MATRIX APPROACH OF AN ENCRYPTION SYSTEM BASED ON CELLULAR AUTOMATA AND ITS NUMERICAL IMPLEMENTATION

J. S. MURGUÍA, G. FLORES-ERAÑA, M. MEJÍA CARLOS, and H.C. ROSU

Author 1: UASLP-Facultad de Ciencias; Authors 2, 3: UASLP-IICO; Author 4: IPICyT

Int. J. Mod. Phys. C 23(11), 1250078 (2012)

The main components of an encryption system based on rule 90 cellular automata, i.e., two indexed families of permutations and a pseudorandom bit generator are implemented in a very flexible way through a convenient matrix approach.

1 Introduction

Many encryption systems based on different approaches are available in the literature, such as DES(Data Encryption Standard), AES(Advanced Encryption Standard), IDEA(International Data Encryption Algorithm), among others.[1, 2, 3] However, an efficient encryption process is still not an easy issue, since a complex encryption system implies more processing time, whereas a simple encryption system presents security problems to be predictable or decipherable. Hence, it is required to have an encryption system that has an optimal balance between both situations.

Cryptography techniques based on the application of cellular automata (CA) could be such a balancing possibility. Some encryption systems have already used CA as pseudo-random generators or combinations of them in the encryption algorithm.[4, 5, 6, 7, 8, 9, 10] The use of CA in cryptography is appealing since it can be adapted to an algorithm of massively parallel computations in computing architectures[11], and VLSI physical realizations.[12]

As is pointed out in Ref. [9], one of the first applications of chaotic CA in cryptography was carried out by Urías *et al* in [7]. They used a class of block cryptosystems comprising two indexed families of permutations and an asymptotically perfect pseudorandom number generator. Such encryption systems based on the synchronization phenomenon of the CA with rule 90 are known as ESCA systems. Their advantage is that they can be implemented using just one basic unit cipher.

In a previous work, we have used a sequence matrix to generate recursively the pseudo-random sequences.[13] In the following, we extend this matrix approach to almost all the components of the ESCA system. We show that just one sequence matrix can be used to implement effectively most of the matrices involved in the encryption system. This alternative matrix procedure is an encryption tool that fits well in the present-day digital technology.

The structure of this paper is as follows. Section 2 gives a general description of the encryption system considered. The way to implement the main elements of the encryption scheme with the proposed matrix approach is discussed in Section 3. In addition, the numerical implementation of this proposal is presented. Finally, the conclusions are drawn in Section 4. An Appendix containing basic CA definitions is included for the convenience of the reader.

2 Encryption System Model

2.1 The encryption system

Reference [7] describes a cryptosystem in terms of the synchronization phenomenon of cellular automata, which has been applied to devise the three main elements of the cryptosystem: the two families of permutations and the asymptotically perfect pseudorandom number generator.

In Urías *et al* [14] it was found that a pair of coupled CA with local rule 90 can synchronize if every pair of consecutive coordinates is separated by a block of $\left(2^k - 1\right)$ uncoupled sites, for k a positive integer. This encryption system is based on a symmetric algorithm, so it is demanded to know the seed that was used to generate the pseudo-random sequence of keys. In other words, the complete encryption scheme is private, and the encryption and decryption processes use the same deterministic generator that is initialized with a common seed. Figure 1 illustrates this type of encryption scheme.

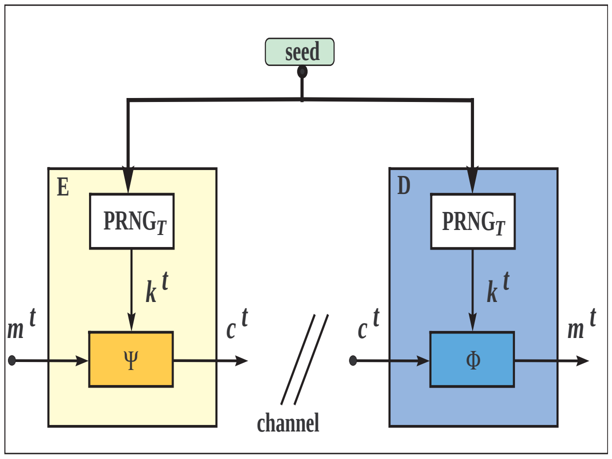


Figure 1: The encryption model used in this work with its main components: the indexed families of permutations, Ψ and Φ, and the pseudorandom generator of keys.

Following the notation of Ref. [7], the system basically transforms a plain-text sequence $\mathbf{m}$ to a sequence $\mathbf{c}$, called the cipher-text. The transformation $\mathbf{m} \mapsto \mathbf{c}$ is selected from an indexed family of permutations $\Psi = \{\psi_{\mathbf{k}} : M \to C | \mathbf{k} \in K\}$ by choosing an index $\mathbf{k}$ from the set of indices K. The sets M, C and K are all sets of binary words of length N, i.e., $\mathbb{Z}_2^N$, where $\mathbb{Z}_2 = \{0, 1\}$. The words in the sets M and C are called the clear-blocks and cipher-blocks, respectively, whereas the words in the set of indices K are the enciphering keys. This symmetric cipher encrypts blocks of plain-text bits of length N, using a different key for each block. To disclose from the sequence of cipher-blocks, the cryptosystem also provides the family of inverse permutations $\Phi = \{\phi_{\mathbf{k}} : C \to M | \mathbf{k} \in K\}$ such that for every $\mathbf{k} \in K$ one has $\mathbf{m} = \phi_{\mathbf{k}}(\varphi_{\mathbf{k}}(m))$. The pseudo-random generator of keys considered in this system has been used to select as random as possible the succession of permutations. With this, we have avoided that any intruder be able to infer any information about the text. The subindex T in PRNG, see Figure 1, indicates how many transformation have been employed to generate the pseudo-random sequences. In the case of $T = 1$, one transformation is used, whereas $T = 2$ implies that three coupled transformations have been considered. In Ref. [13] was checked the quality of the pseudo-random sequences for both cases, and it was found that the generated sequences pass all the NIST statistical tests.

2.2 The basic unit cipher

In order to implement this encryption system, Urías *et al*[7] considered a simple two-dimensional array composed of XOR logical functions, which we call basic unit cipher (BUC). In Fig. 2 the space-pattern of the BUC is displayed, which is defined as the $N \times N$ square pattern in the lattice that consists of N time-running words $(x_1^0, x_1^1, \ldots, x_1^{N-1}), \ldots, (x_N^0, x_N^1, \ldots, x_N^{N-1})$, along with the words $\mathbf{x} = (x_0^0, x_0^1, \ldots, x_0^{N-1})$ on the left side, $\mathbf{c} = (x_{N+1}^0, x_{N+1}^1, \ldots, x_{N+1}^{N-1})$ on the right side, $\mathbf{t} = (x_2^0, x_3^0, \ldots, x_{N+1}^0)$ on the top, and $\mathbf{m} = (x_1^N, x_2^N, \ldots, x_N^N)$ at the bottom. The latter pattern starts from the infinite initial state $(\ldots, x_{-1}^0, x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0, \ldots)$, that evolves according to rule 90 from $t = 0$ to $t = N = 2^k - 1$, where $i \neq 0$ and $i \neq N+1$, since x_0^t and x_{N+1}^t are externally assigned at each time t.[7]

In order to calculate the previous words of the BUC, the CA is required to be iterated along both the forward and backward directions, i.e., when the time evolution of the CA is running forward in time, and backward in time. These operations are illustrated in Fig. 3, where the forward evolution is given by (9), and the backward evolution is given by $x_{i+1}^t = \left(x_i^{t+1} + x_{i-1}^t\right) \bmod 2$. The symbol of a circled + represents a XOR gate and the connectivity of gates follows the automaton rule 90 in the respective evolution.

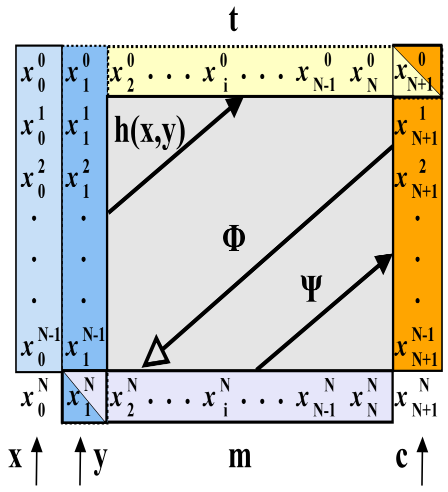

t
x_0^0 x_0^1 x_0^2 x_0^{N-1} x_0^N
x_1^0 x_1^1 x_1^2 x_1^{N-1} x_1^N
$x_2^0 \cdots x_i^0 \cdots x_{N-1}^0$ x_N^0
x_{N+1}^0 x_{N+1}^1 x_{N+1}^2 x_{N+1}^{N-1} x_{N+1}^N
h(x,y)
Φ
Ψ
$x_2^N \cdots x_i^N \cdots x_{N-1}^N$ x_N^N
x
y
m
c

Figure 2: The basic unit cipher, where the primitives of the encryption system are generated in terms of the evolution of the rule 90. The functions h and Ψ are determined by iterating the CA backward in time, whereas the function Φ is computed by running the CA forward in time.

The word located on the right side of the BUC is the cipher-block word $\mathbf{c} = (x_{N+1}^0, x_{N+1}^1, \ldots, x_{N+1}^{N-1})$, and it is obtained using the indexed family permutation $\Psi_\mathbf{x}$, i.e., $\mathbf{c} = \Psi_\mathbf{x}(\mathbf{m})$. This permutation Ψ can be calculated when the CA is iterated backward in time using the input words $\mathbf{x}$ and $\mathbf{m}$.

On the other hand, to bring the word $\mathbf{c}$ back to the plain text sequence $\mathbf{m}$, we consider the inverse permutation Φ, i.e., $\mathbf{m} = \Phi_\mathbf{x}(\mathbf{c})$. This permutation is computed when the automaton runs forward in time, but using the input words $\mathbf{c}$ and $\mathbf{x}$.

In a similar way, the function $\mathbf{t} = h(\mathbf{x}, \mathbf{y})$ can be generated when the automaton is also iterated backward in time. This function or transformation has been used to generate the pseudo-random sequences, and it takes into account the input words located on the left side of the BUC, i.e., the words $\mathbf{x}$ and $\mathbf{y}$. The result of the function $h(\mathbf{x}, \mathbf{y})$ is on the top of the BUC and is identified as $\mathbf{t} = (x_2^0, x_3^0, \ldots, x_{N+1}^0)$, where $\mathbf{x} = (x_0^0, x_0^1, \ldots, x_0^{N-1})$, and $\mathbf{y} = (x_1^0, x_1^1, \ldots, x_1^N)$.

Because the computation of the indexed permutations and the function h require the complete application of the local automaton rule *at all points* of the BUC, the "one-time" version of the algorithm has been considered as well space[15, 16]. The latter suggestion allows us to compute the required words without intermediate steps. While in Ref. [15] the Boolean functions for some coordinates of the function h for $N = 15$ bits are given, in Ref. [16] the respective expressions for the indexed family of permutations have been provided. As an example, in Fig. 4 we display the way the inverse permutation Φ is used, with input words $\mathbf{c}$ and $\mathbf{x}$, to compute the bit m_1 of the word $\mathbf{m}$ for the case $N = 3$ bits. Following the route traced by the arrows, the value of this coordinate is $m_1 = x_1 + x_3 + c_1$. However, this task can be difficult to handle as soon as the number of bits increases. To overcome this situation, a matrix approach is introduced here in the next Section of this work.

3 A Matrix Approach

To carry out effectively the complete encryption system with this approach, we basically need only one sequence matrix denoted by $\mathbf{Q}_N$. This matrix will implement most of the matrices involved in the encryption system.

3.1 The $\mathbf{Q}_N$ matrix

The sequence matrix $\mathbf{Q}_N$ of dimensions $N \times N$ can be generated initially from the vector $\boldsymbol{a} = [a_1, 0, \ldots, 0]$, where the component a_1 has a value of 1, and N is the number of bits, i. e., $\boldsymbol{a}$ is a vector with N components. This vector constitutes the first row of the matrix $\mathbf{Q}_N$ and the $(N-1)$ rows are generated applying the CA rule 90 of the previous row with fixed boundary conditions of zero to the left and right sides. For instance, for $N = 7$ we have that $\mathbf{Q}_7$ has the form

$$\mathbf{Q}_7 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{1}$$

With some basic matrix operations or transformations on $\mathbf{Q}_N$, we are able to implement the great majority of the stages involved in the encryption system, i.e., the family of permutations, and the pseudo-random number generator. Another way to generate the latter matrix is by means of relation (15), and taking into account the periodicity condition (10).

The case of indexed families of permutations

- The Ψ permutation

  For the encryption process, $\mathbf{c} = \Psi_{\mathbf{x}}(\mathbf{m})$, we require two matrices, $\mathbf{P}_N$ and $\mathbf{Q}_N$, such that

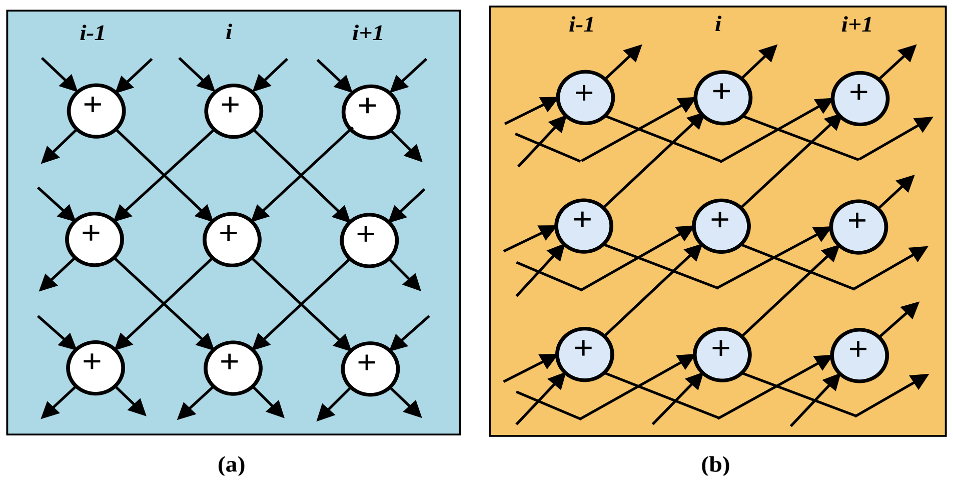


Figure 3: Time evolution of the cellular automata in a (a) forward, and (b) backward form.

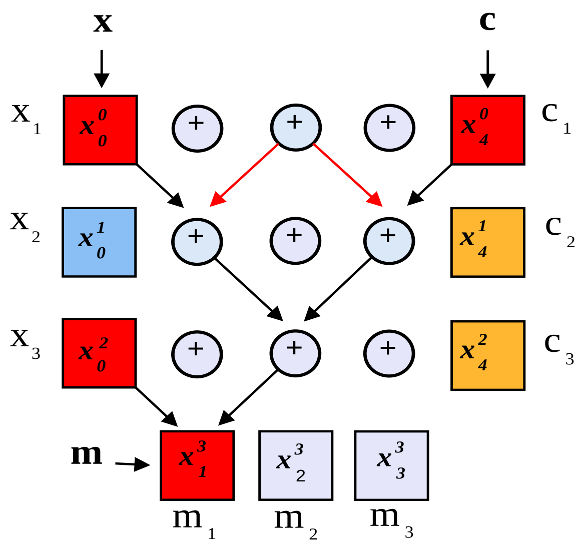


Figure 4: Generation of m_1 of the plain text sequence **m** when the CA is made to run forward in time for the case $N = 3$ bits.

$$\mathbf{c} = \Psi_{\mathbf{x}}(\mathbf{m}) = [(\mathbf{P}_N \times \mathbf{x}) + (\mathbf{Q}_N \times \mathbf{m})] \bmod 2. \tag{2}$$

These matrices have dimensions $N \times N = (2^n - 1) \times (2^n - 1)$, for $n = 1, 2, 3, \ldots$. In this process the main $\mathbf{Q}_N$ matrix is generated as discussed in Section 3.1.

On the other hand, the $\mathbf{P}_N$ matrix is initially generated from the vector $\mathbf{p} = [p_1, p_2, \ldots, p_N]$, which constitutes the first row, and the components with position index $j = (2^n + 1) - 2^{i+1}$, $i = 0, 1, 2, \ldots, (n-1)$, have a value of 1, and 0 otherwise. The $(N-1)$ rows are generated applying a right shift by one position of the previous row with a zero as its first value. As an example, for $N = 7$ we have the following form of the matrix $\mathbf{P}_7$

$$\mathbf{P}_7 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3}$$

- The Φ permutation

  The matrix implementation of the inverse permutation $\mathbf{m} = \Phi_{\mathbf{x}}(\mathbf{c})$ has a similar structure of (2), that is,

$$\mathbf{m} = \Phi_{\mathbf{x}}(\mathbf{c}) = [(\mathbf{R}_N \times \mathbf{x}) + (\mathbf{T}_N \times \mathbf{c})] \bmod 2, \tag{4}$$

  where the matrices have dimensions $N \times N = (2^n - 1) \times (2^n - 1)$, for $n = 1, 2, 3, \ldots$. From (2), $\mathbf{R}_N = [-\mathbf{Q}_N^{-1}\mathbf{P}_N] \bmod 2$, whereas the $\mathbf{T}_N$ matrix is just the inverse of $\mathbf{Q}_N$, i.e., $\mathbf{T}_N = \mathbf{Q}_N^{-1} \bmod 2$.

  Considering again $N = 7$, we have that the matrices $\mathbf{R}_7$ and $\mathbf{T}_7$ are

$$\mathbf{R}_7 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \mathbf{T}_7 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{5}$$

The case of pseudo-random generator

In a previous work,[13] we introduced a sequence matrix $\mathbf{H}_N$ to compute feasibly the pseudo-random sequences of N bits. This matrix has dimensions $(2N+1) \times (2N+1)$ and is formed by two matrices, $\mathbf{H}_{N_t}$ and $\mathbf{H}_{N_b}$, which constitute the top and bottom parts of $\mathbf{H}_N$, that is, $\mathbf{H}_N = \left(\mathbf{H}_{N_t}; \mathbf{H}_{N_b}\right)$. The matrix $\mathbf{H}_{N_t}$ has dimensions of $N \times (2N+1)$ elements, and the matrix $\mathbf{H}_{N_t}$ has dimensions $(N+1) \times (2N+1)$. Since the matrix implementation of the outer matrices is described in Ref. [13], we omit it.

The matrix $\mathbf{H}_N$ in terms of $\mathbf{Q}_N$ has the form

$$\mathbf{H}_N = \begin{pmatrix} \mathbf{H}_{N_t} \\ \mathbf{H}_{N_b} \end{pmatrix} = \left(\begin{array}{c:c} \mathbf{Q}_N^{-1} & \hat{\mathbf{Q}} \\ \hdashline \mathbf{I} & \mathbf{0} \end{array}\right), \tag{6}$$

where $\mathbf{I}$ is the identity matrix with dimensions $(N+1)\times(N+1)$, and $\mathbf{0}$ is a zero matrix with dimensions $N\times(N+1)$. The matrix $\hat{\mathbf{Q}}$ has dimensions $N\times(N+1)$, where the first $(N-1)$ rows and N columns are comprised of the matrix $\mathbf{Q}_N^{-1}$, but without the first row. The components of the N-row and the $(N+1)$-column of $\hat{\mathbf{Q}}$ have a value of 0 except in the intersection of both, which has a value of 1. For instance, for $N=7$ and considering (1) we have the following matrix

$$\mathbf{H}_7 = \begin{pmatrix}
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\
0&1&0&0&0&0&0&1&0&1&0&0&0&0&0 \\
1&0&1&0&0&0&0&0&0&0&1&0&0&0&0 \\
0&0&0&1&0&0&0&1&0&1&0&1&0&0&0 \\
1&0&1&0&1&0&0&0&1&0&0&0&1&0&0 \\
0&1&0&0&0&1&0&1&0&0&0&1&0&1&0 \\
1&0&0&0&1&0&1&0&0&0&0&0&0&0&1 \\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0
\end{pmatrix}, \tag{7}$$

where we can see the role of the matrix $\mathbf{Q}_7$ in the implementation of the matrix $\mathbf{H}_7$. When the pseudo random sequence length is short, it is suggested to use a modified generating scheme, which comprises three h coupled transformations.[13] In such a case we need two matrices with the structure of (6) for the inner transformations, and a matrix $\mathbf{H}_{N_t}$ for the output transformation. See [13] for more implementation details.

Notice that there are different manners to implement the complete encryption system using matrices, but we use this form because we think that it is one of the most convenient.

3.2 Numerical implementation

Considering the role of the matrix $\mathbf{Q}_N$ discussed in the previous Section, the procedure to carry out the complete encryption system proceeds as follows.

1. Provide the value of n to consider sequences of length $N=(2^n-1)$ bits.
2. Generate the matrix $\mathbf{Q}_N$.
3. Calculate the initial seeds of size N and $(N+1)$ bits.
4. Provide the initial seeds to generate the pseudo random sequences with the matrix corresponding to the pseudo random number generator in terms of the matrix $\mathbf{Q}_N$.
5. Calculate the matrix $\mathbf{P}_N$ and choose the task to be considered. For the encryption case, we use the matrix $\mathbf{P}_N$ and the matrix $\mathbf{Q}_N$ of step 2 to compute (2). Otherwise, the matrices $\mathbf{R}_N$ and $\mathbf{T}_N$ are generated to perform the decryption process by means of relation (4).

The complete numerical implementation of the encryption system was achieved under the LabVIEW graphical programming language, a trademark of National Instruments. To illustrate this implementation we first describe how we implement the main elements of the encryption system. Figure 5 depicts the time evolution of the pseudo random generator using three coupled transformations h, for $N = 31$ bits. In this case, the sequences $\mathbf{x}_i$ and $\mathbf{y}_i$, $i \in \{1, 2\}$ are the initial words for each internal transformation, h_1 and h_2. They generate the random sequences p_k and q_k of length of N bits, which are the initial sequences of the third h_3 transformation. Since any such transformation requires two words, one of N bits and the other of $(N + 1)$ bits, the missing bit is obtained applying an addition modulo 2 operation between the two respective least significant bit that become the most significant bits of their respective previous input sequences. At this point, it is clear that the three maps, p_k, q_k, and x_k generate pseudo random sequences, but only the x_k sequence is released.

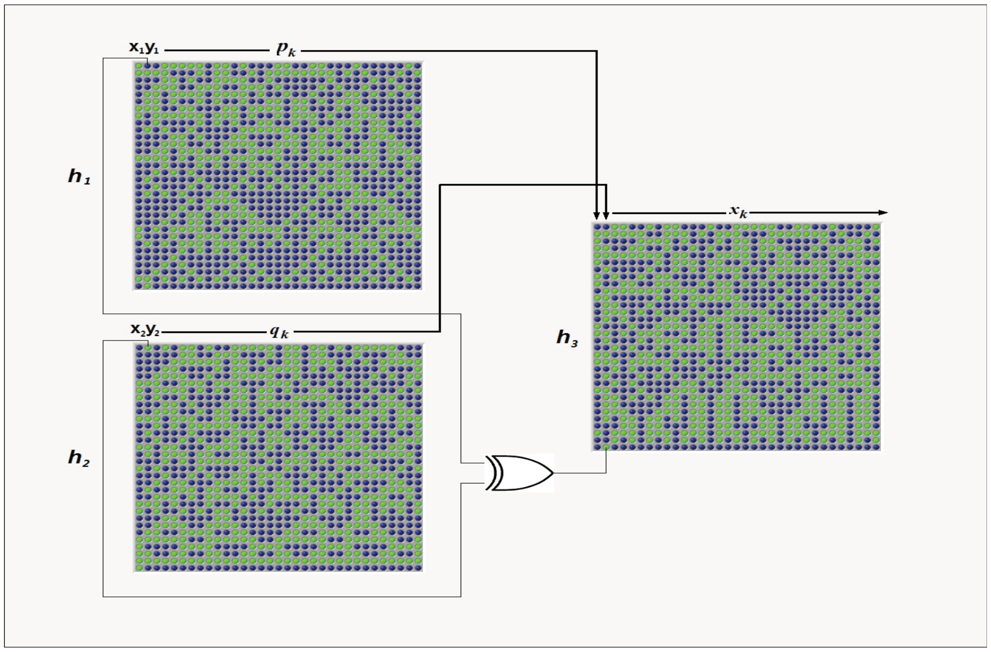


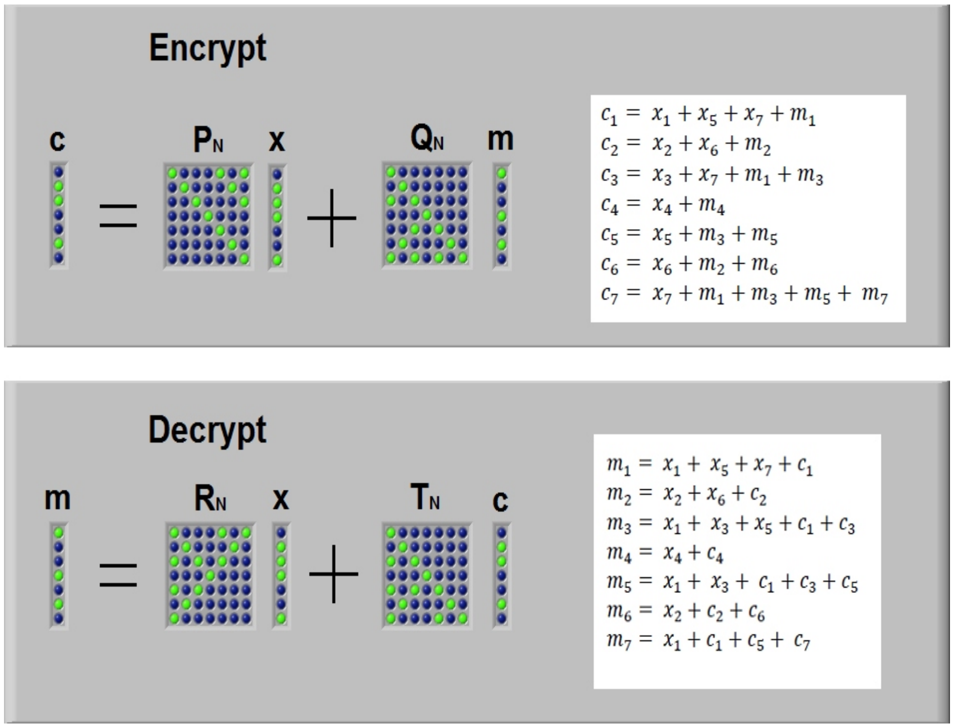
Figure 5: Time evolution of the pseudo random generator that makes use of three coupled transformations for $N = 31$ bits.

On the other hand, Fig. 6 shows one example of the indexed families of permutations for sequences of $N = 7$ bits. If we want to carry out an encryption process, we consider the top figure, whereas for the decryption process we look to the bottom figure. On the right side, the respective Boolean expressions of the words $\mathbf{c}$ and $\mathbf{m}$ are displayed, and the obtained results can be compared with the results of the matrix approach.

Figure 7 presents the main virtual instrument of the encryption system, with the results obtained when the encryption process is applied to an audio signal. A length of $N = 31$ bits has been considered for this case study, and a pseudo random sequences generator of three coupled transformations has been used.

4 Conclusions

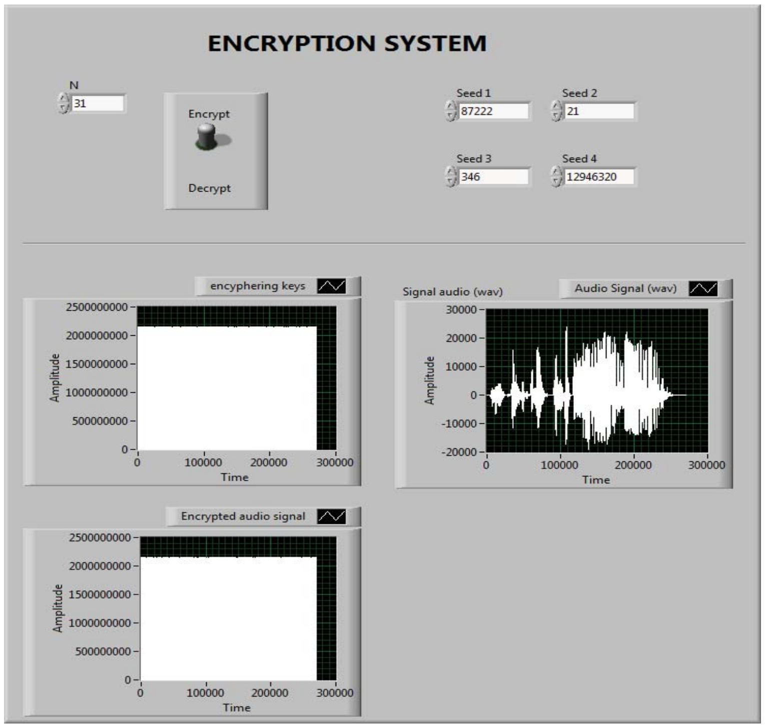
We have proposed a matrix approach to implement in an adjustable way the main elements of an encryption system: the two indexed families of permutation and the pseudo random bit generator. Such a system considers a certain cellular automata rule which occurs in different cryptographic applications,.



$$\begin{aligned}
c_1 &= x_1 + x_5 + x_7 + m_1\\
c_2 &= x_2 + x_6 + m_2\\
c_3 &= x_3 + x_7 + m_1 + m_3\\
c_4 &= x_4 + m_4\\
c_5 &= x_5 + m_3 + m_5\\
c_6 &= x_6 + m_2 + m_6\\
c_7 &= x_7 + m_1 + m_3 + m_5 + m_7
\end{aligned}$$

$$\begin{aligned}
m_1 &= x_1 + x_5 + x_7 + c_1\\
m_2 &= x_2 + x_6 + c_2\\
m_3 &= x_1 + x_3 + x_5 + c_1 + c_3\\
m_4 &= x_4 + c_4\\
m_5 &= x_1 + x_3 + c_1 + c_3 + c_5\\
m_6 &= x_2 + c_2 + c_6\\
m_7 &= x_1 + c_1 + c_5 + c_7
\end{aligned}$$

Figure 6: Numerical implementation of the indexed families of permutations $\mathbf{c} = \Psi_{\mathbf{x}}(\mathbf{m})$ (top), and $\mathbf{m} = \Phi_{\mathbf{x}}(\mathbf{c})$ (bottom), considering $N = 7$ bits.


ENCRYPTION SYSTEM

N: 31

Seed 1: 87222
Seed 2: 21
Seed 3: 346
Seed 4: 12946320

Figure 7: Numerical implementation of the encryption system under the LabVIEW graphical programming language, a trademark of National Instruments.

This matrix procedure appears to be an attractive and flexible way to implement an encryption system completely or partially. We believe that this proposal can be an accessible and affordable solution for encryption issues, since the cryptosystem will be a flexible and reconfigurable system that fits in the present digital technology with a wide range of applications.

Acknowledgments

G. Flores-Eraña is a doctoral fellow of CONACYT (México) in the program of "Ciencias Aplicadas" at IICO-UASLP. HCR thanks CONACyT for a sabbatical fellowship.

Appendix CA: Some Definitions

A cellular automaton (CA) is a discrete nonlinear dynamical system that evolves at discrete time steps. It consists of a chain of N lattice sites where each site is denoted by an index i. A dynamic variable x_i that can take only k discrete values is associated to each site i. Considering the common value of $k = 2$, we have that $x_i = 0$ or 1, and the state space of a CA of size N is the set $\Omega = \mathbb{Z}_2^N$ of all sequences of N cells that take values from $\mathbb{Z}_2 = \{0, 1\}$, where $\mathbb{Z}$ is the set of integers. The evolution of the CA is defined by the repeated iteration of an evolution operator $\mathcal{A} : \mathbb{Z}_2^N \to \mathbb{Z}_2^N$, and the automaton state at time $t \geq 0$ is denoted by $\underline{x}^t \in \mathbb{Z}_2^{\mathbb{Z}}$, where its evolution is defined iteratively by the rule $\underline{x}^{t+1} = \mathcal{A}(\underline{x}^t)$.

A CA with radius r has a neighborhood of $2r + 1$ cells, a cell with r neighbors on each side. The evolution of a CA of radius 1 has the form

$$x_i^{t+1} = (c_{-1}x_{i-1}^t + c_0 x_i^t + c_1 x_{i+1}^t) \bmod 2. \tag{8}$$

where c_{-1}, c_0, and c_1 are integer constants 0 or 1. The CA that evolves according to the local rule

$$x_i^{t+1} = \left(x_{i-1}^t + x_{i+1}^t\right) \bmod 2, \tag{9}$$

corresponds to the CA with rule 90. Since the lattice used in this work is finite in extent, we consider the following periodic boundary condition of equation (9)

$$x_{i+N}^t = x_i^t, \qquad \text{for all } i, \tag{10}$$

that is, the arithmetic on the subscripts of (9) is performed modulo N. Thus x_N^t should be replaced by x_0^t , x_{N+1}^t by x_1^t, and so on.

As is pointed out in Ref. [17, 18] the complete configuration of a cellular automaton is specified by the values of its N sites, and it may be represented by a generating polynomial $X^t(p)$

$$X^t(p) = \sum_{i=0}^{N-1} x_i^t p^i. \tag{11}$$

Furthermore, the time evolution can be carried out by means of the multiplication of the generating polynomial by a fixed dipolynomial, $\mathbf{T}(p) = c_{-1}p + c_0 + c_1 p^{-1}$, i.e.,

$$X^{t+1}(p) = \mathbf{T}(p) X^t(p) \mod (p^N - 1), \tag{12}$$

where the arithmetic is performed modulo 2.

Kar *et al.* [18] presented an explicit formula to compute the configuration of additive cellular automata rules at any time step without any explicit simulation. Our interest is for the additive rule 90 (9), where $c_0 = 0$, $c_{-1} = c_1 = 1$, so the fixed dipolynomial is

$$\mathbf{T}(p) = (p + p^{-1}) \bmod 2. \tag{13}$$

Taking an impulse as initial configuration, i.e., the first row is comprised of a "1" in the center and "0"s elsewhere, we have that

$$X^{(t)}(p) = \left(p + p^{-1}\right)^t \mod (p^N - 1),$$

which can be written as

$$X^t(p) = \sum_{k=0}^{t} \binom{t}{k} p^{2k-t} = \sum_{\substack{i=-t \\ (t+i)\text{even}}}^{t} \binom{t}{(t+i)/2} p^i \qquad \mod (p^N - 1), \tag{14}$$

where

$$x_i^t = \binom{t}{(t+i)/2} \bmod 2, \qquad -t \leq i \leq t. \tag{15}$$

Therefore, from relation (15) and the periodicity condition (10) it is possible to get the explicit values of the rule 90 for the standard span, i.e., for $i = 0, \ldots, N-1$.

References

[1] C. Paar, and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners* (Springer-Verlag Berlin Heidelberg, 2010).

[2] T. W. Cusick, and P. Stanica, *Cryptographic Boolean Functions and Applications* (Academic Press, 2009).

[3] C. K. Koc, *Cryptographic engineering* (Springer Science+Business Media, LLC, 2009).

[4] S. Wolfram, *Lecture Notes Comput. Sci.* **218**, 429 (1986).

[5] S. Nandi, B. K. Kar, and P. P. Chaudhuri, *IEEE Transactions on Computer* **43**, 1346 (1994).

[6] M. Sipper, and M. Tomassini, *Int. J. Mod. Phys. C* **7**, 181 (1996).

[7] J. Urías, E. Ugalde, and G. Salazar, *Chaos* **8**, 819 (1998).

[8] F. Seredynski, P. Bouvry, and A. Y. Zomaya, *Parallel Computing* **30**, 753 (2004).

[9] A. Fúster-Sabater, and P. Caballero-Gil, *Applied Soft Computing* **11**, 1876 (2011).

[10] S. Das, and D. R. Chowdhury, *Lecture Notes in Comput. Sci.* **6584**, 77 (2011).

[11] G. Zied, M. Mohsen, Z. Medien, and T. Rached, *Int. J. Comput. Sci. Eng. Syst.* **2**, 185 (2008).

[12] P. D. Hortensius, R. D. McLeod, and H. C. Card, *IEEE Transactions on Computer* **38**, 1466 (1989).

[13] J. S. Murguía, M. Mejía Carlos, H. C. Rosu, and G. Flores-Eraña, *Int. J. Mod. Phys. C* **21**, 741 (2010).

[14] J. Urías, G. Salazar, and E. Ugalde, *Chaos* **8**, 814 (1998).

[15] M. Mejía, J. Urías, *Discrete Cont. Dynamical Syst.* **7**, 115 (2001).

[16] M. Mejía Carlos, Ph. D. Thesis, Universidad Autónoma de San Luis Potosí, SLP (2001).

[17] O. Martin, A. M. Odlyzko, and S. Wolfram, *Communications in Mathematical Physics* **93**, 219 (1984).

[18] B. K. Kar, A. Gupta, and P. P. Chaudhuri, *Information Sciences* **72**, 83 (1993).