# Genetic Operators Applied to Symmetric Cryptography

Jefferson Rodríguez, Brayan Corredor, César Suárez *

Systems Engineer, District University Francisco José de Caldas, Bogotá (Colombia)

Received 28 November 2018 | Accepted 14 June 2019 | Published 31 July 2019

## Abstract

In this article, a symmetric-key cryptographic algorithm for text is proposed, which applies Genetic Algorithms philosophy, entropy and modular arithmetic. An experimental methodology is used over a deterministic system, which redistributes and modifies the parameters and phases of the genetic algorithm that directly affect its behavior, carrying out a constant evaluation using the fitness function, in order to optimize the results. An independent encryption is established for the auxiliary key, using a main key, in charge of increasing security. The tests are performed over different text sizes, manipulating the parameters and criteria proposed to obtain their appropriate values. Finally, a comparison is presented against the following cryptographic algorithms DES (Data Encryption Standard), RSA (Rivest, Shamir and Adleman) and AES (Advanced Encryption Standard), exposing factors such as processing time, scalability, key size, etc. It is shown that the proposed algorithm has a better performance.

## Keywords

Genetic Algorithms, Symmetric Cryptographic, Entropy, Modular Arithmetic, Computer Security.

DOI: 10.9781/ijimai.2019.07.006

## I. Introduction

COMPUTER security has always been the discipline responsible for the protection of data stored in a physical or logical computer system. In recent years, technological growth has been such that this security has focused on minimizing any risk to information, combating all types of vulnerabilities that may exist in a network environment. Cryptographic algorithms confront computer attacks that are increasingly complex, forcing them to evolve by participating in recent and reliable methods [1].

There are many algorithms of this type, categorized in Symmetric (private key) and Asymmetric (public key). In symmetric cryptography, a key is used to encrypt and decrypt data, while in asymmetric cryptography two keys are used to perform these tasks; a public figure and a private decipher (e.g., RSA [Rivest, Shamir and Adleman]). Encryption is based on intensively computed mathematical functions and deciphering is usually the reverse process using the key(s) [2], for this DES (Data Encryption Standard) uses a 64-bit key, while AES (Advanced Encryption Standard) uses keys of 128, 192 and 256 bits [3].

In recent years, several investigations have been presented that relate cryptography with genetic concepts, making this field known as an alternative to solve computer security problems. B. Beegom and S. Jose [4] present an asymmetric cryptographic model based on a genetic approach and expose an efficient method in which they make use of the complexity of the DNA chains. N. Srilatha and G. Murali [5] define their work as an efficient three-level cryptographic technique, based on processes that use DNA sequences to transmit information.

Nowadays, for data encryption through the Internet, the HTTPS (Hypertext Transfer Protocol Secure) protocol uses SSL/TLS-based encryption to create a secure channel to shared data [6]. The cryptographic protocols TLS (Transport Layer Security) and SSL (Secure Sockets Layer) used by HTTPS use asymmetric cryptography which uses a pair of keys for sending information, authenticating the receiver more reliably [7].

The presented proposal uses symmetric cryptography and supports its security on the keys more than on the same algorithm, since under an attack it is useless to have knowledge of the algorithm used in the encryption if the key(s) are unknown [13]. Therefore, the use of the phases of the GA is proposed, taking advantage of numerous initial conditions that are included in the key, achieving a quite secure encryption. It is important to add that the security increases based on the length of the key, but at the same time, its access is slowed, which causes more processing time [14].

The proposed algorithm is based on operators used in Genetic Algorithms (GA), adapted to encrypt and decrypt text. The principles of GA were exposed by Holland in 1975 [8], and described more broadly by Goldberg in 1989 [9], Poli, Langdon, McPhee, Michelle and Davis [23].

The main limitation at the time of the creation of a cryptographic algorithm is the idea of innovation, since, currently, there are large amounts of algorithms of this type. With this motivation and in order to generate an original proposal, we decided to take advantage of GA techniques in cryptographic development, making some modifications without neglecting its philosophy. These alterations arise from the interest of creating a different work, which through a clear exemplification demonstrates that it is possible to create developments in the area of computer security using basic concepts. Our work takes the randomness and the order of operation of the genetic phases to arrive at a deterministic development, proposing in addition the use of two keys (auxiliary and main) without ceasing to be a symmetric algorithm.

## II. Theoretical Background

### A. Genetic Algorithms

The Genetic Algorithms are an adaptive method used to solve search and optimization problems, inspired by biological evolution

* Corresponding author.

E-mail addresses: jefsrodriguezr@correo.udistrital.edu.co (J. Rodríguez), bjcorredorp@correo.udistrital.edu.co (B. Corredor), casuarezp@udistrital.edu.co (C. Suárez)

[10] and based on molecular genetics; a genetic algorithm makes use of terms specific to this field, as well as its main phases: Selection, Crossing and Mutation [11].

***Selection***: The selection is the stage in which each chromosome (representing a potential solution to the problem) goes through a process of evaluation on a certain fitness value, where some of them are chosen to be later transformed by the crossing and/or mutation operators. In this process the number of chromosomes, genes and alleles of the genotype is kept constant [12].

***Crossing***: The cross is a genetic operator that allows information to be exchanged between two chromosomes to form a new one. For binary-chain individuals, ring, one-point, two-point, and uniform crossings are often used [12]. Fig. 1 shows the offspring that occurs with a crossing point 4 between two parents of length 5.

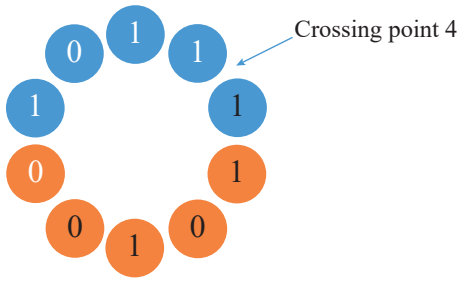



Fig. 1. Example of ring Crossing example.

***Mutation***: The mutation changes the value of the selected allele [12]. In Fig. 2 the mutation of the selected allele is shown as an example.

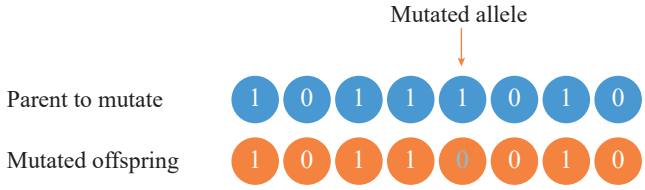



Fig. 2. Illustration of Mutation.

***Fitness Function***: It is the quality control within the GA and plays a vital role in the guide of the same, it helps to explore the search space more effectively and efficiently [12].

The main idea of the GAs is to reproduce the random nature where the population of individuals adapts to their environment through natural selection, as well as the behavior of the ecosystem. Once the genetic representation of the initial population has been defined, a set of stochastic operators are applied iteratively: selection, crossing and mutation; under certain quality criteria called fitness function. The application of the GAs to optimization problems provides flexibility and adaptability, combined with the robustness and the advantages of the global search [16].

In the philosophy of genetic algorithms, a set of terms of the genetic language have been adopted to clarify and unify the concepts of development of this type of algorithms, as follows: Allele: Each bit is called Allele. Gene: Each group of alleles is called Gen. Chromosome: Each group of genes is called Chromosome.

### *B. Computer Security*

Cryptographic algorithms, have mainly three measurable characteristics: Capacity, Security and Robustness. Capacity is related to the amount of information that the algorithm can process. Security refers to the protection that data receive against possible attacks. Robustness is the resistance that the method has in its entirety against external attacks [15].

There are different types of computer attacks that try to obtain unauthorized access to a network service, among the most common is the Brute Force attack, which makes repeated and systematic attempts using possible credentials, based on different parameters that usually come from sets of credentials set by default, commonly used or valid in previous attacks [17].

### *C. Related Concepts*

The work below has a strong relationship with Entropy, known as the measure of the uncertainty associated with a random variable [18], conceived as a measure of disorder, as well as the repetition of certain combinations. It is measured in bits, where the number of information bits of each character is given by:

$$S = log_2 k \quad (1)$$

where k is the total number of characters. The entropy of a random source is the expected information content of the symbols it has, that is, the expected uncertainty of each symbol, knowing only the distribution according to the symbol [19].

As a last concept, due to its constant use in the different phases of the proposed algorithm, we have the Modular Congruence or Modular Arithmetic, defined as an arithmetic system for whole-number equivalence classes introduced by Carl Friedich Gauss in his book Disquisitiones Arithmeticae (1801) [20]. It is defined as follows:

*Let a and b be any integer, and n a positive integer. If n | (a-b) we say that a and b are congruent modulo n and we write* [21]:

$$a \equiv b \ (mod\ n) \quad (2)$$

## III. Methodology

An investigation-action is established, whose objective is the development of a cryptographic text Algorithm based on principles of computer security and genetic algorithms. The development and the tests are carried out in MATLAB R2017a, with academic license. The chosen methodology is experimental and is summarized in the diagram shown in Fig. 3.

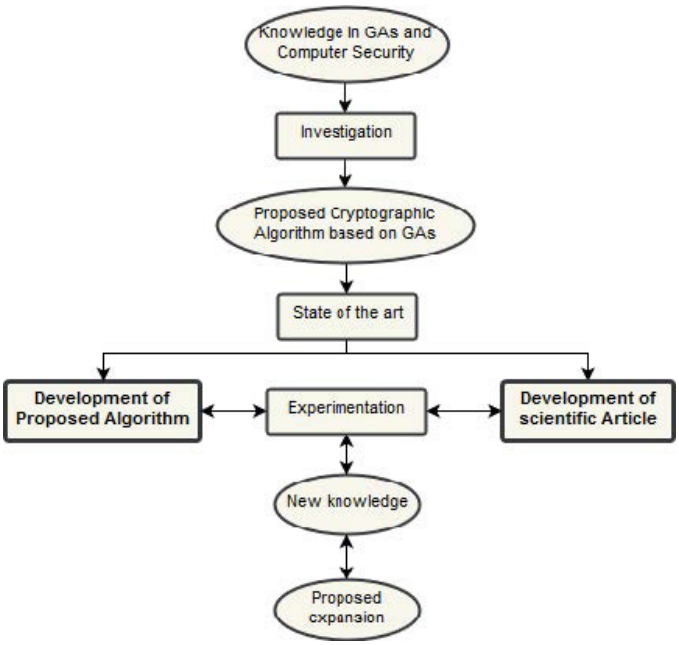



Fig. 3. Applied Methodology.

The methodology is based on previous knowledge in the area of GA and Computer Security, which is accompanied by a deep research and a state of the art description, which aim to establish a clear idea of the development of a symmetric cryptographic Algorithm that takes advantages of the phases of the GA and other related concepts in the matter.

## IV. Proposed Cryptographic Algorithm

The proposal below is part of a deterministic system, which implies elimination of randomness and full knowledge of the input variables. It is important to consider that in the nomenclature worked below; the parameters are variables established by the user under certain conditions, while the criteria refer to constant data used to optimize the algorithm and established under a range of tests. The flow diagram (Fig. 4) explains the operation of the proposed algorithm.

### *A. Initial Population Preparation*

It starts from a random message to be encrypted, as a string of characters based on the Latin alphabet and modern English. It is random in nature since it is unknown. As an example, we have the text "Hola mundo" (10 characters counting the space). Table I (annexes) shows the conversion of the entire message to American Standard Code for Information Interchange (ASCII) and then to binary, in this strict order and **taking the interspaces**.

### *B. Encryption Process*

Once the Initial Text has been converted, the Initial Population is taken as the starting point of the GA. The ring-type crossing of the population is carried out with a single point, taking said point from the initial conditions. One child is generated for each pair of parents (genes). Each parent crosses 2 times, with a different partner, which originates a new generation, replacing the previous one. In order to achieve a greater diversity of genes, it is proposed to use the reversalCriterion, which allows to revert the order of the second parent digits of each crossing (or iteration) depending on the value of the criterion. Applying a crossParameter of 6 and a reversalCriterion of 3 (highlighted in bold) the offspring shown in Table I (annexes) is obtained.

The Last Generation is taken to continue with the Selection of the individuals, using the proposed Selection Equation for the Mutation, also called Mutation Clock, taken from the definition of modular congruence (equation 2), and defined as follows:

$$i \equiv 0 \ (mod\ k) \tag{3}$$

Where $i$ is the position of each allele $A_i$ being $1 \leq i \leq n$, where $n$ is the Population Total; and $k$ is he mutationParameter defined between $0 < k < n$ in the initial conditions.

The mutation clock selects the alleles to be mutated, taking the last generation as a string of bits **without counting spaces between genes**.

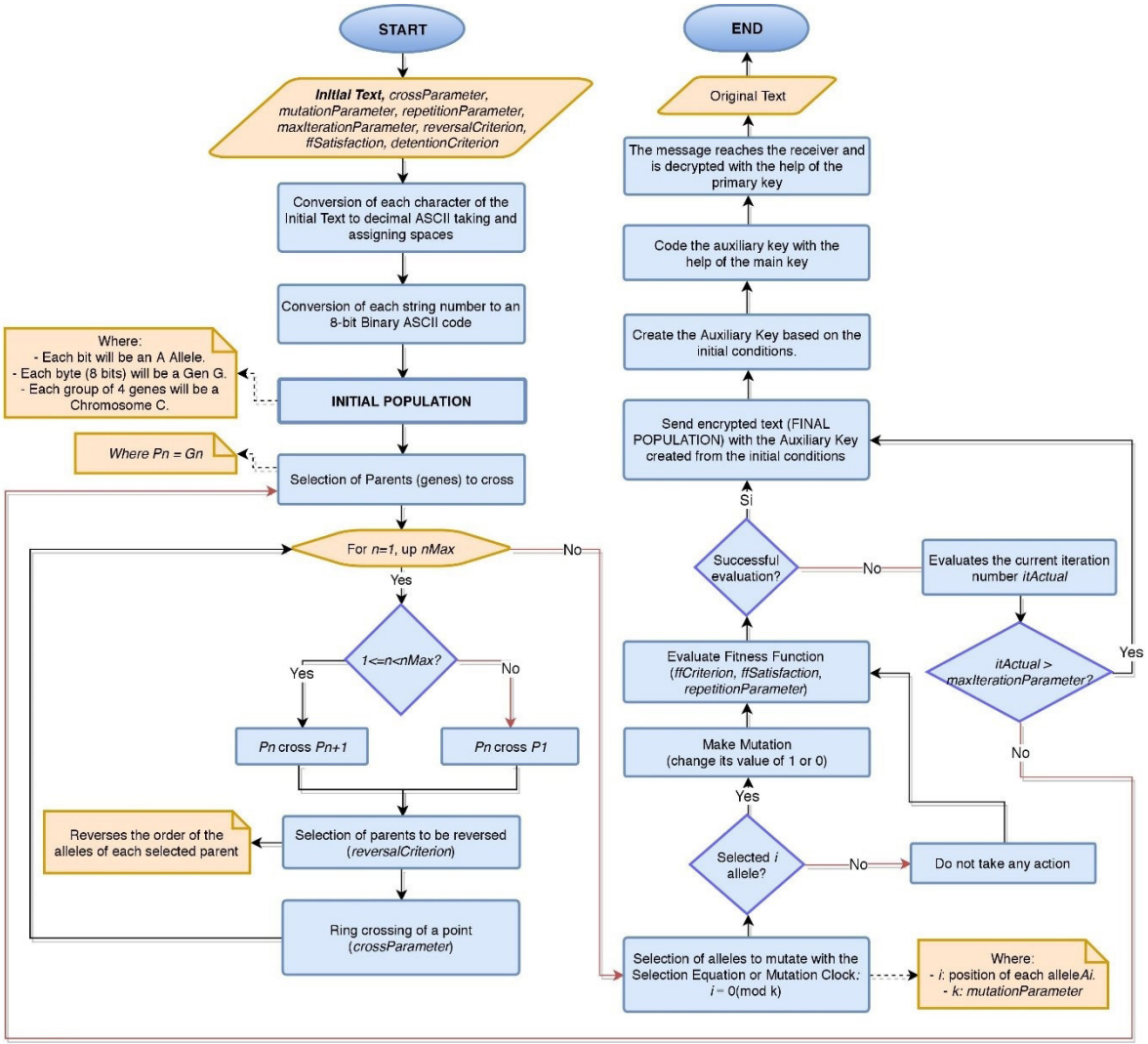



Fig. 4. Flow Diagram of the proposed cryptographic algorithm.

Thus, the algorithm continues with the alleles Mutation, to change their values and generating a new population. For example, the value of the mutation parameter k is taken arbitrarily equal to 5, whose process is evidenced (in bold) in the column Alleles selected by the Mutation Clock, in Table II (annexes).

The last population is taken as the Mutated Population, that at the same time is the **Population to Evaluate**. It is evaluated with the proposed Fitness Function based on Entropy. This function compares groups or allele chains (not necessarily the size of the original genes) to find their frequency within the population. Table III (annexes) shows the value that *ffCriterion* must take based on the length of the population to be evaluated.

In some cases, the number of genes is not an integer number, for these cases, the last chain that is not the size of the *ffCriterion* value is omitted and is not evaluated in the Fitness Function.

For the example case, the range of genes is from 32 to 63, since the total number of genes is 39; therefore, the population is divided in such a way that each gene has 6 alleles, given the value of ffCriterion. It is important to know the total number of alleles in the population:

$$totalAlle = 39 * 8 = 312 \ alleles \tag{4}$$

Next, *totalAlle* is divided into *ffCriterion* to know the number of genes that the Population to Evaluate will have.

$$\frac{totalAlle}{ffCriterion} = \frac{312}{6} = 52 \ genes \tag{5}$$

## C. Fitness Function Evaluation

As a result, the population to be evaluated consists of 52 genes, each one of 6 alleles, for a total of 312 alleles. Knowing these data, we proceed to perform the evaluation with the Fitness Function exposed in the pseudocode.

```
function fitness(popToEval, repetitionParameter)
  validChainsCount ← 0
  // C is a different chain's array
  // ia is an array of the different chain's positions
  // ic is an array of the first coincidence's positions
  [C ia ic] ← function FindUniques(popToEval)
  for i ← 0 to length(popToEval)
    count ← length(find(in ic all the positions of i))
    if count <= RepetitionParameter
      countValidChains ← countValidChains + 1
    End
  End
  return [length(C), countValidChains]
End
[DifferentChains, ValidChains] = fitness(popToEval, DetentionCriterion)
SatisfactionFF = (ValidChains/DifferentChains) * 100
DetentionCriterion = 95
If DetentionCriterion <= SatisfactionFF
  Return new generation
If Not
  Continue to the algorithm
```

Where *populationToEvaluate* is the current population, divided into *ffCriterion* size chains; *repetitionParameter* corresponds to a given value in the initial conditions that defines the maximum number of times that each of the chains of the Population to be Evaluated can be repeated; *unique*, *length* and *find* are functions of MATLAB R2017a; *differentChains* are all different chains within the population to be evaluated; *validChains* are strings that meet the value of *repetitionParameter*; *ffSatisfaction* is the percentage of satisfaction to the fitness function given by the *validChains* divided by *differentChains*; *detentionCriterion* is the value of the percentage that is needed to deliver the optimal population (See section Tests and Results, section criteria).

To elucidate the fitness function procedure, we have the following data as input:

$$\boldsymbol{populationToEvaluate} = [0001, 0010, 0001, 0100, 0000, 0010, 0010]$$
$$\boldsymbol{repetitionParameter} = 2$$
$$\boldsymbol{detentionCriterion} = 80\%$$

And as output data:

$$\boldsymbol{differentChains} = [0001, 0010, 0100, 0000]$$
$$\boldsymbol{Quantity\ different\ chains} = 4$$
$$\boldsymbol{validChains} = [0001, 0100, 0000]$$
$$\boldsymbol{Quantity\ valid\ chains} = 3$$
$$\boldsymbol{ffSatisfaction} = \frac{3}{4} * 100\% = 75\%$$

Given the previous case, we obtain a percentage of satisfaction to the Fitness Function of 75%, that represents the total of different chains in the population to be evaluated, 75% is repeated 2 times or less as it is restricted by *repetitionParameter*. The value of *detentionCriterion* established in 80% is greater than *ffSatisfaction*, for this reason, the population does not pass the assessment and the algorithm continues with a new iteration. When the percentage of *ffSatisfaction* is greater than the *detentionCriterion*, the iteration will be considered optimal and the final population is delivered in a large gene chain of 8 alleles, as it was originally stated.

Further, the variable *maxIterationParameter* is defined in the initial conditions, which limits the maximum number of times the algorithm repeats its entire process. If the number of iterations becomes equal to said parameter and an optimal population is not found, the algorithm delivers the population with the best performance.

## D. Decryption Process

The decryption of the cypher text is considered as the inverse process of the encryption using the key, in this way the initial text is delivered in a secure way to the receiver. Fig. 5 exposes the diagram of this process.

We propose a function that computes the approximate number of total alleles *at* as a function of the characters *c* of the initial text:

$$at \approx 8 * (4c - 1) \tag{6}$$

# V. Keys Generation

The Keys of a cryptographic process are pieces of information whose main objective is to allow the encryption and decryption of data. For the present proposal we have an auxiliary key and a main one.

## A. Main Key

It is previously created by the receiver and used by the algorithm (like private key). It consists of 16 characters included in ASCII code (which has 95 printable characters) arranged in a table. The selection of this key depends on the receiver, who creates it under his own criteria,

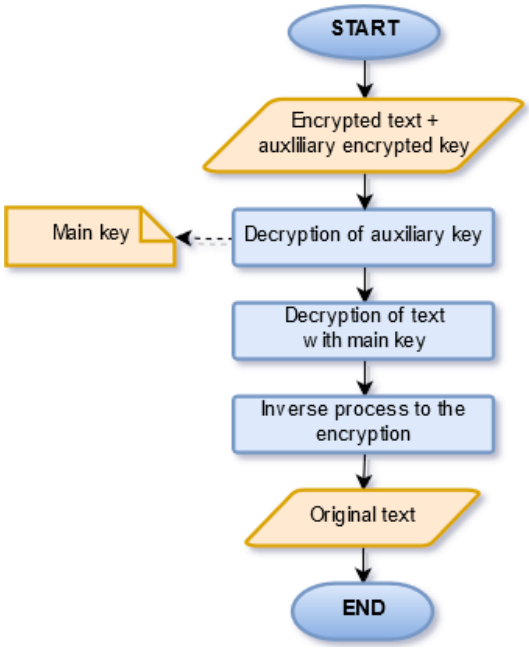



Fig. 5. Flow Diagram of the Decryption process.

always fulfilling that **there cannot be repeated characters**

$$mainKeyExamp = \boldsymbol{Tr}3\boldsymbol{V}QW90/\boldsymbol{am2}PLu* \quad (7)$$

For greater understanding, observe Table IV (annexes), which shows the correct order to locate the key.

### B. Auxiliary Key

It is generated in a strict order that only the algorithm knows, it is created as follows:

*auxKey=*

*[crossParameter,mutationParameter,*

*repetitionParameter,maxIterationParameter,*

*reversalCriterion,ffSatisfaction,*

*detentionCriterion,optimalIteration*

As an example, we have:

$$auxKeyExamp = [6, 5, 3, 200, 3, 6, 95, 2] \quad (8)$$

Based on the above, we proceed to perform the conversion of each number to ASCII (taken from '0' ASCII code 48, to '9' ASCII code 57) and then to binary. As a result, the Auxiliary Key is obtained together with the information on the length of each of its elements (highlighted in bold).

$auxKeyExamp =$ [00110101, 00110100, 00110101, 00110011,
00110101, 00110001, 00110101, 00110000,
00110100, 00111000, 00110100, 00111000,
00110101, 00110001, 00110101, 00110100,
00110101, 00110111, 00110101, 00110110,
00110101, 00110000, **00110001**, **00110001**,
**00110001**, **00110011**, **00110001**, **00110001**,
**00110010**, **00110001**] (9)

### C. Auxiliary Key Encryption

The generated Auxiliary key is encrypted with the help of the Main Key, as the first measure the gene is separated into two parts of four alleles each one; the first sub-chain is taken and the first two alleles of it are allocated in the values of the first column of the Table, subsequently the next two alleles are allocated in the first row of the table and the sub-chain is assigned the letter corresponding to the matching coordinate. For the first gene we have:

$$gene1 = \mathbf{0011}0101 \rightarrow gene1_1 = \mathbf{00\ 11}, gene1_2 = 01\ 01$$

The character corresponding to $gene1 = 0011$ is *V*, since it is the point of intersection between row 00 and column 11. In the same way, the process is carried out with the other genes that are part of the main key, obtaining for each gene a pair of characters. The next step is to convert each character to ASCII and then to binary code. Table V (annexes) shows the information corresponding to the first two genes. As a result, we obtain the *encryptedAuxKey* that consists of 180 genes, that is, 1440 alleles.

Each digit in the criteria and parameters represent 2 characters in the auxiliary key, which are taken from the main key and then converted into 8 numbers in ASCII code, or 12 genes in the encrypted auxiliary key, that is, 96 alleles; In addition, the values representing the total number of digits of each condition in the initial auxiliary key always add 384 alleles in the encrypted auxiliary key. The minimum number of possible initial digits is 9 (*crossParameter* 1 digit, *mutationParameter* 1 digit, *repetitionParameter* 1 digit, *maxIterationParameter* 1 digit, *reversalCriterion* 1 digit, *ffSatisfaction* 1 digit, *detentionCriterion* 2 digits, *optimalIteration* 1 digit), that is, 864 alleles in the encrypted auxiliary key, which means that the key will be at least 864 + 384 = 1248 alleles. Given this value, the probability of being guessed is:

$$\frac{1}{2^{1248}} \approx 2{,}06331 * 10^{-376}\% \quad (10)$$

The purpose of encrypting the auxiliary key is to increase security, because if it becomes a victim of a *Brute Force* computer attack, the encrypted string but not the original key could be discovered. This key goes from being a chain composed of 30 genes to have 180 after the encryption process, which implies a significant advance for the algorithm in terms of the security, since its length is considerably increased.

### D. About the Keys

The amount of information in bits that each key carries, is based on the possible combinations that the same one can have (value that differs from its representation in bits). Each parameter and criterion takes a value or another depending on its own restrictions (if it has any). The parameter value of crossParameter ranges from 1 to 7, *mutationParameter* between 1 and 999, *repetitionParameter* between 1 and 99, *maxIterationParameter* between 1 and 999, *reversalCriterion* between 1 and 999, *ffSatisfaction* between 1 and 99, *detentionCriterion* between 72 and 98, optimalIteration between 1 and 999. According to the above, the length of the auxiliary key is determined as follows -based on equation (1), where *k* would be the number of total combinations-:

$$log_2\,(7 * 999 * 99 * 999 * 999 * 99 * 27 * 999) \approx 60.67\ bits \quad (11)$$

Furthermore, given that the possibilities of each of the 16 characters in the **main key** is 95 (characters printable in ASCII code) its information in bits is determined as follows:

$$log_2\,(95^{16}) \approx 105.11\ bits \quad (12)$$

## VI. Quality Analysis

For the quality analysis we have the data generated of the algorithm developed, the tests and discussion about it, a comparative table of speed between DES and RSA algorithms, and finally a comparison of different factors between the AES and DES algorithms.

### A. Test and Results

The tests below are carried out based on the development of the

proposed algorithm, it allows to conclude on the results thrown by the same. Variations are made in some criteria (treated under the tests, however, are established under the user's criterion) to optimize the encryption process aimed at security and performance, the treatment of restricted parameters within their ranges is also performed.

***crossParameter***: This parameter is used by the algorithm in the crossing stage, it defines the start point to select the offspring between two genes, the maximum value that can take must not exceed the length of a gene. Since there is always an initial population composed of genes of 8 alleles (length 8), this variable must be contained within a range between 1 and 7.

***mutationParameter***: The mutation parameter defines the alleles to be selected in order to modify them, allowing the algorithm to diversify the population. It is suggested to set this variable to an odd value, because the genes are of length 8 (even number). The number of mutated alleles varies according to the value of the parameter, if this is greater than the size of the population, the algorithm does not make any mutation.

***repetitionParameter***: This variable defines the maximum number of times that each of the chains of the population to be evaluated can be repeated. Its value is established according to the performance of the algorithm, **note that if the text size is long, more repeataded chains are found**.

***reversalCriterion***: It allows to invert the digits of the second father of each crossing (or iteration), its value is selected based on the results thrown by the tests itself. For these, a value is taken between 1 to 6 and its performance is evaluated in 3 different cases according to the Fitness Function. It is iterated 4 times and the average satisfaction percentage is found. Table VI (annexes) shows the values that each initial condition takes for three different cases. Table VII (annexes) shows the results obtained by the tests carried out on the algorithm for each case.

It is observed that for Case 1 the best result (97%) is obtained in the values of *reversalCriterion* of 1, 2 and 4, implying a great homogeneity and allowing to conclude that if the initial text is short (until 4 words or 25 characters), the value of the criterion can vary without affecting the efficiency of the algorithm. For Cases 2 (90%) and 3 (97%) the best performance is achieved in the value of 3, it is suggested that this should be taken as *reversalCriterion* when the string is greater than 4 words.

***detentionCriterion***: The detention criterion establishes the percentage of different chains that comply with the *repetitionParameter*, it is used in the fitness function to evaluate if an iteration passes the evaluation. Based on the tests carried out for the *reversalCriterion*, it is suggested to establish the value of this variable in a range between 72 and 98, since this was the minimum and the maximum percentage obtained, respectively.

### *B. Runtime and Call Functions*

Some compilation data of the algorithm in MATLAB are presented in Table VIII (annexes), where the number of calls, the time spent by each of the present functions and the total value are described. For these tests, the cases described in Table VI are taken.

Runtime depends directly on the iterations made by the algorithm to arrive at the optimal solution; for the tests, it was established a *detentionCriterion* of 98, it carried out 2, 52 and 10 iterations for Case 1, 2 and 3 respectively. It is also observed that the function with the highest number of calls is *Convert* for all cases, this function aims the conversion of ASCII code to binary code and is used to transform the initial text into chains that the algorithm manipulates to perform the encryption, besides it is used in the process of creating keys.

It is also noted that the function that covers the longest runtime (a little more than the half) is A*lgorithmEncryption*, since it complies the task of carrying out the encryption process, starting from the initial population previously converted and getting a final population, ready for the evaluation.

## VII. Comparison

In order to establish objective conclusions about the performance of the proposed cryptographic algorithm, a comparison is made (at encryption execution time) against DES and RSA [22], allowing a comparison to be made as shown in Table IX (annexes). The value of *Total Time Algorithm Proposed* takes into account the time of the evaluation of the Fitness Function, additionaly to the creation and encryption of the keys, but not the time of decryption.

The text size for the proposed algorithm is taken as the bit representation of the original text, that is, the so-called *Initial Population*. The values of *detentionCriterion* and *reversalCriterion* were established in 97 and 3 respectively for the tests, while the values of *crossParameter*, *mutationParameter*, *repetitionParameter* and *maxIterationParameter*, respectively: 6, 5, 3 and 200 were set for text sizes of 128 and 256 bits; 5, 7, 5 and 100 for the text sizes of 512, 1000, 2000 and 5000 bits and 5, 17, 10 and 100 for the text size of 10000.

The key length (understood as the amount of information it carries, but not the size of its representation in bits) is 56 bits for DES and 22 bits for RSA. For the proposed Algorithm it is 105.11 bits for the main key and 60.67 bits for the auxiliary key.

It is remarkable the performance against the DES and RSA algorithms, the superiority in runtime against RSA is shown in Table IX. On the other hand, the Proposed Algorithm is superior to DES in execution time, for texts greater than 512 bits. In addition, it is observed that each of the total times of the proposed algorithm, mostly exceed that of the RSA encryption (except for 128 bits) and are very close to the DES encryption time.

In Table X (annexes), different features of the AES, RSA and DES algorithms [2] are exposed, to compare against the proposed Algorithm.

## VIII. Conclusion

The proposed algorithm modifies the order and process of the phases of the genetic algorithms, by applying a deterministic system, leaving aside some random procedures.

When comparing the proposed algorithm against RSA and DES, satisfactory performance is evidenced in several factors, demonstrating that Genetic Algorithms are a good alternative to face problems in computer security.

The proposed algorithm manages to disrupt the information through entropy, evidenced in the fitness function as the different chains.

The length (amount of information that it transports) of the auxiliary key is of 60.67 bits and of the main key is of 105.11 bits, overcoming in this aspect the cryptographic algorithm DES and approaching considerably to AES.

The present work exposes a development based on basic concepts like GA, entropy, modular congruence and determinism, that together make an efficient cryptographic process.

## IX. Considerations

The code of the algorithm developed is in a private *github* repository, with the option of being visible for those who request access to any of the contact emails. In addition there is a demo in *heroku* that performs the whole process of encryption of a given text: https://symmetric-cryptography-genetic.herokuapp.com/

It is expected to be able to use the principles of the proposed

algorithm in the encryption of images and audio. On the other hand, there is a possible application in data compression.

## Annexes

TABLE I. Conversion to ASCII and Binary Code. Crossing of the Initial Population (First Iteration)

| Initial Message | Conversion to ASCII | Conversion to Binary (Initial Population) | Position of each parent or Gene | Crossed Parents | Offspring (First Generation) |
|---|---|---|---|---|---|
| H | 0 | **00110000** | 1 | 1 y 2 | 00001101 |
| | 7 | 00110111 | 2 | 2 y 3 | 11001100 |
| | 2 | 00110010 | 3 | 3 y **4** | 10000001 |
| | (space) | **00100000** | 4 | 4 y 5 | 00001100 |
| o | 1 | 00110001 | 5 | 5 y 6 | 01001100 |
| | 1 | 00110001 | 6 | 6 y **7** | 01100011 |
| | 1 | **00110001** | 7 | 7 y 8 | 01001000 |
| | (space) | 00100000 | 8 | 8 y 9 | 00001100 |
| l | 1 | 00110001 | 9 | 9 y **10** | 01000011 |
| | 0 | **00110000** | 10 | 10 y 11 | 00001110 |
| | 8 | 00111000 | 11 | 11 y 12 | 00001000 |
| | (space) | 00100000 | 12 | 12 y **13** | 00000011 |
| a | 0 | **00110000** | 13 | 13 y 14 | 00001110 |
| | 9 | 00111001 | 14 | 14 y 15 | 01001101 |
| | 7 | 00110111 | 15 | 15 y **16** | 11000001 |
| | (space) | **00100000** | 16 | 16 y 17 | 00001100 |
| (space) | 0 | 00110000 | 17 | 17 y 18 | 00001100 |
| | 3 | 00110011 | 18 | 18 y **19** | 11010011 |
| | 2 | **00110010** | 19 | 19 y 20 | 10001000 |
| | (space) | 00100000 | 20 | 20 y 21 | 00001100 |
| m | 1 | 00110001 | 21 | 21 y **22** | 01000011 |
| | 0 | **00110000** | 22 | 22 y 23 | 00001110 |
| | 9 | 00111001 | 23 | 23 y 24 | 01001000 |
| | (space) | 00100000 | 24 | 24 y **25** | 00100011 |
| u | 1 | **00110001** | 25 | 25 y 26 | 01001100 |
| | 1 | 00110001 | 26 | 26 y 27 | 01001101 |
| | 7 | 00110111 | 27 | 27 y **28** | 11000001 |
| | (space) | **00100000** | 28 | 28 y 29 | 00001100 |
| n | 1 | 00110001 | 29 | 29 y 30 | 01001100 |
| | 1 | 00110001 | 30 | 30 y **31** | 01000011 |
| | 0 | **00110000** | 31 | 31 y 32 | 00001000 |
| | (space) | 00100000 | 32 | 32 y 33 | 00001100 |
| d | 1 | 00110001 | 33 | 33 y **34** | 01000011 |
| | 0 | **00110000** | 34 | 34 y 35 | 00001100 |
| | 0 | 00110000 | 35 | 35 y 36 | 00001000 |
| | (space) | 00100000 | 36 | 36 y **37** | 00100011 |
| o | 1 | **00110001** | 37 | 37 y 38 | 01001100 |
| | 1 | 00110001 | 38 | 38 y 39 | 01001100 |
| | 1 | 00110001 | 39 | 39 y **1** | 01000011 |

Source: Authors.

TABLE II. Population's Selection and Mutation

| Selected alleles by the mutation clock | Mutated Population |
|---|---|
| 0000**1**101 1**1**0011**0**0 100**0**0001 **0**0001**1**00 01**0**0110**0** 0110**0**011 0**1**001**0**0<br>000**0**1100 **0**1000**0**11 00**0**0111**0** 0000**1**000 0**0**00001**1**<br>000**0**1110 **0**1001**1**01 11**0**0000**1** 0000**1**100 0**0**0011**0**0 110**1**0011 **1**0001**0**00<br>00**0**01100 0100**0**011 0**0**0011**1**0 0100**1**000 **0**0100**1**11<br>01**0**0110**0** 0100**1**101 1**1**00000**1** 000**0**1100 **0**1001**1**00 01**0**0001**1** 00001000<br>0**0**0011**0**0 010**0**0011 **0**0001**1**00 00**0**0100**0** 0010**0**011<br>0**1**0011**0**0 010**0**1100 **0**1000**0**11 | 00000101 1**1**001**1**10 100**1**0001 **1**0001**0**00 01**1**01**1**01 0110**1**011 00**0**01**1**0<br>0001**1**100 **1**1000**1**11 00**1**01**1**1 00000000 0**1**000**0**01<br>0001**1**110 **1**1001**0**01 11**1**00000 00000100 0**1**001**1**10 11**0**00011 **0**0001**1**00<br>00**1**011**0**1 0100**1**011 0**1**001**1**00 0101**1**000 **1**0100**1**11<br>01**1**01101 0100**0**101 1**0**00001**1** 000**1**1100 **1**1001**0**00 01**1**0001**0** 00000000<br>0**1**001**1**10 010**1**0011 **1**0001**0**00 00**1**01001 00101011<br>00**0**01**1**10 0101**1**100 **1**1000**1**11 |

Source: Authors.

TABLE III. *ffCriterion* Value According to the Range Where the Population Size Is Located

| Genes range of numbers in the population | Value *n* | Value according to the range | Alleles chains length to compare (ffCriterion) |
|---|---|---|---|
| 4 - 7 | 2 | $2^2 = 4$ | 3 |
| 8 - 15 | 3 | $2^3 = 8$ | 4 |
| 16 - 31 | 4 | $2^4 = 16$ | 5 |
| 32 - 63 | 5 | $2^5 = 32$ | 6 |
| $2^n$ - $2^{n+1}-1$ | $n$ | $2^n$ | $n+1$ |

Source: Authors.

TABLE IV. Main Key Distribution

| | **00** | **01** | **10** | **11** |
|---|---|---|---|---|
| **00** | **T** | **r** | **3** | **V** |
| **01** | Q | W | 9 | 0 |
| **10** | / | **a** | **m** | **2** |
| **11** | P | L | u | * |

Source: Authors.

TABLE V. Corresponding Characters for the First Two Genes of Auxiliary Key

| *Gene* | $Gene_n$ | Substring value | Corresponding character (according to table 4) | Conversion to ASCII | Conversion to binary (*encryptedAuxKey*) |
|---|---|---|---|---|---|
| *gene1* | $gene1_1$ | 0011 | V | 0 | 00110000 |
| | | | | 8 | 00111000 |
| | | | | 6 | 00110110 |
| | $gene1_2$ | 0101 | W | 0 | 00110000 |
| | | | | 8 | 00111000 |
| | | | | 7 | 00110111 |
| *gene2* | $gene1_3$ | 0011 | V | 0 | 00110000 |
| | | | | 8 | 00111000 |
| | | | | 6 | 00110110 |
| | $gene1_4$ | 0100 | Q | 0 | 00110000 |
| | | | | 8 | 00111000 |
| | | | | 1 | 00110001 |

Source: Authors.

TABLE VI. Initial Conditions Value for the Three Tests Cases

| Initial condition | Case 1 | Case 2 | Case 3 |
|---|---|---|---|
| *Initial text* | Hola mundo | Esta es una cadena mas larga que la anterior | See **Text 3** |
| *Initial population (bits representation)* | 39 | 175 | 2051 |
| *Population to Evaluate* | 52 | 175 | 1367 |
| *crossParameter* | 6 | 5 | 5 |
| *mutationParameter* | 5 | 5 | 17 |
| *repetitionParameter* | 3 | 4 | 7 |
| *maxIterationParameter* | 200 | 200 | 200 |

Source: Authors.

TABLE VII. TESTS ON *REVERSALCRITERION* FOR CASE 1, CASE 2 AND CASE 3

| *reversalCriterion* | **Different strings** | | | **Valid strings** | | | **Satisfaction percentage** | | | **Average Satisfaction percentage** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Case** | **1** | **2** | **3** | **1** | **2** | **3** | **1** | **2** | **3** | **1** | **2** | **3** |
| 1 | 36 | 68 | 552 | 36 | 59 | 516 | 100% | 87% | 93% | **98%** | 83% | 94% |
| | 33 | 57 | 569 | 32 | 45 | 536 | 97% | 79% | 94% | | | |
| | 32 | 56 | 547 | 32 | 44 | 511 | 100% | 79% | 93% | | | |
| | 34 | 65 | 546 | 32 | 58 | 513 | 94% | 89% | 94% | | | |
| 2 | 36 | 72 | 606 | 36 | 63 | 578 | 100% | 88% | 95% | **98%** | 81% | 95% |
| | 35 | 63 | 576 | 34 | 48 | 550 | 97% | 76% | 95% | | | |
| | 33 | 67 | 607 | 32 | 57 | 575 | 97% | 85% | 95% | | | |
| | 35 | 58 | 583 | 34 | 43 | 555 | 97% | 74% | 95% | | | |
| 3 | 32 | 78 | 659 | 28 | 68 | 635 | 88% | 87% | 96% | 93% | **90%** | **97%** |
| | 33 | 70 | 704 | 33 | 59 | 684 | 100% | 84% | 97% | | | |
| | 36 | 84 | 645 | 33 | 77 | 625 | 92% | 92% | 97% | | | |
| | 34 | 85 | 628 | 32 | 81 | 602 | 94% | 95% | 96% | | | |
| 4 | 37 | 54 | 528 | 36 | 52 | 497 | 97% | 96% | 94% | **98%** | 82% | 95% |
| | 35 | 49 | 590 | 34 | 33 | 567 | 97% | 67% | 96% | | | |
| | 33 | 62 | 579 | 33 | 50 | 548 | 100% | 81% | 95% | | | |
| | 27 | 65 | 596 | 26 | 55 | 564 | 96% | 85% | 95% | | | |
| 5 | 29 | 50 | 606 | 26 | 39 | 585 | 90% | 78% | 97% | 91% | 72% | 96% |
| | 33 | 42 | 633 | 31 | 27 | 606 | 94% | 64% | 96% | | | |
| | 30 | 47 | 641 | 28 | 34 | 619 | 93% | 72% | 97% | | | |
| | 28 | 47 | 636 | 24 | 35 | 616 | 86% | 74% | 97% | | | |
| 6 | 34 | 64 | 622 | 32 | 53 | 594 | 94% | 83% | 95% | 96% | 83% | 95% |
| | 31 | 63 | 631 | 31 | 50 | 601 | 100% | 79% | 95% | | | |
| | 31 | 68 | 641 | 30 | 55 | 611 | 97% | 81% | 95% | | | |
| | 31 | 72 | 624 | 29 | 65 | 596 | 94% | 90% | 96% | | | |

Source: Authors.

TABLE VIII. ALGORITHM PERFORMANCE FOR CASE 1, CASE 2 AND CASE 3

| **Function's name** | **Function calls** | | | **Time (seconds)** | | |
|---|---|---|---|---|---|---|
| | **Case 1** | **Case 2** | **Case 3** | **Case 1** | **Case 2** | **Case 3** |
| *AlgorithmEncryption* | 1 | 1 | 1 | 0,027 | 0,133 | 0,261 |
| *AuxiliaryKeyEncrypt* | 60 | 64 | 72 | 0,011 | 0,012 | 0,012 |
| *FitnessFunction* | 2 | 52 | 10 | 0,002 | 0,011 | 0,028 |
| *Unique* | 2 | 52 | 10 | 0,008 | 0,023 | 0,018 |
| *Convert* | 249 | 399 | 2299 | 0,004 | 0,004 | 0,008 |
| *Strlength* | 12 | 13 | 15 | 0,001 | 0,001 | 0,001 |
| *NewDivisionPopulationToEvaluate* | 2 | 52 | 10 | 0,001 | 0,029 | 0,045 |
| **TOTAL** | **328** | **633** | **2417** | **0,055** | **0,215** | **0,374** |

Source: Authors, MATLAB R2017a.

TABLE IX. EXECUTION TIME IN SECONDS FOR THE ENCRYPTION DES, RSA AND PROPOSED ALGORITHM

| **Text size (bits)** | **DES Encrypt** | **RSA Encrypt** | **Proposed algorithm Encrypt** | **Total time of the Proposed algorithm** |
|---|---|---|---|---|
| 128 | 0.054945 | 0.0549 | 0.055 | 0.100 |
| 256 | 0.054946 | 0.1098 | 0.058 | 0.100 |
| 512 | 0.070976 | 0.2197 | 0.083 | 0.137 |
| 1000 | 0.1418 | 0.3846 | 0.125 | 0.187 |
| 2000 | 0.2835 | 0.7142 | 0.157 | 0.228 |
| 5000 | 0.6816 | 1.7032 | 0.479 | 0.704 |
| 10000 | 1.3601 | 3.402 | 1.000 | 1.441 |

Source: [22] and Authors.

TABLE X. Comparison Between AES, DES, RSA and Proposed Algorithm

| Factors | AES | DES | RSA | Proposed algorithm |
|---|---|---|---|---|
| **Development year** | 2000 | 1977 | 1978 | 2017 |
| **Key's length** | 128, 192, 256 bits | 56 bits | >1024 bits | 105.11 bits |
| **Key for encrypt and decrypt** | Same password | Same password | Different password | Same passwords |
| **Block size** | 128 bits | 64 bits | least 512 bits | 8 bits |
| **Scalability** | Not scalable | Scalable (It depends on the block size and the password) | Not scalable | Scalable (It depends on the initial text's size) |
| **Algorithm's kind** | Symmetric | Symmetric | Asymmetric | Symmetric |
| **Execution time** | Fast | Moderate | slow | Fast |
| **Key's tank** | Necessary | Necessary | Necessary | Necessary |
| **Inherent vulnerability** | Force attack | Force attack, linear and differential cryptanalysis attack | Force attack and "Oracle attack" | Force attack, linear and differential cryptanalysis attack |
| **Rounds** | 10/12/14 | 16 | 1 | Depends on the maximum iteration parameter value or the optimal iteration number |

Source: [3] and Authors.

## References


[1] Carle, G. (2003). *Network Security Chapter 7: Cryptographic Protocols*. In: http://www.ccs-labs.org/~dressler/teaching/netzsicherheit-ws0304/07_CryptoProtocols_2on1.pdf.

[2] Abd, D., Abdual, H. and Hadhoud, M. (2010). *Evaluating the Performance of Symmetric Encryption Algorithms.* International Journal of Network Security, Vol. 10, No. 3, pp. 213–219.

[3] Mahajan, P. and Sachdeva, A. (2013). A Study of Encryption Algorithms AES, DES, and RSA for Security. *Global Journal of Computer Science and Technology*, vol. 13. In: https://computerresearch.org/index.php/computer/article/view/272/272.

[4] Beegom, B. y Jose, S. (2017). "*An Enhanced Cryptographic Model Based on DNA Approach*". Presented at: *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, Idukki, India. Doi: 10.1109/ICECA.2017.8212824.

[5] Srilatha, N. and Murali, G. (2016). "*Fast three level DNA Cryptographic technique to provide better security*". Presented at: *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, Bangalore. Doi: 10.1109/ICATCCT.2016.7912037.

[6] Lawrence, E., (2006). *HTTPS Security Improvements in Internet Explorer 7*. In: https://msdn.microsoft.com/en-us/library/bb250503(v=vs.85).aspx.

[7] Dierks, T. and Rescorla E., (2008). *The Transport Layer Security (TLS) Protocol, Version 1.2.* In: https://tools.ietf.org/html/rfc5246.

[8] Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. London, England: MIT Press, pp. 6-16.

[9] Goldberg, D. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Boston, USA: Addison-Wesley, pp. 1-68.

[10] Hermawanto, D. (2013). Genetic Algorithm for Solving Simple Mathematical Equality Problem. *e-print arXiv:* 1308.4675.

[11] Syswerda, G. (1991). *Schedule optimization using genetic algorithms*. New York, USA: Lawrence Davis, pp. 322-349.

[12] Jhajharia, S., Mishra, S., and Bali, S. (2013). "Public key cryptography using neural networks and genetic algorithms". Presented at: 2013 Sixth International Conference on Contemporary Computing (IC3), Noida, India. Doi: https://doi.org/10.1109/IC3.2013.6612177

[13] Simmons, G.J. (1988). A survey of Information Authentication. *IEEE Proceedings*, vol. 76, pp. 603-620, doi: https://doi.org/10.1109/5.4445

[14] Bhowmik, S. and Acharyya, S. (2011, June). "*Image Cryptography: The Genetic Algorithm Approach*". Presented at: *International Conference on Computer Science and Automation Engineering*, Shanghai, China. Doi: 10.1109/CSAE.2011.5952458

[15] Conci, A., Brazil, A.L., Leal, S.B. and MacHenr, T. (2015, November). "*AES Cryptography in Color Image Steganography by Genetic Algorithms*". Presented at: *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, Marrakech, Morocco. Doi: https://doi.org/10.1109/AICCSA.2015.7507100

[16] Al-Tabtabai, H., and Alex, A.P. (1999). Using Genetic Algorithms to solve optimization problems in construction. *Engineering, Construction and Architectural Management*, vol. 6, Issue: 2, pp. 121-132, doi: https://doi.org/10.1108/eb021105

[17] Traberg, G., Moliari, L., Venosa, P. and Macia, N. (2015). "*Automatizando el descubrimiento de portales de autenticación y evaluación de la seguridad mediante ataques de fuerza bruta en el marco de una auditoría de seguridad".* Presented at: *XXI Congreso Argentino de Ciencias de la Computación,* Argentina. In: http://sedici.unlp.edu.ar/handle/10915/50589

[18] Shannon, C.E. (1949). Communication Theory of Secrecy Systems. *The Bell System Technical Journal*, vol. 28, Issue: 4, doi: http://dx.doi.org/10.1002/j.1538-7305.1949.tb00928.x

[19] Othman, H., Hassoun, Y. and Owayjan, M. (2015, October). "*Entropy Model for Symmetric Key Cryptography Algorithms Based on Numerical Methods*". Presented at: *2015 International Conference on Applied Research in Computer Science and Engineering (ICAR)*, Beirut, Lebanon. Doi: 10.1109/ARCSE.2015.7338142

[20] Gauss, C.F. (1801) and Clarke, A.A. (1965). *Disquisitiones Arithmeticae Translated.* New Haven, Connecticut, USA, pp. 1-4.

[21] Jimenez, R., Gordillo, E. and Rubiano, G. (2012). *Teoría de Números (para principiantes)*. Bogotá, Colombia: Pro–Offset Editorial Ltda., pp. 98-104.

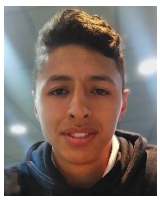
[22] Narasimham, C. and Jayaram, P. (2008). *Evaluation of performance characteristics of Cryptosystem using text files.* Journal of Theoretical and Applied Information Technology, pp. 55-59.

[23] Zaldaña, HR. and Castañeda E. (2015). *The Use of Genetic Algorithms in UV Disinfection of Drinking Water*. International Journal of Interactive Multimedia and Artificial Intelligence, vol. 3, pp. 43-48.



Jefferson Rodríguez Rodríguez

Jefferson Rodríguez Rodríguez was born in Bogotá, Colombia in 1994. Aspiring to Computer Engineer in Universidad Distrital Francisco José de Caldas. He works in Exsis SAS since 2017 as a systems analyst. His area of interest includes AI, computer security and backend development.


Brayan Julián Corredor

Brayan Julián Corredor was born in Bogotá, Colombia in 1994. Computer Engineer from Universidad Distrital Francisco José de Caldas since 2017. He works in Stefanini IT since 2017 as an Engineer jr. His area of interest includes AI, computer security and frontend development.

César Suárez Parra

César Suárez Parra was born in Bogotá, Colombia in 1956. He got his industrial engineering degree from Fundacion Universidad Incca de Colombia in 1980, and his mechanical engineer degree from Fundacion Universidad Incca de Colombia in 1992. He received his specialist degree in applied mathematics from Universidad Sergio Arboleda in 2003 and his master degree in materials and manufacturing processes from Universidad Nacional de Colombia in 1998. He is a teacher in Universidad Distrital Francisco José de Caldas, dedicated to higher education in engineering and investigations groups. He has a strong research record in cryptography, complexity and genetic algorithms.