Open Systems & Information Dynamics
Vol. 18, No. 1 (2011) 1–19
DOI: xxx
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

From Cryptography to Biology and Vice Versa

Massimo Regoli

University of Rome "Tor Vergata" — Centro Interdipartimentale "Vito Volterra"

(Received: September 14, 2010)

Abstract. We introduce a new symmetric cryptographic algorithm, based on the biological principle of *redundancy.* This algorithm has very good security and statistical properties. In addition, it suggests a nontrivial information-theoretical interpretation of the redundancy mechanism in DNA sequences which does not seem to be present in biological literature: according to this interpretation the introns do not directly code information, but play an essential role in the decoding procedure.

1. Introduction

The RNA-Crypto System (shortly RCS) is a symmetric key algorithm to cipher data. This algorithm, as shown below, has the peculiarity to expand the message to be encrypted, hiding the ciphered message itself within a set of *garbage* and *control information*.

The idea for this new algorithm has originated from the observation of nature, in particular from the observation of RNA behaviour and some of its properties. The RNA sequences include some sections called *introns* (the name is derived from the term "intragenic regions"). These are non-coding sections of the precursor mRNA (*pre-m*RNA) or other RNAs, that are removed (spliced out of the RNA) before the mature RNA is formed. Once the introns have been spliced out of a *pre-m*RNA, the resulting mRNA sequence is ready to be translated into a protein. The corresponding parts of a gene are known as introns as well. The nature and the role of introns in the *pre-m*RNA is not clear and is under intensive research by biologists. The algorithm described below introduces a mathematical analogue to intron sequences in the RNA-Crypto System output as a device to add information which is only apparently chaotic and non-coding, but in fact plays an essential role in decoding the message.

A general conclusion that can be drawn from the present paper is that, while the use of cryptanalytic tools in the attempt to decode genetic sequences has a long history, the converse direction, i.e. the exploitation of biological ideas to construct cryptographic algorithms, may produce new fruitful tools in cryptography as well as nontrivial suggestions in the direction of an information-theoretical interpretation of biological phenomena, such as the *redundancy* in DNA sequences.

This direction will be the object of future investigations.

In the following we describe the mathematical structure of the new cryptographic algorithm.

1.1. Interpretation

The main goal of the present paper is to use the algorithm described below to code messages. But the basic idea of the algorithm may also suggest the possibility to consider redundancy in the *pre-m*RNA sequences as a protection mechanism used by nature against possible decoding attacks from the outside or, from another point of view, as a mechanism giving introns an important role for coding the resulting *mRNA*, for example to achieve future functionalities or realize old ones.

1.2. Properties: disadvantages and peculiarities

As mentioned above, this system has the peculiarity to expand the original text in our implementation and this expansion is huge. This feature, which obviously makes the system inapplicable in all those cases in which the plaintext to be encoded is large, it is on the contrary an advantage in all those cases in which the size of the plaintext is small, because chaos introduced by non-coding components (junk), creates a great obstacle to eavesdropping attempts. As an example, to encode passwords, pin-code, small sentences that are of the order of tens/hundreds of bytes, this algorithm proves to be very powerful.

2. Ingredients of the algorithm

2.1. The spaces

For any set X we denote by $|X|$ its cardinality and by $X_0^{\mathbb{N}}$ the family of all finite ordered sequences of elements of X, i.e.

$$X_0^{\mathbb{N}} := \bigcup_{n=0}^{\infty} X^n ,$$

with the convention

$$X^0 := \{\emptyset\} .$$

"$\emptyset$" denotes here the null sequence and it is easy to be distinguished by context from its conventional use as an empty set symbol. The *length function*

$\ell^X : X_0^{\mathbb{N}} \to \mathbb{N}$ is defined by

$$\ell^X(x) \equiv \ell_x^X := n \iff x \in X^n \,, \qquad x \in X_0^{\mathbb{N}} \,,$$

and its value in $x \in X_0^{\mathbb{N}}$ is called the *length of* x and often simply denoted ℓ_x.

DEFINITION 1 For $x \in X_0^{\mathbb{N}}$, $m \in \mathbb{N}$ and $i \in \{1, \ldots, \ell_x\}$ we define the subsequence:

$$\tilde{x}_{i,m} := \Big\{ x_i, x_{i+_{\ell_x} 1}, \ldots, x_{i+_{\ell_x}(m-1)} \Big\} \,, \tag{1}$$

where, here and in the following, for $\ell \in \mathbb{N}$, the symbol $+_\ell$ denotes addition modulo ℓ. By definition $\tilde{x}_{i,0}$ is the empty sequence. A subset $\mathcal{S} \subseteq X_0^{\mathbb{N}}$ will be called hereditary if, for any sequence $x \in \mathcal{S}$, any $m \in \mathbb{N}$ and any $i \in \{1, \ldots, \ell_x\}$, the sequence $\tilde{x}_{i,m}$, defined by (1), also belongs to $\mathcal{S}$.

Remark 1 Most functions used to manipulate sequences will be of *l*ocal type, i.e. not depending on the whole sequence but on some sub-block of it, in the sense of Definition 1. Therefore, it is notationally convenient to suppose that all sequence spaces considered (messages, keys, coded messages, ...) are hereditary. In the following we will make this assumption.

The algorithm makes use of several sets playing different roles and a priori different, namely:

— the key alphabet K,
— the message alphabet M,
— the *exons* alphabet B_{ex},
— the *introns* alphabet B_{in},
— the *control* space $S = S_1 \cup S_2$, where $S_1, S_2 \subset S$ are non-empty subsets satisfying

$$S_1 \cap S_2 = \emptyset \,, \qquad |S_j| < \infty \,, \quad j = 1, 2,$$

and, in most concrete cases, we shall choose:

$$K = M = \{0, 1\} \,. \tag{2}$$

Sets of finite ordered sequences with values in one of these spaces will play an important role in the algorithm:

— the space of *Secret Shared Keys* (SSK) $\mathcal{K} \subseteq K_0^{\mathbb{N}}$, which are finite ordered sequences of symbols in K;
— the space of *messages* $\mathcal{M} \subseteq M_0^{\mathbb{N}}$, i.e. finite ordered sequences of symbols in M;
— the *exons* space $\mathcal{B}_{\text{ex}} \subseteq (B_{\text{ex}})_0^{\mathbb{N}}$, which are finite ordered sequences of symbols in B_{ex};

— the *introns* space $\mathcal{B}_{\text{in}} \subseteq (B_{\text{in}})_0^{\mathbb{N}}$, i.e. finite ordered sequences of symbols in B_{in};

— The union of the exons and introns spaces defines the output alphabet $B = B_{\text{ex}} \cup B_{\text{in}}$, the union being disjoint;

— the space of *Coded Messages* (or *Output sequences*)

$$\mathcal{C} \subseteq (B_{\text{ex}} \cup B_{\text{in}})_0^{\mathbb{N}} ,$$

i.e. finite ordered sequences of symbols in the output alphabet.

With the choice (2) the spaces $\mathcal{K}$ and $\mathcal{M}$ become simply finite ordered sequences of standard binary digits and the coded messages will be finite ordered sequences of blocks, representing exons or introns.

2.2. Functions

The algorithm uses several classes of functions.

2.2.1. Coding functions

Their role is twofold:

(i) to transform a portion of an open (clear) message into a portion of the coded message (exons);

(ii) to insert some *apparently* redundant information in the coded message (introns).

First we start with the definition of a family of *coding functions* parametrized by the control space, i.e. $\forall\, s \in S$ a function f_s is given,

$$f_s : \mathcal{M} \times \mathcal{K} \to \mathcal{B}_{\text{ex}} \cup \mathcal{B}_{\text{in}} ,$$

the union again being disjoint, with the following properties:

(i) $\forall\, s \in S$ and $\forall\, (\sigma, \kappa) \in \mathcal{M} \times \mathcal{K}$

$$f_s\,(\sigma, \kappa) \;=\; \Sigma \in \begin{cases} \mathcal{B}_{\text{ex}} & \text{if } s \in S_1 \\ \mathcal{B}_{\text{in}} & \text{if } s \in S_2 \,. \end{cases} \tag{3}$$

(ii) $\forall\, s \in S_1$ and $\forall\, \kappa \in \mathcal{K}$ the function $f_{s,\kappa} : \mathcal{M} \to \mathcal{B}_{\text{ex}}$ such that

$$f_{s,\kappa}(\sigma) \;=\; f_s\,(\sigma, \kappa)$$

is invertible. Its inverse will be denoted $\bar{f}_{s,\kappa}$ or, if no confusion is possible, simply by $\bar{f}$.

(iii) For $s \in S_2$ and $\kappa \in \mathcal{K}$, we set

$$\bar{f}_{s,\kappa}(\Sigma) \;=\; \bar{f}(\Sigma) \;=\; \emptyset \,.$$

Using this notation we can simply write: $\bar{f} : (\mathcal{B}_{\text{ex}} \cup \mathcal{B}_{\text{in}}) \times \mathcal{K} \to \mathcal{M}$ satisfies

$$\bar{f}\,(\Sigma, \kappa) \;=\; \begin{cases} \sigma & \text{if } \Sigma \in \mathcal{B}_{\text{ex}} \\ \emptyset & \text{if } \Sigma \in \mathcal{B}_{\text{in}}\,. \end{cases} \tag{4}$$

2.2.2. Operational functions

We have already said that all functions considered are local. The operational functions specify which block of the key, or of other control sequences defined in the system, have to be used at each step of the algorithm. These functions are parametrized by the output (i.e. coded) sequences:

DEFINITION 2 *Let $\bar{g}_\kappa : \mathcal{C} \to \mathcal{K}$ be a family of functions, where $\mathcal{K}$ and $\mathcal{C}$ are defined in Sect. 2.1. These functions transform blocks of a key to be used during the encoding/decoding process.*

For convenience and simplicity we will use in the following the notation:

$$\bar{g}(\kappa, \Sigma) = \bar{g}_\kappa(\Sigma)\,. \tag{5}$$

Intuitively this function indicates which part of the SSK has to be used in the next step of the algorithm as a function of the output in the previous step or as a function of the coded message (depending on whether we are encoding or decoding).

Another interpretation is that the function $\bar{g}$ produces, at each step, the SSK that has to be used in the next step (for this reason some *chaos* properties are desirable).

Another operational function used in the algorithm is the characteristic function $\chi_{S_1} : S \to \{0, 1\}$, i.e. $\chi_{S_1}(s) = 1 \Leftrightarrow s \in S_1$.

2.3. Global variables and further notations

In cryptography a Secret Shared Key (SSK) is a sequence of symbols (usually binary digits) which has been previously shared between the two parties. Its role is to allow coding of information in such a way that only those who know this key are able to recover them. The present algorithm is based on such a SSK (i.e. it belongs to the class of symmetric algorithms) which in the following will be denoted by $\kappa \in \mathcal{K}$ and whose length by N_K. We shall denote by

$$\alpha = (\alpha_i)\,, \qquad \alpha_i \in S\,,$$

a sequence of symbols in the control space whose length will depend on the message to be coded. We require this sequence to possess "good random properties", i.e. to mimic the properties of generic sequences obtained by picking at random, with uniform distribution, symbols from the space S.

In practice such sequences are produced using pseudo-random generators (PRNG) and their "randomness" can be measured by popular packages of statistical tests (see Sect. 6.). Any kind of information will be called a *message* (also *clear text*) and denoted $\sigma \in \mathcal{M}$. Its length will be denoted by N_M. The role of cryptography is to protect this information by preventing any unauthorized access to it. This goal is achieved by transforming the message into a new sequence of symbols, the *coded (or crypted) message*, using a transformation which is invertible, but whose inverse depends on the SSK so that only those who possess it can apply this inverse and recover the message.

In the following the coded message will be denoted by $\Sigma \in \mathcal{C}$ and its length will depend on the system and on the control variable α.

Let us emphasize:

(i) that the algorithm can produce different coded messages, corresponding to the same message σ and the same SSK, simply by using different control sequences (α);

(ii) that it is not necessary, during the decoding phase, to know the sequence (α) used in the coding phase. This is because the information about (α) can be deduced from the pair (Σ, κ) (i.e. coded message and SSK: see formula (5) for a precise formulation of this fact);

(iii) that possible role of *introns* is to carry important information for the decoding phase, for example the function $\bar{g}(\kappa, \Sigma)$ can use the information in $\Sigma \in \mathcal{B}$ to produce a change in the SSK.

Finally, during the coding phase, there is an expansion of the original messages into the coded message depending on several factors, such as the dimensions of the intron sequences and the number of elements of S_2 in the control sequence (α).

2.4. Biological parallelism

In biological interpretation, the combined action of the SSK and of the control sequence (α) mimic the mechanism of *splicing* through which the *pre-m*RNA is modified by removing certain stretches of non-coding sequences (*introns*), while the coded message is the *pre-m*RNA itself and the clear message is the final *mRNA*.

3. The algorithm

3.1. Coding

The coding operation requires:

— a pre-shared key (SSK) $\kappa \in \mathcal{K}$, of length $l_\kappa = N_K$;

— a message $\sigma \in \mathcal{M}$ of length $l_\sigma = N_M$;
— a pseudo-random S-valued sequence $\alpha = (\alpha_i)$, $\alpha_i \in S$;
— two fixed pre-shared numbers $n, m \in \mathbb{N}$ such that

$$n \,|\, l_\sigma \qquad (n \text{ divides } l_\sigma = N_M)\,.$$

Define the i-th bit of the output

$$\Sigma_i \;=\; f_{\alpha_i}(\tilde{\sigma}_{l_i}, \tilde{\kappa}_{j_i}) \;=\; \begin{cases} \varepsilon_i \in \mathcal{B}_{\text{ex}} & \text{if } \alpha_i \in S_1 \\ \iota_i \in \mathcal{B}_{\text{in}} & \text{if } \alpha_i \in S_2\,, \end{cases} \tag{6}$$

where $\tilde{\sigma}_{l_i}$ and $\tilde{\kappa}_{j_i}$ are as in Definition 1, ε_i is the i-th bit of the message, ι_i is an arbitrary intron, and l_{i+1}, i.e. the index of the next bit of the message to be processed, is given by

$$\begin{aligned} l_{i+1} &= l_i + \chi_{S_1}(\alpha_i) n\,, \\ \tilde{\kappa}_{j_{i+1}} &= \bar{g}(\kappa, \Sigma_i)\,. \end{aligned}$$

DEFINITION 3 The sequence $\Sigma = (\Sigma_1, \ldots, \Sigma_N) \in \mathcal{C}$ is the coded message.

3.2. Decoding

The crucial point for the definition of the function $\bar{f}$, which is the basis of the decoding procedure, is that, given the output sequence Σ (i.e. the entire coded message), which is a sequence of 0's and 1's, one is able to distinguish the single block Σ_i. In particular one can recognize if $\Sigma_i \in \mathcal{B}_{\text{ex}}$ or $\Sigma_i \in \mathcal{B}_{\text{in}}$.

The decoding operation requires:

— an SSK $\kappa \in \mathcal{K}$, of length $l_\kappa = N_K$;
— the same fixed pre shared numbers $n, m \in \mathbb{N}$ as in the coding operation;
— a coded message $\Sigma = (\Sigma_1, \ldots, \Sigma_N) \in \mathcal{C}$ of length l_Σ.

Using the above objects and the knowledge of the single blocks $(\Sigma_1, \ldots, \Sigma_N)$ with $\Sigma_i \in \mathcal{B}_{\text{ex}} \cup \mathcal{B}_{\text{in}}$, the decoding phase computes the i-th block of the message $\tilde{\sigma}_i$ and the j_{i+1}-th block of the SSK $\tilde{\kappa}_{j_{i+1}}$, to be used in the next step, by distinguishing two cases:

Case A: if $\Sigma_i \in \mathcal{B}_{\text{ex}}$, then the i-th block of the message is

$$\tilde{\sigma}_i \;=\; \bar{f}(\Sigma_i, \tilde{\kappa}_{j_i}) \tag{7}$$

and the j_{i+1}-th block of the SSK $\tilde{\kappa}_{j_{i+1}}$ is

$$\tilde{\kappa}_{j_{i+1}} \;=\; \bar{g}(\kappa, \Sigma_i)\,.$$

Case B: if $\Sigma_i \in \mathcal{B}_{\text{in}}$, then the i-th block of the message is

$$\bar{f}(\Sigma_i, \tilde{\kappa}_{j_i}) \;=\; \emptyset$$

		$\tilde{\kappa}_j$							
		000	001	010	011	100	101	110	111
$(\Sigma_{i,1}, \Sigma_{i,2}, \Sigma_{i,3})$	000	a	b	c	d	a	b	c	d
	001	b	c	d	a	b	c	d	a
	010	c	d	a	b	c	d	a	b
	011	d	a	b	c	d	a	b	c
	100	a	b	c	d	a	b	c	d
	101	b	c	d	a	b	c	d	a
	110	c	d	a	b	c	d	a	b
	111	d	a	b	c	d	a	b	c

Table 1: The seek table $T_{r,c}$

and the j_{i+1}-th block of the SSK $\tilde{\kappa}_{j_{i+1}}$ is

$$\tilde{\kappa}_{j_{i+1}} \;=\; \bar{g}(\kappa, \Sigma_i)\,.$$

Remark 2 As mentioned at the end of Sect. 2.2., thanks to the definition of $\bar{g}$, in the decoding phase it is not necessary to know the random sequence (α).

4. A concrete implementation

4.1. The environment

In this section, we introduce a concrete realization of the above abstract scheme. We make the following choices:

$$K = M = B_{\text{ex}} = B_{\text{in}} = \{0,1\}\,,$$

$$|\kappa| = N_K\,, \qquad |\sigma| = N_M\,, \qquad N_K, N_M \in \mathbb{N} \text{ arbitrary},$$

$$\begin{aligned} N_{B_{\text{ex}}} &= 3, & N_{B_{\text{in}}} &= 2, \\ n &= 1, & m &= 3, \\ S_1 &= \{a,b\}, & S_2 &= \{c,d\}, \end{aligned}$$

where a, b, c, d are arbitrary symbols,

$$\mathcal{B} = \mathcal{B}_{\text{ex}} \cup \mathcal{B}_{\text{in}} = \{0,1\}^3 \cup \left(\{0,1\}^3 \times \{0,1\}^2\right) \text{ (disjoint union)}\,.$$

Finally we fix the table $T : \{0,1\}^3 \times \{0,1\}^3 \to S$, that will be used to define the functions f_α. Table 1 may be public.

4.2. Functions and implementation

A produces the output (i.e. coded) message in blocks: having produced the first $i-1$ block (i.e. having coded the first $i-1$ bits of the message), the i-th one is produced using:

— the i-th bit α_i, of the pseudo-random sequence α,
— the block $\tilde{\kappa}_{j_i,3}$ of the SSK, produced at the i-th step according to the following rules:

Case (1) If $\alpha_i \in S_1$ then the i-th output block is an exon given by

$$\Sigma_i \;=\; (\Sigma_{i,1}, \Sigma_{i,2}, \Sigma_{i,3}) \;=\; (r_1, r_2, r_3) \;=\; r \in \{0,1\}^3,$$

where $r = (r_1, r_2, r_3)$ is a binary number $000 \le r \le 111$ defined as follows.

Case (1.1) If $\tilde{\sigma}_{i,1} = \sigma_i = 0$ then $r \equiv (r_1, r_2, r_3)$ (see the notation (1)) is any solution of the equation

$$T_{r,\tilde{\kappa}_{j_i,3}} \;=\; a$$

randomly chosen from Table 1.

Case (1.2) If $\tilde{\sigma}_i = \sigma_i = 1$ then $r \equiv (r_1, r_2, r_3)$ is any solution of the equation

$$T_{r,\tilde{\kappa}_{j_i,3}} \;=\; b\,.$$

Case (2) if $\alpha_i \in S_2 = \{c, d\}$, then the i-th output block is an intron given by

$$\Sigma_i = ((\Sigma_{i,1}, \Sigma_{i,2}, \Sigma_{i,3}), (\Sigma_{i,4}, \Sigma_{i,5})) = ((r_1, r_2, r_3), (\Sigma_{i,4}, \Sigma_{i,5})) \in \{0,1\}^3 \times \{0,1\}^2,$$

where $r \equiv (r_1, r_2, r_3)$ is any solution of the equation

$$T_{r,\tilde{\kappa}_{j_i,3}} \;=\; \alpha_i$$

and the other 2 components of the vector (i.e. $\Sigma_{i,4}, \Sigma_{i,5}$) are chosen randomly in the set $\{00, 01, 10, 11\}$.

The function $\bar{f}$ depends on the the first 3 bits of the i-th block Σ_i of the output, denoted $r = (r_1, r_2, r_3)$, and on the block $\tilde{\kappa}_{j_i,3}$ of the SSK, as follows. If $T_{r,\tilde{\kappa}_{j_i,3}} = a$ then

$$\bar{f}(r, \tilde{\kappa}_{j_i,3}) \;=\; 0 \;=\; \sigma_i \tag{8}$$

else if $T_{r,\tilde{\kappa}_{j_i,3}} = b$ then

$$\bar{f}(r, \tilde{\kappa}_{j_i,3}) \;=\; 1 \;=\; \sigma_i \tag{9}$$

else obviously

$$T_{r,\tilde{\kappa}_{j_i,3}} \notin \{a, b\}$$

then

$$\bar{f}(r, \tilde{\kappa}_{j_i,3}) \;=\; \emptyset\,. \tag{10}$$

		$\tilde{\kappa}_j$			
		00	01	10	11
$(\Sigma_{i,1}, \Sigma_{i,2})$	00	a	b	c	d
	01	b	c	d	a
	10	c	d	a	b
	11	d	a	b	c

Table 2: An alternative seek table $T_{r,c}$

Notice that such a function $\bar{f}$ satisfies condition (4). The function $\bar{g}$ depends on the same variables $r = (r_1, r_2, r_3)$ and $\tilde{\kappa}_{j_i,3}$ as the function $\bar{f}$ and is defined as follows. If $T_{r,\tilde{\kappa}_{j_i,3}} \in \{a, b\}$ then

$$\bar{g}(r, \tilde{\kappa}_{j_i,3}) = \tilde{\kappa}_{j_i +_{\ell_\kappa} (\Sigma_{i,4}\Sigma_{i,5})_{10}} , \tag{11}$$

where the subscript 10 means the decimal representation of the binary number in parenthesis and $j_i +_{\ell_\kappa}$ denotes addition of decimal numbers modulo ℓ_κ. If $T_{r,\tilde{\kappa}_{j_i,3}} \in \{c, d\}$ then, with the same notations:

$$\bar{g}(r, \tilde{\kappa}_{j_i,3}) \;=\; \tilde{\kappa}_{j_i -_{\ell_\kappa} (\Sigma_{i,4}\Sigma_{i,5})_{10}} .$$

4.3. Observation

In this simple implementation of the RNA algorithm, redundancy is given by the following considerations:

— for each processed bit in the clear message 3 bits will be inserted in the coded one;
— each *intron* will insert in the coded message 5 bits;
— we use an uniform distribution for the generation of the sequence (α_i);
— then we can suppose that the length of the sequence (α) is $|(\alpha)| = |(\alpha_i|\alpha_i \in S_1)| + |(\alpha_i|\alpha_i \in S_2)|$;
— clearly in this implementation one must have $|(\alpha_i|\alpha_i \in S_1)| = N_M$.

Then, if the original message has the length of N_M bits, the length of coded one is $l_c \approx 3N_M + 5N_M = 8N_M$.

If we want to reduce the expansion of the message we can replace the redundant Table (1), in which the equation $T_{r,\tilde{\kappa}} = s$, in the unknown $r \in \{0,1\}^3$, admits two solutions for any $\tilde{\kappa} \in \{0,1\}^3$ and $s \in \{a,b,c,d\}$, with Table 2 (in this version must be $N_A = 2$ and $m = 2$).

In this case the message explosion will be $l_c \approx 2N_M + 4N_M = 6N_M$.

Another solution in order to diminish the explosion of the coded message is that to use an asymmetric probability function for the generation of (α_i). See for example Table 3.

α	p
a	3/8
b	3/8
c	1/8
d	1/8

Table 3: **Caption required**

Using these last two optimizations the approximate length of the coded message will be reduced to $l_c \approx 3N_M$.

4.4. A more realistic biological implementation

As a further example we discuss an implementation of our model which mimics in a more realistic way the biological phenomenology. The ingredients for the new implementation are the following:

— the key alphabet $K := \mathbb{N}$;

— the message, intron and exon alphabets coincide and are equal to: $M = A = \mathcal{B}_{\text{ex}} = \mathcal{B}_{\text{in}} := \{a, b, c, d\}$;

— the *introns* space: $B_{\text{in}} = \{A^3 \backslash (a, a, a)\}$ (thus $N_A = 3$);

— the *Exons* space: $B_{\text{ex}} = \left(\{a, a, a\} \times \left(\bigcup_i A^i\right)\right)$, so that

$$\mathcal{B} = \{A^{N_A} \backslash (a, a, a)\} \cup \left(\{a, a, a\} \times \left(\bigcup_i A^i\right)\right)$$

(since in nature the length of intron sequences is finite, we have *abused* of the notation $\bigcup_i A^i$ in the above formulas);

— the *control* space $S = S_1 \cup S_2$ with $S_1 = \{c\}$ and $S_2 = \{nc\}$;

— we fix $m = 1$;

— in the notation (1) the function f will be:

$$f_\alpha(\tilde{\sigma}_{i,1}, \tilde{\kappa}_{j_i,1}) = \begin{cases} \tilde{\sigma}_i & \text{if} \quad \alpha \in S_1 \\ (a, a, a, \Sigma_i) \quad \text{with} \quad \Sigma_i \in A^{\tilde{\kappa}_{j_i,1}} & \text{if} \quad \alpha \in S_2\,, \end{cases} \tag{12}$$

in this case the element $(a, a, a) \in A^3$ is the codon to localize the *beginning of the introns* in the sequence (it will never appear in the clear code) and the function $\bar{f}$ will be:

$$\sigma_{i^*} = \bar{f}(\Sigma_i, \tilde{\kappa}_{j_i}) = \begin{cases} (\Sigma_{i,1}, \Sigma_{i,2}, \Sigma_{i,3}) & \text{if} \quad (\Sigma_i) \neq (a, a, a) \\ \emptyset & \text{otherwise}\,. \end{cases} \tag{13}$$

In both cases one has $\tilde{\kappa}_{j_{i+1},1} = \tilde{\kappa}_{j_i+1,1}$.

5. The role of the coding functions

In Section 2.2. we introduce the definition of coding functions. These functions play an important role in the complexity and in the lengths of the cipher text.

It is also very important to understand that the number of functions involved in the process of encryption can be huge. In fact, besides the canonical functions introduced in Sect. 4.1. we can add functions that have the role of:

— Changing the public/secret keys (for example in the cipher text we can insert a sequence of bits of arbitrary length that will replace the public (private) key to decode the rest of the message).

— Inserting long sequences of random bits (as before but the inserted bits are ignored (real redundancy)).

— Moving forward or backward the *key pointer* position (the presence of a disruptive effect on the sequencing access to the secret key adds more noise against the attacks).

— Resetting global parameters like message pointer position, key pointer position, secret and public data (see before).

It is easy to see that the number of *coding functions* that can be inserted in the encoding mechanism is great and may become part of the shared secret information.

In the same way it is obvious that some *coding functions* can greatly increase the size of the cipher text as it is also obvious that the inclusion of random bits within the text could create an increase in the randomness of the cipher text.

6. Statistical analysis

6.1. Cryptanalysis

While cryptography studies techniques for concealing a message, cryptanalysis (from the greek kryptos, “hidden” and analyein, “break”) is the study of methods for recovering the encrypted information without having direct access to secret informations used by the algorithm to achieve this goal (secret keys or SSK). Typically the result of cryptanalysis is the recovery of either the secret key of one of the two interlocutors or directly of the SSK.

Cryptanalysis is thus the “counterpart” of cryptography and together they form *cryptology*.

The term *cryptanalysis* is usually referred to the *logical* aspects, i.e. to attacks directed to point out potential intrinsic weakness of the algorithm.

It excludes the so-called *physical attacks* such as bribery, physical coercion, theft, reverse engineering, etc., which are possible for every algorithm and must be handled with different methods.

An important cryptanalytic tool used to analyze the RNA-Crypto System is based on statistical analysis. Our statistical analysis of RNA-Crypto System is based on a well-known battery of tests called Diehard or Dieharder in the new version (DH). The DH tests is a repertoire of statistical tests for measuring the quality of a set of random numbers. It is cited by NIST as one of the best statistical suite for testing randomness. It was developed by George Marsaglia over several years and first published in 1995 and then maintained and improved by Robert Brown at Duke University.

We focused our attention on the following tests:

1. Birthday spacings
2. Overlapping permutations
3. Ranks of matrices
4. Monkey tests
5. Count the 1s
6. Parking lot test
7. Minimum distance test
8. Random spheres test
9. The squeeze test
10. Overlapping sums test
11. Runs test
12. The craps test

All these tests are well described in the software package and in literature.

6.2. Our idea

In nature a nucleotides ribbon is a sequence of (almost always) 4 symbols (a, c, g, t) (sometimes fifth symbol u is added). In computer science a 'string' is a sequence of symbols $\{0, 1\}$. Thus, if we translate the 4 biological symbols as in Table 4, then we obtain the correspondence between bits and nucleotides which allows one to apply the DH tests both to sequences of *nucleotides* and to *output (encrypted)* sequences of our algorithm, thus allowing to compare the randomness of the two types of sequences by looking at the results of the tests.

The goal of the present section is to answer the following questions:

1. Do RNA ribbons have a good random behaviour according to DH tests?
2. Do RNA-Crypto System sequences have a good random behaviour according to DH tests?

Base	Binary
a	00
c	01
g	10
t|u	11

Table 4: Translation table

6.3. BIO results

6.3.1. The experiment

Since the statistical tests run well when more than 12 Mbytes of data are available, we used only some very long sequences of nucleotides, taken from on line standard free databases.

Assuming that we use 2 bits to code each base, in this way 1 byte of binary data encodes 4 bases. Thus we need sequences of at least 48 M bases to supply 12 Mbytes of binary data.

6.3.2. The protocol

1. Get a sequence from the database (longer than 48 M bases).
2. Translate it in binary mode.
3. Run the DH test on it.

6.3.3. The data

For the biological sequences we uses the following:

— *Caenorhabditis elegans* chromosomes I–V, complete sequences,
— *Wallaby*, whole genome,
— *Human* chromosome 14 complete sequence,
— *Drosophila melanogaster* some chromosomes, complete sequences,

all encoded using Table 4.

6.3.4. The results

In Table 5 we can see the results of our experiment using the biological data. They clearly show that bio-sequences are not random at all. This is due to several reasons, for example:

— Some parts of a *pre-m*RNA sequence could be highly repetitive (Satellite, Minisatellite, ...);
— Some parts of a *pre-m*RNA sequence could be made up by a very long sequence of the same nucleotide (Polyadenylation tail, ...).

n	Test name	Status
1	Birthday Spacings	FAIL
2	Overlapping Permutations	FAIL
3	Ranks of 31x31 and 32x32 matrices	FAIL
4	Ranks of 6x8 Matrices	FAIL
5	Monkey Tests on 20–bit Words	FAIL
6	Monkey Tests OPSO,OQSO,DNA	FAIL
7	Count the 1's in a Stream of Bytes	FAIL
8	Count the 1's in Specific Bytes	FAIL
9	Parking Lot Test	FAIL
10	Minimum Distance Test	FAIL
11	Random Spheres Test	FAIL
12	The Squeeze Test	FAIL
13	Overlapping Sums Test	FAIL
14	Runs Test	FAIL
15	The Craps Test	FAIL

Table 5: Bio results

6.4. RNA-Crypto System results

6.4.1. The experiment

In the RNA-Crypto System experiment we use long sequences of binary data encrypted with our protocol (approximately 100 MBytes of data for each experiment).

6.4.2. The protocol

The protocol used to estimate the randomness of RNA-Crypto System is:

1. run the DNACrypto program on a message of given length,
2. save the encrypted message,
3. run the DH test considering it as a *random* sequence.

6.4.3. The data

For the cryptographic sequences we uses:

— a 10 Mbytes file filled with ASCII char 'A',

— a 10 Mbytes file filled with random binary numbers,

— one of the above biological sequence,

all of them encoded using our Algorithm.

n	Test name	Passed
1	Birthday Spacings	PASS
2	Overlapping Permutations	PASS
3	Ranks of 31x31 and 32x32 matrices	PASS
4	Ranks of 6x8 Matrices	PASS
5	Monkey Tests on 20–bit Words	PASS
6	Monkey Tests OPSO,OQSO,DNA	PASS
7	Count the 1's in a Stream of Bytes	PASS
8	Count the 1's in Specific Bytes	PASS
9	Parking Lot Test	PASS
10	Minimum Distance Test	PASS
11	Random Spheres Test	PASS
12	The Squeeze Test	PASS
13	Overlapping Sums Test	PASS
14	Runs Test	PASS
15	The Craps Test	PASS

Table 6: Crypto results

6.4.4. The results

In Table 6 we can see the results of our experiment using the RNA-Crypto System data. They clearly show that RNA-Crypto System output sequences are random according to the DH tests.

6.5. First results — Differences

As expected the results correspond to our ideas on the sequences of nucleotides and maybe also to the ones about the Crypto System. Of course natural phenomena are not really random like a cryptographic system. Some obvious questions is why *pre-m*RNA sequences do not pass the statistical tests? An immediate answer is that they are not random (life is not random). But some of the motivations, from the statistical point of view, may be the following:

— Some parts of a *pre-m*RNA sequence can be highly repetitive (Satellite, Minisatellite, . . .);

— Some parts of a *pre-m*RNA sequence can be made up a very long sequence of the same nucleotide (Polyadenylation tail, . . .).

Is it possible to modify the RNA-Crypto System protocol to obtain the results similar to those obtained with truly biological sequences? In the following section we prove that this is indeed the case.

6.6. Changes: Informatics emulates biology

6.6.1. Modification of the cryptographic protocol — Phase I

The cryptographic model has been built so to avoid the appearance of long repeated blocks inside the same sequence in order to increase randomness properties. Here we introduce a new coding function ρ (the replicator function) that will artificially add such repeated sequences inside the code.

These additions can be of two classes:

— active (they act in some way with the system),
— passive (just redundancy).

6.7. Changes: Informatics emulates biology

6.7.1. Modification of the cryptographic protocol — Phase I

The cryptographic model has been built so to avoid the appearance of long repeated blocks inside the same sequence in order to increase randomness properties. Here we introduce a new coding function ρ (the replicator function) that will artificially add such repeated sequences inside the code. These additions can be of two classes:

— active (they act in some way with the system),
— passive (just redundancy).

6.7.2. Strategies

The *replicator* function is given by

DEFINITION 4 *Let $\rho : \mathcal{M} \times \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{C}$ be a junk function,*

$$\rho\,(\sigma, \kappa, o) = \Sigma = \iota, \tag{14}$$

where $\mathcal{C} \ni \iota = o_{\sigma,\kappa}$ and $o_{\sigma,\kappa}$ is a subsequence of o.

The subsequence ι can be determined (as an example) by the state of the key κ that can fix the portion to replicate of the message already encoded, (o), in terms of starting point and length. The symbol σ appears here just for compatibility with the definition of coding functions. In this way, ι is a replicated part of the encoded message, and then this mechanism implements the repeated sequences phenomenon.

6.7.3. Alter cryptographic protocol — Phase II

Due to its random nature the cryptographic model has not significant *all-equal* subsequences. We introduce a new coding function τ (the stutter function) that will add such *mono-symbol* subsequences artificially. Also in this case this sub-sequences can be created to be:

— active (they act in some way with the system),
— passive (just redundancy).

n	Test name	Status
1	Birthday Spacings	FAIL
2	Overlapping Permutations	FAIL
3	Ranks of 31x31 and 32x32 matrices	FAIL
4	Ranks of 6x8 Matrices	FAIL
5	Monkey Tests on 20–bit Words	FAIL
6	Monkey Tests OPSO,OQSO,DNA	FAIL
7	Count the 1's in a Stream of Bytes	FAIL
8	Count the 1's in Specific Bytes	FAIL
9	Parking Lot Test	FAIL
10	Minimum Distance Test	FAIL
11	Random Spheres Test	FAIL
12	The Squeeze Test	FAIL
13	Overlapping Sums Test	FAIL
14	Runs Test	FAIL
15	The Craps Test	FAIL

Table 7: New crypto results

6.7.4. Phase II — Strategies

The *stutter* function is given by the following definition.

DEFINITION 5 *Let* $\tau : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ *be a junk function,*

$$\tau(\sigma, \kappa) = \Sigma = \iota, \tag{15}$$

where $\mathcal{C} \ni \iota = \tilde{\iota}^{n_{\sigma,\kappa}}$ *and* $\tilde{\iota} \in (B_{\text{ex}} \cup B_{\text{in}})$, $n_{\sigma,\kappa}$ *is an integer.*

The subsequence ι can be determined (as an example) by the state of the key κ: the function τ fixes, *at random*, a symbol in the output set (e.g. $\tilde{\iota} \in (B_{\text{ex}} \cup B_{\text{in}})$) and extracts a number $n_{\sigma,\kappa}$ from the secret key κ. This mechanism implements the *polyadenylation tail.*

6.7.5. Notes

In both cases, the encoding functions can be replaced with other techniques.

6.8. Results

New results come from statistical analysis on the new cryptographic data. Note that the result in Table 7 holds for all the data in the cryptographic set introduced in 6.4.2.

6.9. New results — Differences

For the moment there is no proof that the security of the cryptographic system is not affected by the added redundancy: we cannot find any attack that could exploit this feature, and we are confident that the new system has the same level of security as the previous one. In fact, even if an eavesdropper could isolate the introns, the rest of the message would be exactly equal to the previous version. Then the safety remains the same (or better). Moreover, adding redundancy we obtained, as a side effect, the following features:

- Certain robustness to error in transmission. Indeed, in a very redundant system, the probability that an error prevents decoding is very low. This also suggests a particular interpretation of redundancy in RNA, i.e. protection against excessive mutations for example.
- Furthermore, an eavesdropper, during his attack, does not know whether the piece of code that he is trying to attack is an exon or an intron. This also suggests another particular interpretation of redundancy in RNA, i.e. protection against pathogens.

6.10. Conclusions

The results of the above tests lead to some conclusions which may be of more general significance than their application to the present class of algorithms. In fact, it is well known that some algorithms can produce good random sequences without being cryptographically secure.

The results in the present section point out that the converse conclusion is also true: some algorithms may have definitively bad statistical properties and yet be quite secure from the cryptographic point of view.

In fact we have seen that the sequences produced by the RNA-Crypto System algorithm have very good statistical properties but the complexity of breaking for the modifications of these algorithms (obtained by adding artificial redundancies) can be even greater than that of the original one.

Bibliography

[1] B. Lewin, *Gene VI*, Oxford University Press, 1997.

[2] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[3] M. Regoli, *Bio-Cryptography: A Possible Coding Role for RNA Redundancy*, Foundations of Probability and Physics 5, Vaxjo, Sweden, 24–27 August 2008, Series: AIP Conference Proceedings, Vol. 1101, L. Accardi, G. Adenier, A. Y. Khrennikov, C. Fuchs, G. Jaeger, J.-A. Larsson, S. Stenholm, eds., 2009.