 = \Xi\ ({}_{c_{31}}\ \Xi_{31} \Xi \langle {}_{c_{21}}\ \Xi_{21} \Phi(x) \Xi\ [{}_{c_{11}}\ \Xi_{11} \Phi(y) \Xi_{11}]_{c_{11}}\ \Xi\Xi_{21} \rangle_{c_{21}}\ \Xi\Xi \langle {}_{c_{22}}\ \Xi_{22} \Xi\ [{}_{c_{12}}\ \Xi_{12} \Phi(x) \Xi_{12}]_{c_{12}}\ \Xi \Phi(y) \Xi_{22} \rangle_{c_{22}}\ \Xi\Xi_{31})_{c_{31}}\ \Xi \tag{6}$$

$$\begin{aligned} \vartheta_2 = {} & \$\,[_{c_{11}} \Phi(\bar{y})]_{c_{11}}\ \$\,[_{c_{11}} \bar{\Phi}(\bar{y})]_{c_{11}}\ \$\,[_{c_{12}} \Phi(\bar{x})]_{c_{12}}\ \$\,[_{c_{12}} \bar{\Phi}(\bar{x})]_{c_{12}}\ \$\,\langle_{c_{21}} \Phi(x \wedge \bar{y}) \rangle_{c_{21}}\ \$\,\langle_{c_{21}} \bar{\Phi}(x \wedge \bar{y}) \rangle_{c_{21}} \\ & \$ \langle_{c_{22}} \Phi(\bar{x} \wedge y) \rangle_{c_{22}}\ \$ \langle_{c_{22}} \bar{\Phi}(\bar{x} \wedge y) \rangle_{c_{22}}\ \$ (_{c_{31}} \Phi((x \wedge \bar{y}) \vee (\bar{x} \wedge y)))_{c_{31}}\ \$ (_{c_{31}} \bar{\Phi}((x \wedge \bar{y}) \vee (\bar{x} \wedge y)))_{c_{31}}\ \$ \end{aligned} \tag{7}$$

The input of our AIR system (G_c) is then $\alpha_0 = YE\vartheta_1\vartheta_2 ES$ and G_c accepts the input string if result of XOR is 1 and the axiom *YES* is produced. Therefore if result of XOR equals 1, G_c accepts input string in 4 steps. In the following we construct splicing schemes necessary for computation of XOR in the proposed AIR system.

$$(\Xi, [_{c_{11}}, \Xi_{11} u_{11} \Xi_{11}) \sim (\$, [_{c_{11}}, T),\ T, \Xi_{ij} \notin Sub(u_{11}) \tag{8}$$

$$(\Xi_{11} u_{11} \Xi_{11},]_{c_{11}}, \Xi) \sim (T,]_{c_{11}}, \$),\ T, \Xi_{ij} \notin Sub(u_{11}) \tag{9}$$

$$\begin{aligned} & (\Xi, [_{c_{11}}, \Xi_{11} u_{11} \Xi_{11}) \sim (\$, [_{c_{11}}, F), \\ & T \in Sub(u_{11}), \Xi_{ij} \notin Sub(u_{11}) \end{aligned} \tag{10}$$

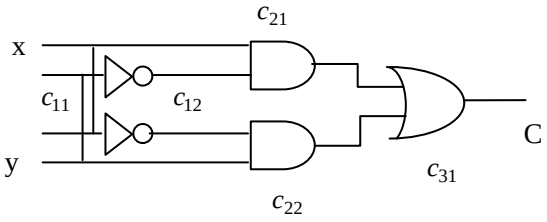

Figure 5. Operation of XOR.

$$\left(\Xi_{11}u_{11}\Xi_{11},]_{c_{11}},\Xi\right)\sim\left(F,]_{c_{11}},\$\right),\quad T\in Sub\left(u_{11}\right),\Xi_{ij}\notin Sub\left(u_{11}\right) \tag{11}$$

$$\left(\Xi,[_{c_{12}},\Xi_{12}u_{12}\Xi_{12}\right)\sim\left(\$,[_{c_{12}},T\right),\ T,\Xi_{ij}\notin Sub\left(u_{12}\right) \tag{12}$$

$$\left(\Xi_{12}u_{12}\Xi_{12},]_{c_{12}},\Xi\right)\sim\left(T,]_{c_{12}},\$\right),\ T,\Xi_{ij}\notin Sub\left(u_{12}\right) \tag{13}$$

$$\left(\Xi,[_{c_{12}},\Xi_{12}u_{12}\Xi_{12}\right)\sim\left(\$,[_{c_{12}},F\right),\quad T\in Sub\left(u_{12}\right),\Xi_{ij}\notin Sub\left(u_{12}\right) \tag{14}$$

$$\left(\Xi_{12}u_{12}\Xi_{12},]_{c_{12}},\Xi\right)\sim\left(F,]_{c_{12}},\$\right),\quad T\in Sub\left(u_{12}\right),\Xi_{ij}\notin Sub\left(u_{12}\right) \tag{15}$$

$$\left(\Xi,\langle_{c_{21}},\Xi_{21}u_{21}\Xi_{21}\right)\sim\left(\$,\langle_{c_{21}},T\right),\ F\notin Sub\left(u_{21}\right) \tag{16}$$

$$\left(\Xi_{21}u_{21}\Xi_{21},\rangle_{c_{21}},\Xi\right)\sim\left(T,\rangle_{c_{21}},\$\right),\ F\notin Sub\left(u_{21}\right) \tag{17}$$

$$\left(\Xi,\langle_{c_{21}},\Xi_{21}u_{21}\Xi_{21}\right)\sim\left(\$,\langle_{c_{21}},F\right),\ F\in Sub\left(u_{21}\right) \tag{18}$$

$$\left(\Xi_{21}u_{21}\Xi_{21},\rangle_{c_{21}},\Xi\right)\sim\left(F,\rangle_{c_{21}},\$\right),\ F\in Sub\left(u_{21}\right) \tag{19}$$

$$\left(\Xi,\langle_{c_{22}},\Xi_{22}u_{22}\Xi_{22}\right)\sim\left(\$,\langle_{c_{22}},T\right),\ F\notin Sub\left(u_{22}\right) \tag{20}$$

$$\left(\Xi_{22}u_{22}\Xi_{22},\rangle_{c_{22}},\Xi\right)\sim\left(T,\rangle_{c_{22}},\$\right),\ F\notin Sub\left(u_{22}\right) \tag{21}$$

$$\left(\Xi,\langle_{c_{22}},\Xi_{22}u_{22}\Xi_{22}\right)\sim\left(\$,\langle_{c_{22}},F\right),\ F\in Sub\left(u_{22}\right) \tag{22}$$

$$\left(\Xi_{22}u_{22}\Xi_{22},\rangle_{c_{22}},\Xi\right)\sim\left(F,\rangle_{c_{22}},\$\right),\ F\in Sub\left(u_{22}\right) \tag{23}$$

$$\left(\Xi,(_{c_{31}},\Xi_{31}u_{31}\Xi_{31}\right)\sim\left(\$,(_{c_{31}},T\right)\quad T\in Sub\left(u_{31}\right),\Xi_{ij}\notin Sub\left(u_{31}\right) \tag{24}$$

$$\left(\Xi_{31}u_{31}\Xi_{31},)_{c_{31}},\Xi\right)\sim\left(T,)_{c_{31}},\$\right)\quad T\in Sub\left(u_{31}\right),\Xi_{ij}\notin Sub\left(u_{31}\right) \tag{25}$$

$$\left(\Xi,(_{c_{31}},\Xi_{31}u_{31}\Xi_{31}\right)\sim\left(\$,(_{c_{31}},T\right),\ T,\Xi_{ij}\notin Sub\left(u_{31}\right) \tag{26}$$

$$\left(\Xi_{31}u_{31}\Xi_{31},)_{c_{31}},\Xi\right)\sim\left(T,)_{c_{31}},\$\right),\ T,\Xi_{ij}\notin Sub\left(u_{31}\right) \tag{27}$$

$$\left(Y,E,\Xi\,(_{c_{31}}T)_{c_{31}}\,\Xi\right)\sim\left(\$,E,S\right),\ \text{if result is one} \tag{28}$$

Furthermore, in Equations (8)-(28) $u_{ij},1\le i,\ j\le 3$ are defined as shown in Equation (29).

$$u_{ij}=\begin{cases}\Phi\left(y\right), & i,j=1\\ \Phi\left(x\right), & i=1,j=2\\ \Xi[_{c_{11}}\Phi\left(y\right)]_{c_{11}}\Xi\Phi\left(x\right), & i=2,j=1\\ \Xi[_{c_{12}}\Phi\left(x\right)]_{c_{12}}\Xi\Phi\left(y\right), & i=2,j=2\\ \Xi\langle_{c_{21}}\Xi[_{c_{11}}\Phi\left(y\right)]_{c_{11}}\Xi\Phi\left(x\right)\rangle_{c_{21}}\Xi\Xi\langle_{c_{22}}\Xi[_{c_{12}}\Phi\left(x\right)]_{c_{12}}\Xi\Phi\left(y\right)\rangle_{c_{22}}\Xi, & i=3,j=1\end{cases} \tag{29}$$

and the computed words in each step of computation can be written as Equations (30)-(33):

$$\begin{aligned}\alpha_0=\ &YE\Xi(_{c_{31}}\Xi_{31}\Xi\langle_{c_{21}}\Xi_{21}\Phi\left(x\right)\Xi[_{c_{11}}\Xi_{11}\Phi\left(y\right)\Xi_{11}]_{c_{11}}\Xi\Xi_{21}\rangle_{c_{21}}\Xi\Xi\langle_{c_{22}}\Xi_{22}\Xi[_{c_{12}}\Xi_{12}\Phi\left(x\right)\Xi_{12}]_{c_{12}}\Xi\Phi\left(y\right)\Xi_{22}\rangle_{c_{22}}\Xi\Xi_{31})_{c_{31}}\\ &\Xi\$[_{c_{11}}\Phi\left(\overline{y}\right)]_{c_{11}}\$[_{c_{11}}\overline{\Phi}\left(\overline{y}\right)]_{c_{11}}\$[_{c_{12}}\Phi\left(\overline{x}\right)]_{c_{12}}\$[_{c_{12}}\overline{\Phi}\left(\overline{x}\right)]_{c_{12}}\$\langle_{c_{21}}\Phi\left(x\wedge\overline{y}\right)\rangle_{c_{21}}\$\langle_{c_{21}}\overline{\Phi}\left(x\wedge\overline{y}\right)\rangle_{c_{21}}\$\langle_{c_{22}}\Phi\left(\overline{x}\wedge y\right)\rangle_{c_{22}}\\ &\$\langle_{c_{22}}\overline{\Phi}\left(\overline{x}\wedge y\right)\rangle_{c_{22}}\$(_{c_{31}}\Phi\left(\left(x\wedge\overline{y}\right)\vee\left(\overline{x}\wedge y\right)\right))_{c_{31}}\$(_{c_{31}}\overline{\Phi}\left(\left(x\wedge\overline{y}\right)\vee\left(\overline{x}\wedge y\right)\right))_{c_{31}}\$ES\end{aligned} \tag{30}$$

$$\begin{aligned}\alpha_1=\ &YE\Xi(_{c_{31}}\Xi_{31}\Xi\langle_{c_{21}}\Xi_{21}\Phi\left(x\right)\Xi[_{c_{11}}\Phi\left(\overline{y}\right)]_{c_{11}}\Xi\Xi_{21}\rangle_{c_{21}}\Xi\Xi\langle_{c_{22}}\Xi_{22}\Xi[_{c_{12}}\Phi\left(\overline{x}\right)]_{c_{12}}\Xi\Phi\left(y\right)\Xi_{22}\rangle_{c_{22}}\Xi\Xi_{31})_{c_{31}}\\ &\Xi\$[_{c_{11}}\Xi_{11}\Phi\left(y\right)\Xi_{11}]_{c_{11}}\$[_{c_{11}}\overline{\Phi}\left(\overline{y}\right)]_{c_{11}}\$[_{c_{12}}\Xi_{12}\Phi(x)\Xi_{12}]_{c_{12}}\$[_{c_{12}}\overline{\Phi}\left(\overline{x}\right)]_{c_{12}}\$\langle_{c_{21}}\Phi\left(x\wedge\overline{y}\right)\rangle_{c_{21}}\$\langle_{c_{21}}\overline{\Phi}\left(x\wedge\overline{y}\right)\rangle_{c_{21}}\\ &\$\langle_{c_{22}}\Phi\left(\overline{x}\wedge y\right)\rangle_{c_{22}}\$\langle_{c_{22}}\overline{\Phi}\left(\overline{x}\wedge y\right)\rangle_{c_{22}}\$(_{c_{31}}\Phi\left(\left(x\wedge\overline{y}\right)\vee\left(\overline{x}\wedge y\right)\right))_{c_{31}}\$(_{c_{31}}\overline{\Phi}\left(\left(x\wedge\overline{y}\right)\vee\left(\overline{x}\wedge y\right)\right))_{c_{31}}\$ES\end{aligned} \tag{31}$$

$$\begin{aligned}\alpha_2=\ &YE\Xi(_{c_{31}}\Xi_{31}\Xi\langle_{c_{21}}\Phi\left(x\wedge\overline{y}\right)\rangle_{c_{21}}\Xi\Xi\langle_{c_{22}}\Phi\left(\overline{x}\wedge y\right)\rangle_{c_{22}}\Xi\Xi_{31})_{c_{31}}\Xi\$[_{c_{11}}\Xi_{11}\Phi\left(y\right)\Xi_{11}]_{c_{11}}\$[_{c_{11}}\overline{\Phi}\left(\overline{y}\right)]_{c_{11}}\\ &\$[_{c_{12}}\Xi_{12}\Phi\left(x\right)\Xi_{12}]_{c_{12}}\ \ \$[_{c_{12}}\overline{\Phi}\left(\overline{x}\right)]_{c_{12}}\$\langle_{c_{21}}\Xi_{21}\Phi\left(x\right)\Xi[_{c_{11}}\Phi\left(\overline{y}\right)]_{c_{11}}\Xi\Xi_{21}\rangle_{c_{21}}\$\langle_{c_{21}}\overline{\Phi}\left(x\wedge\overline{y}\right)\rangle)\rangle_{c_{21}}\\ &\$\langle_{c_{22}}\Xi_{22}\Xi[_{c_{12}}\Phi\left(\overline{x}\right)]_{c_{12}}\Xi\Phi\left(y\right)\Xi_{22}\rangle_{c_{22}}\$\langle_{c_{22}}\overline{\Phi}\left(\overline{x}\wedge y\right)\rangle_{c_{22}}\$(_{c_{31}}\Phi\left(\left(x\wedge\overline{y}\right)\vee\left(\overline{x}\wedge y\right)\right))_{c_{31}}\\ &\$(_{c_{31}}\overline{\Phi}\left(\left(x\wedge\overline{y}\right)\vee\left(\overline{x}\wedge y\right)\right))_{c_{31}}\$ES\end{aligned} \tag{32}$$

$$\begin{aligned}\alpha_3 = YE\Xi(_{c_{31}}\Phi\big((x\wedge\bar{y})\vee(\bar{x}\wedge y)\big))_{c_{31}}\Xi\$[_{c_{11}}\Xi_{11}\Phi(y)\Xi_{11}]_{c_{11}}\$[_{c_{11}}\bar{\Phi}(\bar{y})]_{c_{11}}\$[_{c_{12}}\Xi_{12}\Phi(x)\Xi_{12}]_{c_{12}}\$[_{c_{12}}\bar{\Phi}(\bar{x})]_{c_{12}}\\ \$\langle_{c_{21}}\Xi_{21}\Phi(x)\Xi[_{c_{11}}\Phi(\bar{y})]_{c_{11}}\Xi\Xi_{21}\rangle_{c_{21}}\$\langle_{c_{21}}\bar{\Phi}(x\wedge\bar{y})\rangle_{c_{21}}\$\langle_{c_{22}}\Xi_{22}\Xi[_{c_{12}}\Phi(\bar{x})]_{c_{12}}\Xi\Phi(y)\Xi_{22}\rangle_{c_{22}}\$\langle_{c_{22}}\bar{\Phi}(x\wedge\bar{y})\rangle_{c_{22}}\\ \$(_{c_{31}}\Xi_{31}\Xi\langle_{c_{21}}\Phi(x\wedge\bar{y})\rangle_{c_{21}}\Xi\Xi\langle_{c_{22}}\Phi(\bar{x}\wedge y)\rangle_{c_{22}}\Xi\Xi_{31})_{c_{31}}\$(_{c_{31}}\bar{\Phi}\big((x\wedge\bar{y})\vee(\bar{x}\wedge y)\big))_{c_{31}}\$ES\end{aligned} \quad (33)$$

In the next step, by applying deletion operation to α_2, if the result of the computation equals *one*, the axiom *YES* will be produced.

4.3. Simulation of S-Box with AIR Systems

We assume the word of Equation (34) as input to S-boxes of **Figure 3**

$$\begin{aligned}\alpha_{S_i} = OK\Xi\{_{1i}x_{1i}\}_{1i}\Xi\{_{2i}x_{2i}\}_{2i}\Xi\{_{3i}x_{3i}\}_{3i}\\ \Xi\{_{4i}x_{4i}\}_{4i}\Xi\$\{_{1i}T\}_{1i}\$\{_{1i}F\}_{1i}\$\{_{2i}T\}_{2i}\\ \$\{_{2i}F\}_{2i}\$\{_{3i}T\}_{3i}\$\{_{3i}F\}_{3i}\$\{_{4i}T\}_{4i}\\ \$\{_{4i}F\}_{4i}\$K!,(1\le i\le 4)\end{aligned} \quad (34)$$

The following splicing rules will be used to guide recombinations of our AIR system for simulating S-boxes.

$$(\Xi,\{_{1i},,x_{1i}\})\sim(\$,\{_{1i},T) \text{ if } x_{1i}\to T \text{ in } LUT \text{ of } S_i \quad (35)$$

$$(x_{1i},\}_{1i},\Xi)\sim(T,\}_{1i},\$) \text{ if } x_{1i}\to T \text{ in } LUT \text{ of } S_i \quad (36)$$

$$(\Xi,\{_{1i},,x_{1i}\})\sim(\$,\{_{1i},F) \text{ if } x_{1i}\to F \text{ in } LUT \text{ of } S_i \quad (37)$$

$$(x_{1i},\}_{1i},\Xi)\sim(F,\}_{1i},\$) \text{ if } x_{1i}\to F \text{ in } LUT \text{ of } S_i \quad (38)$$

$$(\Xi,\{_{2i},,x_{2i}\})\sim(\$,\{_{2i},T) \text{ if } x_{2i}\to T \text{ in } LUT \text{ of } S_i \quad (39)$$

$$(x_{2i},\}_{2i},\Xi)\sim(T,\}_{2i},\$) \text{ if } x_{2i}\to T \text{ in } LUT \text{ of } S_i \quad (40)$$

$$(\Xi,\{_{2i},,x_{2i}\})\sim(\$,\{_{2i},F) \text{ if } x_{2i}\to F \text{ in } LUT \text{ of } S_i \quad (41)$$

$$(x_{2i},\}_{2i},\Xi)\sim(F,\}_{2i},\$) \text{ if } x_{2i}\to F \text{ in } LUT \text{ of } S_i \quad (42)$$

$$(\Xi,\{_{3i},,x_{3i}\})\sim(\$,\{_{3i},T) \text{ if } x_{3i}\to T \text{ in } LUT \text{ of } S_i \quad (43)$$

$$(x_{3i},\}_{3i},\Xi)\sim(T,\}_{3i},\$) \text{ if } x_{3i}\to T \text{ in } LUT \text{ of } S_i \quad (44)$$

$$(\Xi,\{_{13},,x_{3i}\})\sim(\$,\{_{3i},F) \text{ if } x_{3i}\to F \text{ in } LUT \text{ of } S_i \quad (45)$$

$$(x_{3i},\}_{3i},\Xi)\sim(F,\}_{3i},\$) \text{ if } x_{3i}\to F \text{ in } LUT \text{ of } S_i \quad (46)$$

$$(\Xi,\{_{4i},,x_{4i}\})\sim(\$,\{_{4i},T) \text{ if } x_{4i}\to T \text{ in } LUT \text{ of } S_i \quad (47)$$

$$(x_{4i},\}_{4i},\Xi)\sim(T,\}_{4i},\$) \text{ if } x_{4i}\to T \text{ in } LUT \text{ of } S_i \quad (48)$$

$$(\Xi,\{_{4i},,x_{4i}\})\sim(\$,\{_{4i},F) \text{ if } x_{4i}\to F \text{ in } LUT \text{ of } S_i \quad (49)$$

$$(x_{4i},\}_{4i},\Xi)\sim(F,\}_{4i},\$) \text{ if } x_{4i}\to F \text{ in } LUT \text{ of } S_i \quad (50)$$

As can be seen in splicing relations of S-boxes, *trl* operation is applied to the input word in a parallel fashion in accordance with the look-up table of the corresponding S-box and therefore, all input bits are assigned a *True* or a *False* value at the same time.

After applying the above splicing rules, the following word is obtained in which we assumed that the 4-bits of input in S-box *i* are all mapped to logical *one* or *T*. Other mappings can be defined as well according to the look-up table of each S-box.

$$\begin{aligned}\beta_{S_i} = OK\Xi\{_{1i}T\}_{1i}\Xi\{_{2i}T\}_{2i}\Xi\{_{3i}T\}_{3i}\Xi\{_{4i}T\}_{4i}\\ \Xi\$\{_{1i}x_{1i}\}_{1i}\$\{_{1i}F\}_{1i}\$\{_{2i}x_{2i}\}_{2i}\$\{_{2i}F\}_{2i}\$\{_{3i}x_{3i}\}_{3i}\\ \$\{_{3i}F\}_{3i}\$\{_{4i}x_{4i}\}_{4i}\$\{_{4i}F\}_{4i}\$K!,\ (1\le i\le 4)\end{aligned} \quad (51)$$

Then, the following splicing scheme is applied to the resultant word. Then, the following splicing scheme is applied to the resultant word.

$$(O,K,\Xi\{_{1i}u_{1i}\}_{1i}\Xi\{_{2i}u_{2i}\}_{2i}\Xi\{_{3i}u_{3i}\}_{3i}\Xi\{_{4i}u_{4i}\}_{4i}\Xi)\sim(\$,K,!) \quad (52)$$

After applying the above rule, a deletion rule (which is based on *ld* operation) is applied as follows.

$$\beta_{S_i}\Rightarrow_{del_K}\alpha_2 \quad (53)$$

Therefore, axiom *OK*! Is produced which implies that the S-box has been calculated.

The start sequence for calculation of the substituted words for computing S-box can be written as shown in Equation (54), in which p and β' are assumed blank symbols.

$$\begin{aligned}\alpha_{S_i} = \Xi\{_{1i}x_{1i}\}_{1i}\Xi\{_{2i}x_{2i}\}_{2i}\Xi\{_{3i}x_{3i}\}_{3i}\Xi\{_{4i}x_{4i}\}_{4i}\\ \Xi p\$\{_{1i}T\}_{1i}\$\{_{1i}F\}_{1i}\$\{_{2i}T\}_{2i}\$\{_{2i}F\}_{2i}\$\{_{3i}T\}_{3i}\\ \$\{_{3i}F\}_{3i}\$\{_{4i}T\}_{4i}\$\{_{4i}F\}_{4i}\$p\beta'\end{aligned} \quad (54)$$

After applying the splicing rules of Equations (35)-(50) to α_{S_i} we can obtain β_{S_i} as written in Equation (55).

$$\begin{aligned}\beta_{S_i} = \Xi\{_{1i}T\}_{1i}\Xi\{_{2i}T\}_{2i}\Xi\{_{3i}T\}_{3i}\Xi\{_{4i}T\}_{4i}\\ \Xi p\$\{_{1i}x_{1i}\}_{1i}\$\{_{1i}F\}_{1i}\$\{_{2i}x_{2i}\}_{2i}\$\{_{2i}F\}_{2i}\\ \$\{_{3i}x_{3i}\}_{3i}\$\{_{3i}F\}_{3i}\$\{_{4i}x_{4i}\}_{4i}\$\{_{4i}F\}_{4i}\$p\beta'\end{aligned} \quad (55)$$

Now, we can obtain the output of S-boxes by applying a deletion rule which is guided by the splicing scheme which is written in Equation (56).

$$\begin{aligned}(\Xi\{_{1i}u_{1i}\}_{1i}\Xi\{_{2i}u_{2i}\}_{2i}\Xi\{_{3i}u_{3i}\}_{3i}\Xi\{_{4i}u_{4i}\}_{4i}\Xi,\cup,\$)\\ \sim(\$,\cup,\cup)\end{aligned} \quad (56)$$

In Equation (56), $\cup$ shows the blank symbol and $u_{ji}\in\{T,F\}, i,j=1,\cdots,4$ in which j is the number of output bit of S-boxes and i is number of S-boxes. Therefore, after applying the above rules, we can write the output of S-boxes as shown in Equation (57).

$$\Theta_{S_i} = \Xi\{_{1i}u_{1i}\}_{1i}\Xi\{_{2i}u_{2i}\}_{2i}\Xi\{_{3i}u_{3i}\}_{3i}\Xi\{_{4i}u_{4i}\}_{4i}\Xi \quad (57)$$

In Equation (57), $u_{ji} \in \{T, F\}, i, j = 1, \cdots, 4$ and Θ_{S_i} is the output of S-box number *i*.

4.4. Simulation of P-Box with AIR Systems

In this section, we build a Turing machine based on AIR systems that simulates P-boxes of **Figure 3**. For this reason, we write the input word and splicing schemes for P-box as shown in Equation (58).

$$\begin{aligned} \alpha_P &= \Xi\{_1x_1\}_1\Xi\{_2x_2\}_2\Xi\cdots\Xi\{_{16}x_{16}\}_{16} \\ &\quad \Xi p\$\{_1u_1\}_1\$\{_2u_2\}_2\$\cdots\$\{_{16}u_{16}\}_{16}\$p, \\ u_i &\in \{x_1, \cdots, x_{16}\}, 1 \le i \le 16 \end{aligned} \quad (58)$$

In the above equation which describes input to the P-boxes of "**Figure 3**", we further assume that all u_i, $1 \le i \le 16$ are distinctive and *p* is the blank symbol. Therefore, we can write the splicing schemes as shown in Equation (59).

$$(\Xi, \{_i, x_i) \sim (\$, \{_i, u_i), (x_i, \}_i, \Xi) \sim (u_i, \}_i, \$), 1 \le i \le 16 \quad (59)$$

After applying *trl* rules in a parallel fashion, such that they are guided by splicing rules of Equation (59), we can get to the word that is shown in Equation (60).

$$\begin{aligned} \beta_P &= \Xi\{_1u_1\}_1\Xi\{_2u_2\}_2\Xi\cdots\Xi\{_{16}u_{16}\}_{16}\Xi p \\ &\quad \$\{_1u_1\}_1\$\{_2u_2\}_2\$\cdots\$\{_{16}u_{16}\}_{16}\$p \\ u_i &\in \{x_1, \cdots x_{16}\}, 1 \le i \le 16 \\ u_i &\ne u_j, i \ne j \end{aligned} \quad (60)$$

Then, a deletion rule as shown in Equation (61) which is guided by splicing rule of Equation (62) produces the output of P-boxes. The output word of P-boxes is demonstrated in Equation (63).

$$\beta_P \Rightarrow_{del_P} \Theta_P \quad (61)$$

$$(\Xi\{_1u_1\}_1\Xi\{_2u_2\}_2\Xi\cdots\Xi\{_{16}u_{16}\}_{16}\Xi, \cup, \$) \sim (\$, \cup, \cup) \quad (62)$$

$$\Theta_P = \Xi\{_1u_1\}_1\Xi\{_2u_2\}_2\Xi\cdots\Xi\{_{16}u_{16}\}_{16}\Xi \quad (63)$$

In Equation (63), Θ_P is the output of P-box.

5. The Proposed Attack Plan

In order to demonstrate our algorithm, we use the attack graph of **Figure 6** in each node of which a fraction of the attack takes place. In the following, we explain the operations that are accomplished in each node of the graph of **Figure 6**.

In node Y_{in}, all possible keys are generated the process of which was explained in Section 4.1. In node Y_{11}, all possible keys that were generated in Y_{in} are bitwise XORed with the given plaintext using AIR system of Section 4.2 that is guided by splicing rules of Equations (8)-(28). Then, the generated strings are forwarded to S-boxes $S_1, \cdots, S_4$ of **Figure 3** and in node Y_{12}, S-boxes are applied in parallel to the output of node Y_{11} in accordance with splicing relations of Section (4.3). In Y_{13} the generated strings are permuted in a fashion dictated by the cipher instructions according to the splicing relations of Equations (59) and (62). The process of consecutive mixing with subkey and applying substitution and permutation operations go on in an iterated fashion in the next nodes such that in node Y_{n1}, Y_{n2} and Y_{n3} mixing with the n^{th} subkey, the n^{th} round substitution and the n^{th} round permutation take place, respectively, and in Y_{n+1} these gained strings are XORed with corresponding bits of the $n+1^{st}$ subkey and eventually, in node Y_{out} the generated bits are compared with the given ciphertext and we can find the key of the cipher if they are equal.

6. Performance Evaluation of the Proposed Scheme

Considering that the operations of each node take place concurrently for all strings that exist in that node, we can find the number of steps that takes to produce the key of the cipher. In node Y_{in} by applying different combinations of consecutive ciliate bio-operations to a single initial word which is applied in parallel and takes one step, we can produce all possible key of the cipher. In node Y_{11} the XOR operation is applied between the generated words of node and the plaintext and considering that each XOR operation has a depth of three, four steps are required to evaluate XOR of two bits that produces output of one. In node Y_{12} two steps are required to produce the resulting words. In node Y_{13} two steps should be accomplished to evaluate P-boxes. So, there will be the same steps for other nodes of Y_{i1}, Y_{i2}, Y_{i3} for $i = 2, 3, \cdots, n$. We further assume that there are n-rounds of substitutions and permutations. So, before the last node, Y_{out}, we need to do $1+4n+2n+2n+4$ operations. The operations of the last node can be carried out as follows: Those bits that have been generated in node

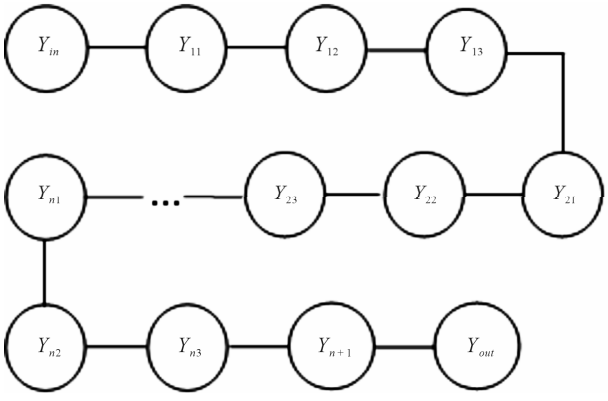

Figure 6. The attack graph for breaking the cipher.

Y_{out} are XORed with the corresponding given ciphertext bits which take 4-steps and then, the AND operation is applied to the results of XORs which takes one step. So, altogether $8n+10$ steps are mandatory to accomplish our attack in which n is the number of SP rounds.

Assuming that we use our proposed scheme to break 16-Round DES cipher (n = 16) which has a Feistel structure which can be assumed as a variant of SPN ciphers, using the analysis of Section 6, we need 138 steps of computation which gives it a superior performance over the proposed schemes of breaking DES using DNA computer [10] which requires 916 steps and membrane computing [11] which requires 278 steps of computations as well as breaking DES using network of evolutionary processors with parallel string rewriting rules (NEPPS) which requires 268 steps [12]. Furthermore, our proposed scheme, as opposed to [11] which needs exponential space, does not need exponential space and for a specific set of instructions of a given cipher, utilizes a constant number of splicing rules. The performance of our proposed scheme for cracking 16R-DES cipher in compareson with the previously proposed schemes has been depicted in **Figure 7**.

7. Simulation of the Proposed Attack

We conduct a computer simulation to test our proposed theoretical scheme to break a sample cipher based on the SPN of **Figure 3** and in this section, first, we introduce the parameters of the sample cipher which we aim to cryptanalyze and then we explain the steps of our simulations and finally, we present the results of simulations.

7.1. Parameters of the Considered Cryptosystem

In what follows, we introduce the cryptosystem under consideration for simulation in this paper. As can be seen in **Figure 3**, the input block size of our cryptosystem equals to 16 bits and a certain operation takes place for n rounds. Each round consists of substitution block, permutation and mixture of bits. This structure looks like the one that is used in DES and other modern block ciphers such as Rijndael [13]. The utilized blocks of the cipher can be defined as follows.

7.1.1. Substitution Block

In the cipher of **Figure 3**, we break the 16-bit block of input into four 4-bit blocks. Each sub-block constitutes an input to each S-box (a 4-to-4-bit S-box) that can be realized with a look-up table containing sixteen 4-bit values that can be defined with the integer numbers that are shown in the input bits of **Table 1**. The considered S-box is non-linear and the output bits of each S-box cannot be written as a linear combination of the inputs bits. Furthermore, it is assumed that all substitution boxes of $S_1,\cdots,S_4$ are equivalent. Our considered substitution box for the cipher is shown in **Table 1**.

7.1.2. Permutation Block

The considered permutation schedule for the cipher of **Figure 3** is shown in **Table 2**. Numbers of this table show the position of the bits in the block such that 1 shows the leftmost bit and 16 is the rightmost bit of the input block.

7.1.3. Mixing with Subkey

We define the operation of mixture with the key to be simply equal to applying the XOR gate between round key bits and the input block to the round.

In what follows, we design different sub-systems necessary for implementation of our proposed theoretical attack to the cipher.

Considering the above parameters in the cipher of

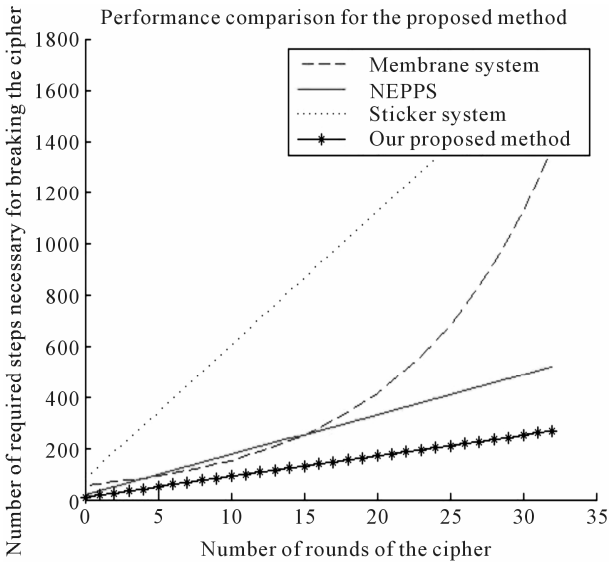

Figure 7. Graph of comparison for performance evaluation.

Table 1. Look-up table for the considered S-box.

Input	0	1	2	3	4	5	6	7
Output	E	4	D	1	2	F	B	8
Input	8	9	10	11	12	13	14	15
Output	3	A	6	C	5	9	0	7

Table 2. Bit permutation schedule of Figure 3.

Input	0	1	2	3	4	5	6	7
Output	1	5	9	13	2	6	10	14
Input	8	9	10	11	12	13	14	15
Output	3	7	11	15	4	8	12	16

Figure 3 and using splicing rules derived in Section 4, we simulate the proposed attack on the cipher.

We utilize Turing Machine Simulator software [14] to simulate our proposed attack on the cipher. For this reason, based on the splicing rules of Section 4, we write an initial program to simulate the main blocks of cryptanalysis according to the cipher parameters (such as generation of all keys, doing XOR, S-boxes and P-boxes). We write appropriate programs for each block to do computations on the words written on the tape of the Turing machine which simulate the AIR systems designed in Sections 4.1-4.4. Then we combine all the written programs based on the graph of **Figure 4**. In our simu- lation, we consider the following assumptions: we assume the number of rounds of the cipher equal to 1. Therefore we have one subkey string which is same as the main key and has a length of 16 bits. We further assume that the plaintext equals to "0000000000000000" and also the ciphertext is assumed to be "01011110011 00110". Now, based on the our designed AIR system, we write the applied algorithm for cryptanalysis as follows.

Initially, by applying different splicing rules on an initial sequence, we can produce all possible combinations of the key on the tape and then we put these strings on different segments of the tape with a certain distance from each other. Then, according to the splicing rules of Equations (8)-(28) we apply bitwise XOR operation between key bits of each key and the corresponding plaintext bits and their output is placed in another place on the tape. Then, by using splicing rules of Equations (35)-(50) and (56), we apply the substitution boxes of **Table 1** to the derived words in a parallel fashion and the resultant words are rewritten on the same words. Then, P-box of **Table 2** is applied in to the derived words in a parallel manner which utilizes the splicing rules of Equations (59) and (62) and we rewrite them on the previously derived words. Then, we apply the logical XOR operation between bits of the resultant words of this stage and the key bits which had been mixed with the resultant sequence of the previous steps (which have known location on the tape) and therefore, we can get to the enciphered message. Now, in accordance with the proposed operations in the last node of the graph of **Figure 4**, in this node, the achieved strings must be compared with the predefined ciphertext and if all the corresponding bits were equal, the used key is known as the cipher key.

We wrote appropriate programs to accomplish the abovementioned instructions and ultimately, we successfully derived the secret key equal to "0001001000 110100" which was predicted.

It is noteworthy that breaking the cipher of "**Figure 3**" with more rounds can be done in the same way. But for the sake of demonstration of our algorithm and also the problem of taking a long time for execution of the program we tested it on the cipher with 1 round of operations.

The codes shown in Appendix 1 are our specifically written programs to simulate Turing machines related to different building blocks necessary for the cryptanalysis procedure as demonstrated in Section 4 which are appropriate for execution on the Turing Machine Simulator software [14].

Note that all the written programs halt and generate appropriate output strings in a finite number of iterations The results of execution of codes of Appendix 1 for the considered cipher, defined in Section 7.1, are shown in **Tables 3-6**.

8. Conclusion

In this paper, we proposed a language-theoretic notion to break SPN class of block ciphers which is based on the gene assembly process that naturally occurs in ciliated protozoa during the procedure of converting scrambled MIC gene to MAC gene. Our scheme utilizes the AIR system which includes two modified versions of intramolecular ciliate bio-operations del_p and $trl_{p,q}$ which renders it the computational flavor of Turing machine. Assuming that we use our proposed scheme to break 16-Round DES cipher (n = 16) which has a Feistel structure which can be assumed as a variant of SPN ciphers, using the analysis of Section 6, we need 138 steps of computation which gives it a superior performance over the proposed schemes of breaking DES using DNA computer [13] which requires 916 steps and membrane

Table 3. Results of simulation of S-boxes.

Number of parallel operations	Initial state	Final state	Number of iterations
16	A0	HALT	80

Table 4. Results of simulation of P-boxes.

Number of parallel operations	Initial state	Final state	Number of iterations
1	A0	HALT	17

Table 5. Results of simulation of mixing with subkey.

Number of parallel operations	Initial state	Final state	Number of iterations
16	R2	HALT	12

Table 6. All operations of attack on the cipher.

Number of parallel operations	Initial state	Final state	Number of iterations
16-16-1-16	R2	HALT	121

computing [14] which requires 278 steps of computations as well as breaking DES using network of evolutionary processors with parallel string rewriting rules (NEPPS) which requires 268 steps [15]. Furthermore, our proposed scheme, as opposed to [14] which needs exponential space, does not need exponential space and for a specific set of instructions of a given cipher, utilizes a constant number of splicing rules. Finding nature-inspired computational models like L-systems or modified versions of P-systems or our utilized model, seem to be promising in developing efficient computational models which simulate universal Turing machines and for future works, other computational problems can be suggested to be solved using these models.

REFERENCES

[1] L. M. Adleman, "Molecular Computation of Solutions to Combinatorial Problems," *Science*, Vol. 266, No. 5187, 1994, pp. 1021-1024. doi:10.1126/science.7973651

[2] L. Kari and L. F. Landweber, "Computational Power of Gene Rearrangement," In: V. E. Winfree and D. K. Gifford, Eds., *Proceedings of DNA Bases Computers*, American Mathematical Society, 1999, pp. 207-216.

[3] L. F. Landweber and L. Kari, "The Evolution of Cellular Computing: Nature's Solution to a Computational Problem," *Proceedings of the 4th DIMACS Meeting on DNA-Based Computers*, Philadelphia, 15-19 June 1998, pp. 3-15.

[4] R. Lipton, "Using DNA to Solve NP-Complete Problems," *Science*, Vol. 268, 1995, pp. 542-545. doi:10.1126/science.7725098

[5] L. M. Adleman, P. W. K. Rothemund, S. Roweis and E. Winfree, "On Applying Molecular Computation to the Data Encryption Standard," *Proceedings of 2nd Annual Meeting on DNA Based Computers, DIMACS Workshop*, Princeton University, Princeton, 10-12 June 1996, pp. 28-48.

[6] D. R. Stinson, "Cryptography: Theory and Practice," CRC Press, Boca Raton, 2002.

[7] T. Head, "Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors," *Bulletin of Mathematical Biology*, Vol. 49, No. 6, 1987, pp. 737-759.

[8] T.-O. Ishdorj and I. Petre, "Gene Assembly Models and Boolean Circuits," *International Journal of Foundations of Computer Science*, Vol. 19, No. 5, 2008, pp. 1133-1145. doi:10.1142/S0129054108006182

[9] T.-O. Ishdorj, I. Petre and V. Rogojin, "Computational Power of Intramolecular Gene Assembly," *International Journal of Foundations of Computer Science*, Vol. 18, No. 5, 2007, pp. 1123-1136. doi:10.1142/S0129054107005169

[10] W. Stallings, "Cryptography and Network Security Principles and Practices," 4th Edition, Prentice Hall, Upper Saddle River, 2005, pp. 134-173.

[11] http://www3.wittenberg.edu/bshelburne/Turing.htm

[12] J. Castellanos, C. Martin-Vide, V. Mitrana and J. Sempere, "Networks of Evolutionary Processors," *Acta Informatica*, Vol. 39, No. 6-7, 2003, pp. 517-529.

[13] D. Boneh, C. Dunworth and R. Lipton, "Breaking DES Using a Molecular Computer," In: R. J. Lipton and E. B. Baum, Eds., *DNA Based Computers*: *Proceedings of a DIMACS Workshop*, Princeton, 10-12 June 1996, pp. 37-66.

[14] S. N. Krishna and R. Rama, "Breaking DES Using P System," *Theoretical Computer Science*, Vol. 299, No. 1-3, 2003, pp. 495-508. doi:10.1016/S0304-3975(02)00531-5

[15] A. Choudhary and K. Krithivasan, "Breaking DES Using Networks of Evolutionary Processors with Parallel String Rewriting Rules," *International Journal of Computer Mathematics*, Vol. 86, No. 4, 2009, pp. 567-576. doi:10.1080/00207160701351168

Appendix 1: Program Codes for Simulation of the Proposed AIR Systems

In this section, the code lines written for simulation of the proposed AIR systems for each building block of the attack are shown. Note that the programs have been written to be suitable for the Turing simulator software that was used for our simulations. R and L in the written programs demonstrate directions of right and left, respectively, of head of the considered Turing machines on the tape.

Code Lines for Simulation of S-Boxes Using AIR Systems

(A0, 0, 1, B0, R) (B0, 0, 1, C0, R)
(C0, 0, 1, D0, R) (D0, 0, 0, E0, R)
(E0, , , E0, R) (E0, 0, 0, A1, R)
(A1, 0, 1, B1, R) (B1, 0, 0, C1, R)
(C1, 1, 0, D1, R) (D1, , , D1, R)
(D1, 0, 1, A2, R) (A2, 0, 1, B2, R)
(B2, 1, 0, C2, R) (C2, 0, 1, D2, R)
(D2, , , D2, R) (D2, 0, 0, A3, R)
(A3, 0, 0, B3, R) (B3, 1, 0, C3, R)
(C3, 1, 1, D3, R) (D3, , , D3, R)
(D3, 0, 0, A4, R) (A4, 1, 0, B4, R)
(B4, 0, 1, C4, R) (C4, 0, 0, D4, R)
(D4, , , D4, R) (D4, 0, 1, A5, R)
(A5, 1, 1, B5, R) (B5, 0, 1, C5, R)
(C5, 1, 1, D5, R) (D5, , , D5, R)
(D5, 0, 1, A6, R) (A6, 1, 0, B6, R)
(B6, 1, 1, C6, R) (C6, 0, 1, D6, R)
(D6, , , D6, R) (D6, 0, 1, A7, R)
(A7, 1, 0, B7, R) (B7, 1, 0, C7, R)
(C7, 1, 0, D7, R) (D7, , , D7, R)
(D7, 1, 0, A8, R) (A8, 0, 0, B8, R)
(B8, 0, 1, C8, R) (C8, 0, 1, D8, R)
(D8, , , D8, R) (D8, 1, 1, A9, R)
(A9, 0, 0, B9, R) (B9, 0, 1, C9, R)
(C9, 1, 0, D9, R) (D9, , , D9, R)
(D9, 1, 0, F0, R) (F0, 0, 1, G0, R)
(G0, 1, 1, H0, R) (H0, 0, 0, I0, R)
(I0, , , I0, R) (I0, 1, 1, F1, R)
(F1, 0, 1, G1, R) (G1, 1, 0, H1, R)
(H1, 1, 0, I1, R) (I1, , , I1, R)
(I1, 1, 0, F2, R) (F2, 1, 1, G2, R)
(G2, 0, 0, H2, R) (H2, 0, 1, I2, R)
(I2, , , I2, R) (I2, 1, 1, F3, R)
(F3, 1, 0, G3, R) (G3, 0, 0, H3, R)
(H3, 1, 1, I3, R) (I3, , , I3, R)
(I3, 1, 0, F4, R) (F4, 1, 0, G4, R)
(G4, 1, 0, H4, R) (H4, 0, 0, I4, R)
(I4, , , I4, R) (I4, 1, 0, F5, R)
(F5, 1, 1, G5, R) (G5, 1, 1, H5, R)
(H5, 1, 1, I5, R) (I5, , , I5, R)
(I5, , , Z, L)

Code Lines for Simulation of P-Boxes Using AIR Systems

(A0, a, a, B0, R) (B0, b, e, C0, R)
(C0, c, i, D0, R) (D0, d, m, E0, R)
(E0, e, b, A1, R) (A1, f, f, B1, R)
(B1, g, j, C1, R) (C1, h, n, D1, R)
(D1, i, c, A2, R) (A2, j, g, B2, R)
(B2, k, k, C2, R) (C2, l, o, D2, R)
(D2, m, d, A3, R) (A3, n, h, B3, R)
(B3, o, l, C3, R) (C3, p, p, D3, R)
(D3, , , D3, R) (D3, , , Z, L)

Code Lines for Simulation of XOR Using AIR Systems

(R2, 1, 1, R2, R) (R2, +, +, R2, R)
(R2, 0, 0, R2, R) (R2, 1, 1, R2, R)
(R2, 0, 0, R2, R) (R2, 1, 1, R2, R)
(R2, 0, 0, R2, R) (R2, , , L0, L)
(L0, 1, 0, L1, L) (L0, 0, 1, L0, L)
(L0, +, , R4, R) (R4, 1, , R4, R)
(R4, , , Z, R) (Z, , , Z, L)
(L1, +, +, L3, L) (L3, 1, 0, L3, L)
(L3, , 1, R2, R) (L3, 0, 1, R2, R).

