A New Cryptographic Approach: 
Iterated Random Encryption (IRE)

Osvaldo Skliar∗ Sherry Gapper† Ricardo E. Monge‡

October 30, 2018

Abstract

A new cryptographic approach – Iterated Random Encryption (IRE) – is presented here. Although it is very simple, and easy to implement, it provides a very high level of security. According to this approach, a sequence of operations applied to a message ($M$) yields the encrypted message ($M_E$). In that series of operations, the one with the most important role is operation 6, which involves a random binary sequence (RBS) generated by using the Hybrid Random Number Generator (HRNG) or the Mathematical Random Number Generator (MRNG). A sequence of anti-operations applied to $M_E$ makes it possible to recover $M$.

Keywords: cryptography, random number generators, random binary sequence

2010 Mathematics Subject Classification: 11T71, 68P25, 94A60

1 Introduction

The objective of this article is to present a new cryptographic approach. That proposed here – Iterated Random Encryption (IRE) – is a novel approach. It is different from those currently predominant, such as RSA (Ron Rivest, Adi Shamir, Leonard Adleman), which are based on the tremendous practical difficulty of factoring the product of two very large prime numbers.

∗Escuela de Informática, Universidad Nacional, Heredia, Costa Rica. E-mail: oskliar@costarricense.cr. ORCID: 0000-0002-8321-3858.
†Universidad Nacional, Heredia, Costa Rica. E-mail: sherry.gapper.morrow@una.ac.cr. ORCID: 0000-0003-4920-6977.
‡Universidad de Costa Rica, San José, Costa Rica. E-mail: mongegapper@gmail.com. ORCID: 0000-0002-4321-5410.
Introductory information about cryptography may be found, for example, in [1], [2], [3] and [4].

The terminology used is expected to make this article clear even for readers less familiar with cryptography and cryptanalysis. An effort has been made for basic notions of mathematics (such as permutation) and of computer science (such as bit, byte and pixel) to suffice for a clear understanding of the essential ideas in the article. In the title of this paper, mention is made of an “approach”, rather than of a “method” because the main ideas explained here may be implemented in various ways.

The effective use of this approach leads to the generation of certain products. Each product is composed essentially of a) a computer program oriented toward encrypting information; and b) a computer program which makes it possible to decrypt the information encrypted by the program referred to in a) above. Those products will be referred to as $S_1, S_2, S_3, \ldots$ ($S$ for Software).

Each $S_i$, for $i = 1, 2, 3, \ldots$, can have various users. It is accepted that there will be a minimum of 2 users for each $S_i$, and either of them can encrypt information ($M$), which may consist of either a text or an image ($M_E$), and send the corresponding encrypted message to the other user. The latter will be able to decrypt the message using the computer program mentioned above in b).

$C_i$ stands for the set of users of $S_i$. Only the users who are members of a particular set of users $C_i$, for $i = 1, 2, 3, \ldots$, will be able to use $S_i$. Each user can be a member of more than one $C_i$; for instance, of $C_3$ and $C_7$. Obviously, as a member of $C_3$, a person will use $S_3$, and as a member of $C_7$, that person will use $S_7$.

2 How to Develop the Software Described

An explanation of how to develop a given $S_i$, for $i = 1, 2, 3, \ldots$, will be provided in this section. As seen above, $M$ stands for the information to be encrypted, and $M_E$ for the encrypted message. To encrypt a message, one must follow a sequence of operations.

2.1 Operation 1

It is admitted that the information to be encrypted $M$ is expressed by a sequence of bytes. Each byte is composed of a sequence of 8 bits. Each bit can be represented by a zero (0) or by a one (1).

Given that each byte is composed of a sequence of 8 bits, the number of different bytes is $2^8$ (that is, 256).

The number of bytes composing $M$ is then counted. If that number of bytes is a natural number $N_B$ such that $N_B$ is less than 10 (that is, $N_B < 10$), a number of bytes equal to $(10 - N_B)$, such that each of the latter symbolizes the whitespace character, is added to
that sequence of bytes. Thus, suppose that, for example, $M$ is comprised of a sequence of 8 bytes:

$$M : [\text{first} \text{ byte } ] [\text{second} \text{ byte } ] [ \text{third } \text{ byte } ] \cdots [\text{seventh} \text{ byte } ] [\text{eighth} \text{ byte } ]$$

To that sequence $M$, 2 bytes ($10 - 8 = 2$) are added. Within the symbolic system used, each of them symbolizes the whitespace character. In this way, the sequence $M_1$ of bytes is obtained:

$$M_1 : [\text{first} \text{ byte } ] [\text{second} \text{ byte } ] \cdots [\text{eighth} \text{ byte } ] [\text{ninth} \text{ byte } ] [\text{tenth} \text{ byte } ]$$

In $M_1$, the “ninth byte” in the sequence is equal to the “tenth byte”, and within the symbolic system used, both represent the whitespace character.

If that number of bytes in the sequence comprising $M$ is equal to 10 or greater than 10 (that is, $N_B \geq 10$), the sequence $M_1$ of bytes is considered equal to $M$.

It can be accepted that the operation carried out on $M$ (called “operation 1”) is applied by an operator called $O_1$. The action of $O_1$ on $M$ is symbolized as follows:

$$O_1\{M\} = M_1 \quad (1)$$

For operation 1, as for the following operations to be considered here, there is a corresponding “anti-operation”. In this case, ”anti-operator 1” is called $AO_1$. The action of $AO_1$ on $M_1$ is symbolized as follows:

$$AO_1\{M_1\} = M \quad (1')$$

If it is detected, for example, that $M_1$ is composed of a sequence of 10 bytes and that each of the last two bytes in that sequence symbolizes the whitespace character, then $AO_1$, when acting on $M_1$, will eliminate the last two bytes in that sequence, and $M$ will be recovered.

If, on the other hand, $M_1$ is composed of a number of bytes greater than 10, the operator $AO_1$, when acting on $M_1$, will leave $M_1$ unchanged. In that case, $M_1$ will be the same as $M$.

### 2.2 Operation 2

Suppose that the 256 bytes mentioned above are in a sequence of binary numbers in ascending order. That ordered sequence of 256 bytes will be known as “s”.

Operation 2 consists of a) randomly carrying out a permutation of the 256 bytes in that ordered sequence, and b) establishing a one-to-one correspondence between each byte of s and the byte which corresponds to it according to the permutation carried out.
The operator that acts on \( s \) to generate a new sequence of bytes will be known as “random permutation operator” \((O_{R.P.})\).

\[ O_{R.P.}\{s\} = s_p \]  

Thus, \( s_p \) is the “new sequence” of bytes generated, such that the first byte of \( s_p \) corresponds to the first byte of \( s \), the second byte of \( s_p \) corresponds to the second byte of \( s \), the third byte of \( s_p \) corresponds to the third byte of \( s \), and so on successively.

In this case, the corresponding “anti-operator” acts on \( s_p \) and generates \( s \).

\[ AO_{R.P.}\{s_p\} = s \]  

Hence, for example, if when carrying out this permutation of the bytes in \( s \), bytes \((01101100)\) and \((01010101)\) correspond to bytes \((01001101)\) and \((01110001)\) respectively, then when carrying out the inverse permutation, symbolized by \((2')\), bytes \((01001101)\) and \((01110001)\) would correspond to \((01101100)\) and \((01010101)\) respectively.

There are \(256!\) possible permutations of the 256 bytes in \( s \). Some of these permutations will assign to some byte in \( s \) (or to more than one byte in \( s \)) the same byte (or bytes) in \( s_p \). In particular, one of these \(256!\) permutations is that which assigns to each byte in \( s \) the same byte of \( s_p \). (In this case \( s = s_p \).) The probability that the randomly generated permutation would be this specific one is very low: \( \frac{1}{256!} \).

### 2.3 Operation 3

Consider the sequence \( M_1 \) of bytes mentioned in section 2.1. In “operation 3”, each byte of \( M_1 \) is replaced by the corresponding byte according to the operation symbolized in (2):

\[ O_3\{M_1\} = M_3 \]  

The corresponding anti-operation, whose nature appears obvious, can be symbolized as:

\[ AO_3\{M_3\} = M_1 \]  

### 2.4 Operation 4

Therefore, \( M_3 \) is a sequence of at least 10 bytes.

First, a description will be provided of how to proceed if \( M_3 \) is composed of only 10 bytes; and then of how to proceed if \( M_3 \) is a sequence of more than 10 bytes.

In the first case, the 10 bytes comprising \( M_3 \) are numbered according to the order they are in, and not according to their numerical values in binary arithmetic. Hence, the following sequence of bytes is obtained:

\( s_1 \): byte 1, byte 2, byte 3, byte 4, byte 5, byte 6, byte 7, byte 8, byte 9, byte 10
Next, one of the randomly selected $10!$ possible permutations of those 10 bytes of $s_1$ is carried out. Thus, in $s_2$ the new order of those 10 bytes in $s_1$ could be, for example, as follows:

$s_2$: byte 8, byte 3, byte 7, byte 4, byte 1, byte 10, byte 2, byte 9, byte 6 byte 5

Suppose now that $M_3$ is a sequence of more than 10 bytes; for example, of 15 bytes:

$s_3$: byte 1, byte 2, byte 3, byte 4, byte 5, byte 6, byte 8, byte 9, byte 10, byte 11, byte 12, byte 13, byte 14, byte 15

Let us admit that there is a “window” which makes it possible to visualize exactly 10 consecutive bytes; and that this “window” may be shifted, and that when shifted, it does not take with it the bytes which that window has made it possible to visualize.

Suppose that initially this shiftable “window” makes it possible to visualize the first 10 bytes of $M_3$. (Note that this sequence $M_3$ of 15 bytes was also called $s_3$.)

The first 10 bytes are subjected to an arbitrarily selected permutation. To simplify this description, let us accept that this random permutation is the same as that which went from sequence $s_1$ to sequence $s_2$. (It could have been any other of the $10!$ possible permutations.) Once that random permutation is complete, that “window” is shifted one byte. As a result of that shift, the new sequence of bytes which will be found in the “window” is as follows:

$s_4$: byte 3, byte 7, byte 4, byte 1, byte 10, byte 2, byte 9, byte 6 byte 5, byte 11

Then the 10 bytes of $s_4$ are renumbered: the first byte of $s_4$ (byte 3) is again called “byte 1”; the second byte of $s_4$ (byte 7) is called “byte 2”; the third byte of $s_4$ (byte 8) is called “byte 3”, and so forth, until the tenth byte of $s_4$ (byte 11) is called “byte 10”. Once the renumbering is completed, the “new window” appears as follows:

$s_5$: byte 1, byte 2, byte 3, byte 4, byte 5, byte 6, byte 7, byte 8, byte 9, byte 10

The preceding sequence $s_5$ of bytes is then subjected to the same permutation carried out with the first 10 bytes of $M_3$. This operation will result in a sequence of bytes that will be again numbered as follows:

$s_6$: byte 8, byte 3, byte 7, byte 4, byte 1, byte 10, byte 2, byte 9, byte 6 byte 5

The iteration process continues: 1) a new shift of the “window” by one byte, in the increasing order of the bytes of $M_3$; 2) the renumbering of the bytes that can be visualized in the new position of the window; and 3) the reiteration of the same permutation to which
the first 10 bytes of $M_3$ were subjected.

In the case considered of a sequence $M_3$ of 15 bytes, the window was shifted one byte 5 times. There were 5 processes of renumbering bytes and 6 permutations, the first of which is that applied to the first 10 bytes of $M_3$.

In general, if a sequence $M_3$ of $N_B$ bytes is considered, then there would be $(N_B - 10)$ shifts of the “window”, of one byte each; $(N_B - 10)$ processes of renumbering bytes, and $(N_B - 10 + 1)$ permutations of the type described of 10 bytes of each of the successive windows that must be taken into account.

Operation 4 may be symbolized as follows:

$$O_4\{M_3\} = M_4$$  \hspace{1cm} (4)

It is easy to observe that the inverse process, which goes from $M_4$ to $M_3$, can be symbolized as:

$$AO_4\{M_4\} = M_3$$  \hspace{1cm} (4')

2.5 Operation 5

According to the above, $M_4$ is a sequence of at least 10 bytes: $N_B \geq 10$. $M_4$. Given that each byte is a sequence of 8 bits, it is also possible to consider that $M_4$ is composed of a sequence of at least 80 bits: $N_b \geq 80$.

With that sequence of bits a procedure like that specified in section 2.4 is used, with the following differences: 1) The random permutation considered will not be of 10 bytes but rather of 80 bits; and 2) the shifts of an 80-bit “window” will not be shifted one byte but rather one bit, each time it occurs.

It is clear that for $M_4$, the following equation will be valid: $N_b = 8N_B$.

In operation 5, $(N_b - 80)$ shifts of one bit will be carried out. In addition, $(N_b - 80)$ processes of renumbering bits will take place. Note that if $N_b = 80$, the number of shifts will be equal to 0; that is, no shift will take place.

The same random permutation of 80 bits, which may be one of any of 80! possible permutations of those bits, will be applied to each of the successive “windows” of 80 bits. The number of these permutations will be equal to $(N_b - 80 + 1)$. Note that if $N_b = 80$ (that is, if $M_4$ is a sequence of only 80 bits), only one permutation will be carried out: that of 80 bits.

Operation 5 may be symbolized as:

$$O_5\{M_4\} = M_5$$  \hspace{1cm} (5)

Hence, it is thus easy to see that the inverse process, which goes from $M_5$ to $M_4$, can be symbolized as:

$$AO_5\{M_5\} = M_4$$  \hspace{1cm} (5')
2.6 Operation 6

In operation 6, two random number generators play an essential role (or at least one of them does): the Hybrid Random Number Generator (HRNG) [5], and the Mathematical Random Number Generator (MRNG) [6].

Regarding the HRNG, in [7] the following has been stated: “This HRNG constructs indisputable random numbers which may be used in the field of cryptography and many other computer applications starving for a random seed” (p. 312).

The article on the MRNG was mentioned in the MIT Technology Review [8].

A random binary sequence (RBS) can be generated by using the HRNG or the MRNG. This type of sequence used to generate the results presented in section 3 was composed of \(30,064 \times 10^6\) bits. By using these random number generators, there are no technical difficulties to obtain binary sequences composed of much larger numbers of bits.

Before specifying how the binary sequence is implemented effectively, the main idea of how it is used will be provided below.

Let there be a fragment of the sequence \(M_5\) of bits, such as the following:

\[
s_7: 1,0,1,1,0,0,1,0,1,0\]

It can be seen that the preceding fragment of \(M_5\) is a sequence of 10 bits.

To that fragment of \(M_5\), another fragment (also composed of 10 bits from the RBS), is assigned, having been generated by using the HRNG or the MRNG.

Suppose that the fragment, also composed of 10 bits, from the RBS, generated by using the HRNG or the MRNG is as follows:

\[
s_8: 1,0,0,1,1,0,0,0,0,1\]

Figure 1 displays how a one-to-one correspondence is established between each of the bits comprising the fragment of \(M_5\) and each corresponding bit of the above fragment of RBS.

\[
\begin{align*}
\text{Fragment of sequence } s_7 \text{ of } M_5: & \quad 1,0,1,1,0,0,1,0,1,0 \\
\text{Fragment of sequence } s_8 \text{ of RBS:} & \quad 1,0,1,1,0,0,1,1,0,0,1 \\
\end{align*}
\]

Figure 1: One-to-one correspondence between bits

As seen in figure 1, the first bit of the fragment of the RBS (i.e., 1) corresponds to the first bit of the fragment of \(M_5\) (i.e., 1); the second bit of the fragment of the RBS (i.e., 0)
corresponds to the second bit of the fragment of $M_5$ (i.e., 0); the third bit of the fragment of the RBS (i.e., 0) corresponds to the third bit of the fragment of $M_5$ (i.e., 1); and so on successively until the tenth bit of the fragment of the RBS (i.e., 1) corresponds to the tenth bit of the fragment of the $M_5$ (i.e., 0).

Attention is given below to all of the pairs of bits composed of one of the bits of the fragment of $M_5$ and its corresponding bit in the fragment of RBS. These pairs of bits are as follows:

$$(1, 1) (0, 0) (1, 0) (1, 1) (0, 1) (0, 0) (1, 0) (0, 0) (1, 0) (0, 1)$$

It is accepted that each of these pairs of bits generates a bit according to one of the following rules:

a) If both bits in a pair of bits are equal, a 1 (one) is generated; and if they are not equal, a 0 (zero) is generated; or

b) If both bits in a pair of bits are equal, a 0 (zero) is generated; and if they are not equal, a 1 (one) is generated.

Suppose that b, the second of the above rules, has been chosen. The result obtained when opting for b is displayed in figure 2.

Fragment of sequence $s_7$ of $M_5$: 1 0 1 1 0 1 0 1 1 0
Fragment of sequence $s_8$ of RBS: 1 0 1 1 0 1 0 0 0 1
Fragment of sequence $s_9$ of result: 0 0 1 0 1 0 1 1 1 1

Figure 2: The result of applying rule b above

It is evident that the process specified above must be applied to the entire sequence $M_5$ of bits, and not to only one fragment of it. This operation can be symbolized as follows:

$$O_6\{M_5\} = M_6 = M_E$$

As with the other operations above, it is clear that there is a sixth anti-operation, which when acting on the sequence of bits in $M_6$, generates the sequence $M_5$ of bits:

$$AO_6\{M_6\} = M_5$$

Additional information will be provided below on the effective use of the encryption and decryption programs described for any RBS that may be generated with the HRNG or the MRNG, and used in operation 6.
First, the bits making up the RBS to be used in successive “sixth operations” should be numbered. Thus, the first bit of the sequence is called bit 1, the second bit, bit 2, the third, bit 3, and so on successively until the \( n^{th} \) bit of the sequence is called bit \( n \). For example, the last bit of the RBS used to obtain the results presented in section 3 is called \((30,064 \times 10^6)\).

Second, each sequence of this type (that is, each RBS) is converted into a *loop*. Hence, just as bit 36 is followed by bit 37, and bit 350,258 is followed by bit 350,259, an operation is carried out so that the last bit in the sequence is followed by the first bit. In the case of the software used to obtain the results presented in section 3, an operation is carried out so that bit \( 30,064 \times 10^6 \) is followed by bit 1.

By using a loop (or closed sequence) of bits, the bits of the RBS found in operation 6 are never “used up”.

Although it is unlikely that exactly the same fragment of the loop may be found in two different instances of operation 6, for security reasons that loop should be composed of a number of bits as large as the space available in the memory of the computing device will permit.

Suppose that in a given operation 6, the encryption program begins to use the RBS as of bit 6,723 inclusively. So that the encrypted message to be sent can be decrypted by another user of the same \( S_i \), for \( i = 1, 2, 3, \ldots \), something must be added; for example, that message must be preceded by certain information: the number of the initial bit of the fragment of the RBS used in operation 6. In this case, the number is 6,723. (Of course, in other cases the numbers added to the encrypted messages will usually be different from 6,723.)

### 3 Results Obtained by Using the Encryption and Decryption Programs Described

#### 3.1 Encryption and Decryption of a Text

Consider the text in figure 3. This extract was taken from page 2 of one of our papers on random number generation [6]. This page contains the initial section of the contents of the paper (the Introduction and the first part of Section 1).
1 Introduction

1.1 Main issues addressed in scientific literature on random numbers and pseudorandom numbers

In scientific literature on random numbers and pseudorandom numbers, one of the main topics is the difference between genuine random numbers and others which are purportedly similar to them but not the same. The latter are usually referred to as pseudorandom numbers.

One widely accepted criterion is that purely mathematical procedures cannot generate true random numbers. John von Neumann’s famous remark may have helped to spread this opinion. He once stated jokingly: “Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin” [13]. If this is accepted, then true random numbers may be generated only with certain procedures involving processes found in Physics and other related fields.

In spite of the above, the main theoretical approaches developed to characterize the notion of randomness of certain numbers are independent of how those numbers are generated [1, 4, 8, 10, 12].

Another of the important topics considered in the scientific literature is that of tests for randomness. If one has certain numbers (to be characterized preliminarily in section 1.2, and more precisely, in section 2), it is usually accepted that diverse tests may be carried out to assess whether they are random, or in any case, whether they are “random enough”, according the purpose for which they will be used [5, 7]. These initially general or somewhat vague notions will be further clarified as this paper progresses.

1.2 An ideal random-bit generator (IRBG) and some possible applications

For a clearer understanding of the methods presented here, it is useful to consider an ideal machine that every so often (for instance, every millisecond) generates a digit with the following two characteristics: a) the digit is either a zero (0) or a one (1), and b) there is the same probability that the digit will be a 0 or a 1 (that is, for each digit generated by this ideal machine, the probability that it could be a 1 is 0.5, and that it could be a 0 is 0.5). This ideal machine will be called an “ideal random-bit generator” (IRBG). It will also be supposed that it has an ideal “memory” device (e.g., an “ideal hard disk”) which makes it possible to store a sequence of bits (generated by the IRBG) that is as long, or that contains as many bits, as desired. This sequence will be called a “binary string”.

One possible application of the IRBG is that of obtaining random numbers similar to those provided in a well-known publication of the Rand Corporation [9]. Let us admit, for example, that a sequence of five digits is required such that each of them can be one of the following: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. It can also be accepted that the probability of any one of these digits being in any of the five positions possible in any of the sequences of five digits is the same. (Hence, the probability is equal to 0.1.)

How could this type of five-digit sequence be obtained with an IRBG? The first four digits of a binary sequence generated by the IRBG and stored in a memory device will be used for

Figure 3: Text to be encrypted
Once we use the IRE to process the "PDF binary sequence" corresponding to the text displayed in figure 3, we obtain a sequence of bits (grouped in bytes), the first 2,400 of which are reproduced below:

```
10100001 00010110 11110000 00000000 00001110 01101110 00111010 10011100
01111111 00101110 00111010 10111100 11110011 10100111 10100101 01101111
11011111 10110001 01101100 01100100 00010101 01011000 00110100 01101110
01000010 10101111 00100000 01000000 10000110 00011111 00111011 10011001
01001110 00111000 10100001 01110000 00011110 00001011 00011110 10011101
11001110 11101101 10011110 11110011 10010010 01100110 01010001 11110000
10000111 10000111 01010100 00100111 10110011 01101000 11110000 01100101
10111011 01101110 00110100 00110111 01110110 01011101 01100100 10101101
10101101 01101101 01101101 01101101 01101101 01101101 01101101 01101101
11111110 11101100 11111100 11111100 11111100 11111100 11111100 11111100
10101101 01101101 01101101 01101101 01101101 01101101 01101101 01101101
01111100 10111001 10101101 10101101 10101101 10101101 10101101 10101101
01100000 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01111100 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01100000 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01111100 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01111100 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01111100 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01111100 10111001 10111001 10111001 10111001 10111001 10111001 10111001
10101101 01101000 01101001 01101001 01101001 01101001 01101001 01101001
01100000 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01100000 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01111100 10111001 10111001 10111001 10111001 10111001 10111001 10111001
01101101 01101000 11110000 11110000 11110000 11110000 11110000 11110000
10101101 01111111 01101101 01101101 01101101 01101101 01101101 01101101
01101101 01101101 01101101 01101101 01101101 01101101 01101101 01101101
01101101 01101101 01101101 01101101 01101101 01101101 01101101 01101101
01101101 01101101 01101101 01101101 01101101 01101101 01101101 01101101
01101101 01101101 01101101 01101101 01101101 01101101 01101101 01101101
01101101 01101101 01101101 01101101 01101101 01101101 01101101 01101101
11
```
11000001 10011011 00111111 10010100 10010011 01000110 11010011 01110100 00001000 00011111 10111010 11000100

The use of the corresponding decryption program makes it possible to recover the original text provided above.

If that same text is encrypted again immediately or after other encryption processes, a different encrypted message is generated. The reason for that is the following: The initial bit of the fragment of the RBS used for that second occasion in operation 6 is different from the initial bit of the other fragment of the RBS used previously.

### 3.2 Encryption and Decryption of an Image

Consider the image composed of $(24 \times 10^6)$ pixels (in figure 4). It belongs to a gallery of digital art [9].

![Figure 4: Fractal image to be encrypted](image)

12
Each pixel bears a color which is the result of a combination of certain levels of intensity of three different colors: red (R), green (G), and blue (B). The levels of each of these colors can vary between 0 and 255. In other words, there are 256 levels of color intensity for each of the three colors. That makes it possible to select from $256^3$ (that is, 16,777,216) colors for each pixel in the image. The combination of the levels of 0 for all three colors yields a black pixel, and the combination of the levels of 255 for each color yields a white pixel.

The specification, for each pixel, of its corresponding levels of color requires 3 bytes: one for the level of intensity of color for each of the three colors mentioned above.

Given a) the number of pixels in the image considered $(24 \times 10^6)$, and b) the 3 bytes necessary to specify the information for each pixel, the image can be expressed or coded by a sequence of $(3 \times 24 \times 10^6) = (72 \times 10^6)$ bytes. That sequence can be encrypted using the approach described in section 2.

The decryption program, when operating on the encrypted message, makes it possible to recover the image in figure 4.

It would consume too much space to use a sequence of $(8 \times 72 \times 10^6)$ – that is, $(576 \times 10^6)$ bits – corresponding to the encrypted message. To give an idea below of the appearance of the encrypted message, only the first 800 bits are shown, grouped in bytes.

```
11101000 11000110 10101100 01101111 00010010 00110110 01001100 00100110
11011000 10010111 11001101 00100010 11101000 11000000 11000100 00100001
11000010 11000100 00100101 10101111 11101101 01100111 00000111 11111111
00010010 11101000 00000110 11011000 11000001 00111111 11010110 11111111
11011111 11111101 10001011 11111000 11011100 11111010 01100000 01010000
00111101 11000001 11001101 01100111 00111111 11010110 01100000 01110000
00000001 00111110 01111001 00101001 10101010 01000000 11011111 11001100
10001001 11000101 11100000 10001010 10010001 01110111 11001100 01111000
00000001 11000101 11101111 00110001 11000000 11011000 11011000 10010000
00111000 11100000 11100010 10010010 01011111 11111011 11000000 10101100
00110010 11101111 11000011 00111011
```

The computer equipment used to run the encryption and decryption programs described here has a six-core Intel Core i7 5820K processor, of which one of the cores was used. Each of these programs was developed in two different programming languages: a) Microsoft Visual C# 2017 for the Windows version and b) GNU C++ (using the QT Toolkit) for the multi-platform version.

There was no significant difference between the encryption and decryption time for the text presented in 3.1 – both were less than 1 ms (one millisecond); nor was there a significant difference between the encryption and decryption time for the image presented in 3.2: both were less than approximately 3 ms (three milliseconds). The trials carried out
so far encrypting and decrypting information with some $S_i$, for $i = 1, 2, 3, \ldots$, show that these times increase linearly with the number of bytes in $M$.

### 4 Discussion and Prospects

As seen above, $O_1$, $O_{RP}$, $O_3$, $O_4$, $O_5$ and $O_6$ do not add information about the nature of the operators that they represent. The exact meaning of each of them is specified in sections 2.1, 2.2, 2.3, 2.4, 2.5 and 2.6, respectively. Anyone who, for reasons of mathematical rigor, prefers to eliminate them may do so because only a clear understanding of what they refer to is required.

Nevertheless, they serve to summarize the sequence of operations acting on $M$ (the message to be encrypted) and $M_E$ (the encrypted message itself).

$$O_6\{O_5\{O_4\{O_3\{O_1\{M\}\}\}\}\}\rangle = M_6 = M_E$$

Note that $O_{RP}$ does not appear explicitly in (7), but the operation it represents is indispensable for operation 3.

The sequence of anti-operations which, when acting on $M_6 = M_E$, makes it possible to obtain $M$ once again can be symbolized as follows:

$$AO_1\{AO_3\{AO_4\{AO_5\{AO_6\{M_E\}\}\}\}\}\rangle = M$$

Note that $AO_{RP}$ does not appear explicitly in (7'), but the anti-operation it represents is indispensable for the first anti-operation.

There is no doubt that operations 1, 2, 3, 4 and 5 each contribute to the security of the desired result. However, the most cryptographically important operation is the sixth one, in which the RBS obtained from the HRNG or the MRNG is used.

The computing device containing the programs for encrypting and decrypting messages should not be connected to the internet.

If a member of any $S_i$, for $i = 1, 2, 3, \ldots$, receives an encrypted message from another member of the same $S_i$, that member may do the following: First, that person must copy the message onto a device which is suitable for the task and download the message in the computational device used to encrypt and decrypt messages; then the message may be decrypted.

If someone desires to respond with an encrypted message, that person must 1) write the reply and encrypt it in the encryption-decryption device, and 2) copy the encrypted message onto a computer with access to the Internet, and send it to its respective destination.

Although the effectiveness and efficiency of this approach will be analyzed elsewhere, certain preliminary remarks about its effectiveness are included below.

The diverse random permutations used in IRE are a serious obstacle for any cryptanalyst interested in deciphering a message encrypted by using that approach.
As indicated above, there are \(256!\) possible permutations involved in operation 2 (described in section 2.2). If, when carrying out that permutation, a genuine random number generator is used, all of the possible permutations are equiprobable: Each one has a probability equal to \(\frac{1}{256!}\) of being selected.

Operation 4 (described in section 2.4) implies a permutation of 10 elements (in this case, 10 bytes). If a genuine random number generator is used to carry out this permutation, all of the possible permutations are equiprobable: Each one has a probability equal to \(\frac{1}{10!}\) of being selected.

Operation 5 (described in section 2.5) requires a permutation of 80 elements (in this case, 80 bits). If a genuine random number generator is used to carry out this permutation, all of the possible permutations are equiprobable: Each one has a probability equal to \(\frac{1}{80!}\) of being selected.

Therefore, the probability of discovering what those three permutations are, by mere chance, is the product of the three: \((\frac{1}{256!} \times \frac{1}{10!} \times \frac{1}{80!})\).

The random permutations considered, and especially the use of the RBS in operation 6, eliminate the patterns sought in cryptanalysis work. Of course, in operation 6 a different RBS must be used systematically and unavoidably for each of the \(S_i\).

To be certain about the genuine randomness of each RBS used, one must verify that each sequence passes the most rigorous randomness tests available: \([10]\) and \([11]\).

In future articles, consideration will be given to the use of several variants of the IRE for secure communication between users who do not share an \(S_i\).

References

[1] Easttom, C. (2016) *Modern Cryptography: Applied Mathematics for Encryption and Information Security*, McGraw-Hill Education.

[2] Martin, K. (2017) *Everyday Cryptography: Fundamental Principles and Applications*. Second Edition. Oxford University Press.

[3] Lange, T. and Takagi, T. (Eds.). (2017). *Post-Quantum Cryptography: 8th International Workshop*, PQCrypto 2017, Utrecht, the Netherlands, June 26-28, 2017, Proceedings(Vol. 10346). Springer.

[4] Baumslag, G., Fine, B., Kreuzer, M. and Rosenberger, G. (2015). *A Course in Mathematical Cryptography*. Walter de Gruyter GmbH.

[5] Skliar, O., Monge, R. E., Medina, V., Gapper, S. and Oviedo, G. (2011) “A Hybrid Random Number Generator (HRNG)”, *Revista de Matemática: Teoría y Aplicaciones*, 18(2), pp. 265-297.
[6] Skliar, O., Monge, R. E., Gapper, S. and Oviedo, G. (2012). “A Mathematical Random Number Generator (MRNG)”, arXiv:1211.5052 [cs.NA].

[7] Mohan, A. K., Devi, N., Sethumadhavan, M. and Santhya, R. (2018) “A Selective Generation of Hybrid Random Numbers via Android Smart Phones”, International Journal of Pure and Applied Mathematics, 118(8), pp. 311-317.

[8] MIT Technology Review (2012), “Other Interesting arXiv Papers This Week”, https://www.technologyreview.com/s/507791/other-interesting-arxiv-papers-this-week/.

[9] Skliar, O. (2015). “Image 80”, Digital Art Gallery, http://www.essencescope. com/.

[10] STS NIST. (2010). “Special publication 800-22”, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.

[11] Skliar, O., Monge, R. E., Oviedo, G. and Medina, V. (2009). “Indices of Regularity and Indices of Randomness for m-ary Strings”, Revista de Matemática: Teoría y Aplicaciones, 16(1), pp. 43-59.