Reversed Genetic Algorithms for Generation of Bijective S-boxes with Good Cryptographic Properties

Georgi Ivanov[1], Nikolay Nikolov[2], and Svetla Nikova[3]

[1] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
[2] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
[3] KU Leuven, Dept. ESAT/COSIC and iMinds, Belgium

Abstract. Often S-boxes are the only nonlinear component in a block cipher and as such play an important role in ensuring its resistance to cryptanalysis. Cryptographic properties and constructions of S-boxes have been studied for many years. The most common techniques for constructing S-boxes are: algebraic constructions, pseudo-random generation and a variety of heuristic approaches. Among the latter are the genetic algorithms. In this paper, a genetic algorithm working in a reversed way is proposed. Using the algorithm we can rapidly and repeatedly generate a large number of strong bijective S-boxes of each dimension from (8×8) to (16×16), which have sub-optimal properties close to the ones of S-boxes based on finite field inversion, but have more complex algebraic structure and possess no linear redundancy.

Keywords: Genetic Algorithms, S-boxes, Nonlinearity

1 Introduction

Most of the modern block ciphers have one or more non-linear components providing the effect of confusion [31], which is of vital importance for the strength of the cipher. The most often used non-linear components are n to m Boolean mappings, called S-boxes. Among them, the bijective S-boxes are particularly interesting. Block ciphers have to be resistant to linear and differential cryptanalysis [3,4,17]. It is well known that the weight of a differential trail is equal to or larger than the product of the number of active S-boxes and the minimum (differential) weight per S-box (analogous for the linear trails). There are two known approaches for eliminating the low-weight trails - either by increasing the number of active S-boxes or by choosing stronger S-boxes. The number of active S-boxes can be increased by designing stronger linear layer, e.g., by using the wide trail strategy [10]. Larger S-boxes can be stronger since the minimum differential weight of an S-box is limited by its size. In this paper we focus only on the second approach. In order to be suitable for use in cryptographic applications S-boxes should possess *high nonlinearity* and *low differential uniformity*. In addition, cryptographically strong S-boxes should also satisfy cryptographic criteria such as: *regularity* (*balancedness*) and *high algebraic degree*, as well as they should possess as few *fixed points* and *linear redundancy* as possible.

1.1 Motivation

Large S-boxes can fast improve the security (in the black box model) of a block cipher. In addition, the results in [16] indicate that the resistance to side-channel analysis could also be improved by increasing the size of the S-box. However, finding or constructing such big and cryptographically strong S-boxes is still an open problem.

The approaches for S-box generation available in the literature could be divided into three main streams: *algebraic constructions*, *pseudo-random generation* and *heuristic techniques*. The first approach is based on S-box generation according to mathematical principles. For example the S-box of AES [10] is constructed with inverse mapping followed by an affine transformation in the finite field. S-boxes based on inversion in the finite field $GF(2^n)$ are known to achieve the best values found for the *nonlinearity* and *differential uniformity* ($N_{inv} = 2^{n-1} - 2^{\frac{n}{2}}$ and $\delta_{inv} = 4$ respectively, [24]). Although these values are not the optimal values theoretically possible ($N_{opt} = 2^{n-1} - 2^{\frac{n}{2}-1}$ and $\delta_{opt} = 2$, [23]) and filling up the gap between them and their known theoretical bounds is an open problem in cryptography, it is believed that the finite field inversion-based S-boxes are the optimal ones that exist with respect to the simultaneous satisfaction of all targeted cryptographic criteria. A new method for constructing 4-*uniform* permutations has recently been proposed in [26, 27].

The second approach is based on using some *pseudo-random* generation to generate S-boxes. However, to find good S-boxes quickly becomes infeasible as the size of the input space increases. Also, the probability of finding really strong S-boxes is very small. For example, in the case of (8×8) S-boxes, the highest value for *nonlinearity* found is $98 - 100$ [20, 21], which is rather low compared to the value of 112 for the finite field inversion-based case.

The third approach uses *heuristic algorithms* in a process of iteratively improving given S-box or S-boxes with respect to one or more properties. Because these algorithms use directed search methods, unlike the algebraic constructions they are able to produce a large number of S-boxes, which are not optimal but sub-optimal. Specific heuristic techniques include the *hill climbing* method, the *simulated annealing* method, the *genetic algorithm* or a combination of these. For example the highest *nonlinearity* achieved by: the *hill climbing* method is 100 [20], the *simulated annealing* method is 102 [8], and by a special *genetic algorithm* is 104 [32]. Recently in [15], values of 104 have been achieved for the *nonlinearity* by a method, referred as the modified gradient descent, which is based on swapping a number of values in a permutation.

The construction of a large number of good bijective S-boxes quickly becomes a hard job as their size increases. As the number of input variables n increases by one, the number of Boolean functions in the space increases by a factor of 2^{2^n}. That is why, except for the *algebraic construction* method, the other two methods fail short already when $n = 8$. Therefore, it is a challenge to improve the results for size 8, and even more, to go for larger S-boxes.

1.2 Contribution

Often it is desired (e.g. for efficient implementations) to find a large set of strong bijective S-boxes possessing the same cryptographic properties and to allow the designer to freely choose the S-box within this set to be used in a cryptographic algorithm. This cannot be achieved using *algebraic constructions* and *pseudo-random generation* methods, since either it will not be possible to find many S-boxes or they will not be strong enough. What concerns the known *heuristic techniques*, the conventional approach from bottom to the top, where usually one starts with some initial set of random S-boxes and tries iteratively to improve them, is not quite suitable. The obtained in this way set of values corresponding to the set of targeting criteria is not close enough to the set of values of finite field inversion-based S-boxes. Furthermore, when one searches for a set of good S-boxes which in addition have to be big - from (8×8) to (16×16), all generation methods known are quite impractical, due to the huge time and memory resources needed for the computations. Therefore, in this paper we propose a new method based on a *genetic algorithm* which is working in a "*reversed*" way.

More precisely, the algorithm starts from an initial pool of S-boxes based on inversion in the finite field and searches for S-boxes which are close to them with respect to the targeted set of cryptographic criteria. The detailed description is provided in Section 4. Applying the new algorithm we repeatedly and rapidly obtain *a large set* of S-boxes which are:

1. of dimensions from (8×8) to (16×16);
2. have properties close to the best properties found of S-boxes based on finite field inversion;
3. have more complex algebraic structure;
4. possess little or no *linear redundancy*.

For the case of (8×8) bijective S-boxes the algorithm outputs thousands of S-boxes with *nonlinearities* 106, 108, 110 and 112 $(= N_{inv})$ after 2 days work on a cluster with 32 cores. S-boxes with nonlinearity 110 and possessing zero (i.e. no linear) redundancy as well as S-boxes with nonlinearity 112 which have incomplete redundancy are not reported till now in the literature.

For the case of (16×16) bijective S-boxes 50 S-boxes with values of *nonlinearity* $32400 = N_{inv} - 112$ are obtained ($N_{inv} = 2^{n-1} - 2^{\frac{n}{2}} = 32512$) after 3 months work on the same computer. With *nonlinearities* closer to N_{inv}, for example $N_S = N_{inv} - 36 = 32476$, 50 S-boxes are obtained for less than a month. In fact, thanks to the "*reversed*" way the algorithm works, it can output S-boxes with each possible value of *nonlinearity* up to N_{inv}. Furthermore, the closer to N_{inv} the values of the *nonlinearity* are, the faster the algorithm works. We are not aware of any constructions of S-boxes with size bigger than 8, obtained with either *pseudo-random generation* or *heuristic techniques*. Thus, it is the first time when such large S-boxes with good cryptographic properties are generated. In Table 1 we compare our results with the state of the art results for the case $n = 8$.

Table 1: A comparison between the cryptographic properties of (8×8) bijective S-boxes obtained by various generation methods (NR stands for "not reported")

Methods/Properties	N_S	$\deg(S)$	$AC(S)_{max}$	δ-uniformity	Fixed points	Linear Redundancy
Pseudo-random S-BOX [20], [21]	98	NR	NR	NR	NR	NR (usually *zero*)
Finite Field Inversion [24]	112	7	32	4	2	*complete*
Hill Climbing [20]	100	NR	NR	NR	NR	NR
GA/HC [21]	100	NR	NR	NR	NR	NR
Simulated Annealing [8]	102	NR	80	NR	NR	NR
GaT [32]	104	NR	NR	NR	NR	NR
Tweaking [12]	106	7	56	6	NR	*zero*
Gradient descent method [15]	104	7	80	8	0	NR
4-uniform permutations method [26, 27]	98	NR	NR	4	NR	NR
GA1 [this paper]	106	6	56	6	2	*zero*
GA1 [this paper]	108	6	48	6	0	*zero*
GA2 [this paper]	110	7	40	6	0	*zero*
GA2 [this paper]	112	7	32	6	0	*some* (147 of 255)

Note that *algebraic immunity* is also included in the target set of criteria for the Gradient descent method. The reported value for algebraic immunity is 3 [15]. We do not include algebraic immunity in our target set of criteria.

2 Preliminaries

In this section we will briefly recall some of the basic definitions and properties of Boolean functions. For a comprehensive survey on Boolean functions we refer to [5,6].

Let the substitution table (S-box) of an n-binary input into m-binary output mapping is denoted by S. Then $S : \mathbb{B}^n \to \mathbb{B}^m$ and to each $x = (x_1, x_2, \ldots, x_n) \in \mathbb{B}^n$ some $y = (y_1, y_2, \ldots, y_m) \in \mathbb{B}^m$ is assigned by $S(x) = y$, where $\mathbb{B} = \{0, 1\}$ is the 1-dimensional Boolean space. Clearly, S can be considered as a vectorial Boolean function consisting of m individual Boolean functions $f_1, f_2, \ldots, f_m$, where $f_i : \mathbb{B}^n \to \mathbb{B}$ and $f_i(x) = y_i$ for $i = 1, 2, \ldots, m$. These functions are called *coordinate* Boolean functions of the S-box S and it is well known that most of the desirable cryptographic properties of S can be defined in terms of their linear combinations. S-box *coordinate* Boolean functions and all their linear combinations are referred as the S-box *component* Boolean functions.

2.1 Boolean functions

A Boolean function can be represented by a *truth table*, which is the binary output vector of the function containing 2^n elements. We obtain the *polarity truth table* when instead of $f(x)$, the *signed* function $\widehat{f}(x) = (-1)^{f(x)}$ is considered. Another way of representing a Boolean function is by means of its *algebraic normal form* (ANF):

$$f(x) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \ldots \oplus a_n x_n \oplus a_{1,2} x_1 x_2 \oplus \ldots \oplus a_{1,2,\ldots,n} x_1 x_2 \ldots x_n,$$

where the coefficients $a_I \in \mathbb{B}, I \subseteq \{1, 2, \ldots, n\}$. The *algebraic degree* of an n-variable Boolean function $f(x)$, denoted by $deg(f)$, is the number of variables of the largest product term of the function's ANF having a non-zero coefficient.

Two n-variable Boolean functions $f(x)$ and $g(x)$ belong to the same *equivalence class* (or are affine equivalent) if and only if there exist some invertible $(n \times n)$ binary matrix A, vectors $b, c \in \mathbb{B}^n$ and a scalar $d \in \mathbb{B}$, such that $g(x) = f(Ax \oplus b) \oplus < c, x > \oplus d$.

The *Walsh-Hadamard transform (WHT)* of an n-variable Boolean function $\widehat{f}(x)$, denoted by $\widehat{F}_f(w)$, is defined by:

$$\widehat{F}_f(w) = \sum_{x \in \mathbb{B}^n} \widehat{f}(x)(-1)^{<w,x>} = \sum_{x \in \mathbb{B}^n} (-1)^{f(x) \oplus <w,x>} = \sum_{x \in \mathbb{B}^n} \widehat{f}(x)\widehat{l}_w(x),$$

where $\widehat{l}_w(x)$ is the *signed* function of the linear function $l_w(x) =< w, x >$.

Thus, $\forall w \in \mathbb{B}^n$, $\widehat{F}_f(w) \in [-2^n, 2^n]$. $\widehat{F}_f(w)$ is called a spectral Walsh coefficient and the real-valued vector of all 2^n Walsh coefficients is referred to as the *WHT Spectrum*. We denote the maximum absolute value taken by the WHT by $WHT_{max}(f) = \max_{(w \in \mathbb{B}^n)} |\widehat{F}_f(w)|$.

Some of the most important cryptographic properties that a Boolean function should possess are defined below. An n-variable Boolean function $f(x)$ is called *balanced* if $w_H(f) = 2^{n-1}$, where by $w_H(f)$ is denoted the Hamming weight of f. The *nonlinearity* of an n-variable Boolean function $f(x)$, denoted by N_f, is the minimum distance to the set of all n-variable affine Boolean functions $A(n)$. It is given by $N_f = [2^n - WHT_{max}(f)]/2$. The *autocorrelation transform* (ACT) of $\widehat{f}(x)$, denoted by $\widehat{r}_f(\alpha)$, taken with respect to a vector $\alpha \in \mathbb{B}^n$ is defined by:

$$\widehat{r}_f(\alpha) = \sum_{x \in \mathbb{B}^n} (-1)^{f(x) \oplus f(x \oplus \alpha)} = \sum_{x \in \mathbb{B}^n} \widehat{f}(x)\widehat{f}(x \oplus \alpha)$$

Thus, $\forall \alpha \in \mathbb{B}^n$, $\widehat{r}_f(\alpha) \in [-2^n, 2^n]$ and $\widehat{r}_f(0) = 2^n$. The $\widehat{r}_f(\alpha)$ is called *spectral autocorrelation coefficient* and the real-valued vector of all 2^n *autocorrelation coefficients* representing the ACT of the function is referred to as its ACT *Spectrum.*

The *maximum ACT value* or *absolute indicator* of an n-variable Boolean function in *polarity* form $\widehat{f}(x)$, denoted by $AC_{max}(f)$, is defined by: $AC_{max} = max_{(\alpha \in \mathbb{B}^n \setminus \{0\})} |\widehat{r}_f(\alpha)|$.

The results of Meier and Staffelbach [19] and of Preneel [25] show that some of the most important cryptographic characteristics of Boolean functions as *algebraic degree, nonlinearity* and *absolute indicator* are invariant under affine transformations, while in other, like WHT and ACT, the effect produced by applying affine transformations is permutation in the values of the spectral coefficients, and some change in their signs. Anyway, their magnitudes are not affected by affine transformations. In other words, for any two affine equivalent n-variable Boolean functions f and g, is true that $deg(g) = deg(f)$, $WHT_{max}(g) = WHT_{max}(f)$, $N_g = N_f$ and $AC_{max}(g) = AC_{max}(f)$.

2.2 Vectorial Boolean functions

The properties of Boolean functions discussed in the previous section can be extended to the case of vectorial Boolean functions (S-boxes). There are some conceptual similarities in the transition from the single-output to the multi-output case but there are also some essential differences in the manner by which the S-boxes properties are derived. It is important to note that it is not sufficient only to consider the *coordinate* Boolean function properties when considering S-box cryptographic properties but also their linear combinations.

To avoid trivial statistical attacks, a good S-box should be a *regular* (*balanced*) mapping. An $(n \times m)$ S-box S with $n \geq m$ is said to be *regular* if for each its output $y \in \mathbb{B}^m$ there are exactly 2^{n-m} inputs that are mapped to y. If the S-box is not *regular*, some outputs appear more often than others when the input to the S-box is randomly chosen and the bias can be exploited by the cryptanalyst. An $(n \times m)$ S-box with $n \geq m$ is *regular* if and only if all non-zero its *component* Boolean functions are *balanced* [30].

Linear cryptanalysis [17], is a known-plaintext attack which is trying to approximate the relationship between plaintext, ciphertext and the key bits by constructing a linear expression and then evaluating the probability P attached to this expression. Among all possible expressions, those which achieve highest/lowest probabilities to be the correct are the best linear/affine approximations. Thus, if all the probabilities P are approximately equal to $\frac{1}{2}$, the cipher will be resistant to linear and affine approximation.

An $(n \times m)$ S-box improves the immunity against *linear cryptanalysis* if in its *linear approximation table (LAT)* all the entries magnitudes are as small as possible. In [29] it is shown that the latter is equivalent to the statement that the *nonlinearity* of each non-zero S-box *component* Boolean function should be as high as possible. The *nonlinearity* of an $(n \times m)$ S-box S, denoted by N_S, is defined as the minimum *nonlinearity* of each of the *component* Boolean functions excluding the zero one. It can be expressed as:
$N_S = \min_{(c=(c_1,c_2,\ldots,c_m) \in \mathbb{B}^m \setminus \{0\})} N_{c_1 f_1 \oplus c_2 f_2 \oplus \ldots \oplus c_m f_m}$.

In order to resist *low order approximation* attacks each S-box must have an *algebraic degree* as high as possible ([14, 22]). The *(minimal) algebraic degree* of an $(n \times m)$ S-box S, denoted by $deg(S)$, is defined as the minimum *algebraic degree* of each of its non-trivial *component* Boolean functions. It can be expressed as follows:

$$deg(S) = \min_{c=(c_1,c_2,\ldots,c_m) \in \mathbb{B}^m \setminus \{0\}} deg(c_1 f_1 \oplus c_2 f_2 \oplus \ldots \oplus c_m f_m).$$

Differential cryptanalysis, introduced by Biham and Shamir [3] is applied to block ciphers as a chosen-plaintext attack, which consists in finding relationships between plaintext differences and their corresponding ciphertext differences in order to gain knowledge of the key bits. The *differential uniformity* of an $(n \times m)$ S-box S with $n \geq m$, denoted by δ, is defined as the largest value present in its *difference distribution table (DDT)* not counting the first entry in the first row. That is, $\delta = max_{\alpha \in \mathbb{B}^n \setminus \{0\}} max_{\beta \in \mathbb{B}^m} |\{x \in \mathbb{B}^n | S(x) \oplus S(x \oplus \alpha) = \beta\}|$. Then, S is said to be *differentially δ-uniform*.

Thus, a necessary condition for an S-box to improve the resistance against *differential cryptanalysis* is its DDT not to contain entries with large values (DDT to be flat) or equivalently the *differential uniformity* δ to be as small as possible [1, 7, 11]. In [34] the relation $AC(S) = DDT.H_m$ has been shown, where H_m is the $(2^m \times 2^m)$ *Sylvester-Hadamard matrix* [18] and $AC(S)$ is the $(2^n \times 2^m)$ *autocorrelation matrix* of S which columns represent the *autocorrelation* functions of all *component* Boolean functions of S. Also, in [34] a lower bound on the *differential uniformity* δ of S is given, involving the maximum absolute value in $AC(S)$: $\delta \geq 2^{n-m} + 2^{-m} AC(S)_{max}$, where by $AC(S)_{max}$ it is denoted the maximum *absolute indicator* among the *absolute indicators* of all non-trivial *component* Boolean functions of S. That is, $AC(S)_{max} = \max_{c=(c_1,c_2,\ldots,c_m) \in \mathbb{B}^m \setminus \{0\}} |\widehat{r}_{c_1 f_1 \oplus c_2 f_2 \oplus \ldots \oplus c_1 f_m}(\alpha)|$.

Clearly $\delta \in [2^{n-m}, 2^n]$. If $AC(S)_{max}$ is 0, which is known to be achieved by the *perfect nonlinear (bent)* functions [19, 28], then δ will have the minimum value possible. Thus, small value of δ implies a small value for $AC(S)_{max}$ and hence, minimizing the overall *autocorrelation* of S-boxes, in terms of their $AC(S)_{max}$, will help in providing resistance to *differential cryptanalysis*. Considering that δ is always even, then in the case of bijective S-boxes ($n = m$) can be concluded that the smallest possible value of δ is 2. However, in [3, 4] it is pointed out that the condition of the DDT of an S-box being flat is not sufficient for the S-box to resist *differential cryptanalysis*. In addition, the DDT should also contain as less non-zero entries as possible in its first column. In the case of bijective S-boxes this additional requirement is always fulfilled as the first column of the DDT contains only zeros except for the first entry.

Taking into consideration the mentioned cryptographic properties of S-boxes and the properties of the *coordinate* Boolean functions and their linear combinations, we decided to target the following set of cryptographic criteria for a good bijective S-box S:

1. Minimization of the largest non-trivial value in the LAT of S or $\Longleftrightarrow$
   Maximization of the *nonlinearity* of S, N_S
2. Maximization of the *algebraic degree* of S, $deg(S)$.
3. Minimization of the largest non-trivial value δ in the DDT of S.
4. Minimization of the maximum absolute *autocorrelation* of S, $AC(S)_{max}$.
5. Non-existence of *fixed points*.
6. Non-possession of *linear redundancy*.

We add Criterion 5 to the targeting set of cryptographic criteria as pointed out in [10], although we are not aware of any cryptanalytic attacks in the literature which takes advantage of the existence of *fixed points* in S-boxes.

In [12], Fuller and Millan proposed a new criterion to be added to the above set, the *"non-possession of linear redundancy"*, which is defined bellow. We add it to the targeted set as the Criterion 6.

Definition 1. *An $(n \times m)$ S-box S will possess **linear redundancy** if there are at least two Boolean functions $g(x) = a_1 f_1 \oplus a_2 f_2 \oplus \ldots \oplus a_m f_m$ and $h(x) = b_1 f_1 \oplus b_2 f_2 \oplus \ldots \oplus b_m f_m$ with $a = (a_1, a_2, \ldots, a_m)$ and $b = (b_1, b_2, \ldots, b_m) \in \mathbb{B}^m \setminus \{0\}$, which are affine equivalent.*

When all *component* Boolean functions of S (not counting the zero linear combination) are *affine equivalent*, S will possess *complete linear redundancy*. On the contrary, when no two affine equivalent non-zero linear combinations of the *coordinate* Boolean functions exists, S will possess *zero linear redundancy*. Small S-boxes will always have *linear redundancy* because of the few available equivalence classes for smaller n. However, for larger S-boxes, as the number of equivalence classes becomes quickly infeasible, the presence of *linear redundancy* is claimed to be an indicator for non-randomness and thus a potential source of new cryptanalysis [12].

When optimizing Boolean functions for several cryptographic criteria simultaneously it is known that trade-offs exist. For example, the *perfect nonlinear (bent)* functions, which are known to achieve the highest possible value of *nonlinearity* $2^{n-1} - 2^{\frac{n-2}{2}}$ and the lowest δ-*uniformity* of 2, exist only for even n, but they are never *balanced* and always have small *algebraic degree* ($\leq \frac{n}{2}$). Such trade-offs effect the combined properties of an S-box.

3 Generation Methods

The available in the literature techniques for S-box generation could be divided into three main classes: *algebraic constructions*, *pseudo-random generation* and *heuristic techniques*.

3.1 Algebraic constructions

This approach is based on S-box generation according to certain mathematical principles. The used algebraic constructions rely either on proven mathematical relations or on the construction of bigger S-boxes from smaller ones [13]. Among the first are the finite field inversion mappings, the power mappings, etc. This is the most popular approach, because S-boxes generated in such a way are known to optimize all the desired criteria. The finite field operation of inversion has been shown to achieve the best known combination of high *algebraic degree*, high *nonlinearity* and low *autocorrelation* [24]. For example, an (8×8) S-box constructed in such a way will possess *algebraic degree* of 7, *nonlinearity* of 112, δ-*uniformity* of 4, and maximum non-zero *autocorrelation* of 32. The S-box of AES [10] is constructed with an inverse mapping followed by an affine transformation in the finite field. It is shown that certain algorithmic S-boxes possess *complete linear redundancy* and as such all *component* Boolean functions, except the zero one, are of the same equivalence class [2,33]. The equivalence class of the finite field inversion for the (8×8) case is described in Table 2.

Table 2: Finite Field Inversion Equivalence Class Properties

Algebraic degree: $deg(S)$	7
Nonlinearity: N_S	112
Maximum absolute non-zero autocorrelation: $AC(S)_{max}$	32
Absolute ACT Spectrum Distribution	$\{(0,32),(8,84),(16,74),(24,52),(32,13),(256,1)\}$
Absolute WHT Spectrum Distribution	$\{(0,17),(4,48),(8,36),(12,40),$ $(16,34),(20,24),(24,36),(28,16),(32,5)\}$

Besides the simple algebraic structure of S-boxes obtained by *algebraic constructions* and the potential vulnerability to the algebraic attack [9], these techniques are not typically designed to produce a large set of S-boxes.

3.2 Pseudo-random generation

The second approach is based on using of some *pseudo-random generation* or a table of random numbers to generate the entries in the S-box and then test the S-box whether is good or not. Using of this approach will take a great effort to find a good S-box because of the contradiction of the desired criteria and the small number of good S-boxes among all in the whole space, which quickly becomes infeasible as the size of the input space increases. For example, in the case of (8×8) S-boxes, the highest value for *nonlinearity* found is $98 - 100$ [20, 21], and these with *nonlinearity* 100 found were only four out of 50 million S-boxes generated.

3.3 Heuristic techniques

The *heuristic techniques* involve a process of iteratively improving given S-box or S-boxes with respect to one or more properties. Because they use directed search methods, unlike the *algebraic constructions* they are able of producing a large number of S-boxes, which are not optimal but semi-optimal, often ensure better resistance against algebraic attack and not the least make people believe in the absence of trapdoors. Specific heuristic techniques include the *hill climbing method*, the *simulated annealing method*, the *genetic algorithm* or a combination of these. The *hill climbing method* involves the application of small modifications of one or more distinct elements in order iteratively to improve one or more cryptographic properties. The highest *nonlinearity* achieved by this method is 100 [20]. *Simulated annealing method* provides an extension to the *hill climbing* technique in which the search process is able to move out of a local optimum in order to continue. For the case of (8×8) S-boxes by using this method S-boxes possessing *nonlinearity* 102 are generated [8]. The *genetic algorithms* work with a population of candidate solutions. Aiming to produce future populations of S-boxes possessing desired properties, they apply three operations inspired by natural evolution - *selection, crossover* and *mutation*. During the *selection* process parental pairs, selected from an initial solution pool, interbreed to produce children. The breeding scheme is based on the mechanism of *crossover* like in sexual reproduction. Genetic variation is result of breaking and recombining of parent genes, thus producing offspring with combined parents attributes. The *crossover* mechanism operates on parent genes selected by a crossover point which is chosen randomly. Everything before this point is copied from the first parent and then everything after this point is copied from the other. After the breeding of all parental pairs when all the children are born, a *mutation* is applied on them with the intention of producing random changes in the offspring and thus preventing from falling into a local optimum. Then a *fitness* function to each of the children is applied. Based on the *fitness* values obtained and the solutions in the current pool a decision which of the children and which of the solutions in the current pool will replace the candidate solutions in the parent pool is made. After that a new evolutionary process begins and so on until reaching of some threshold number of iterations chosen in advance. At the end, the fittest solution of the final generation is the best solution. In [32], (8×8) S-boxes having *nonlinearity* 104 are produced by combining a *special genetic algorithm* and *total tree searching*.

4 New Method

The proposed new method is based on a *genetic algorithm* working in a *reversed* way. The basic idea is to start from an initial pool of S-boxes based on a finite field inversion with respect to a polynomial basis and go down from their cryptographic characteristics optimal

values until reaching some close to the optimal and chosen in advance threshold values, which are still satisfactory. The main goal of the algorithm is a rapid construction of a variety of big (from (8×8) up to (16×16)) bijective S-boxes which possess cryptographic properties close to optimal ones, but which have more complex algebraic structure and possess *zero linear redundancy*. A check-up, whether the S-box *nonlinearity* is greater than the *nonlinearity* N_{inv} of S-boxes based on inversion in the finite field, is done and if so, the S-box is saved in a file.

4.1 Genetic Algorithm 1

Genetic algorithm 1 (GA1) works in a *reverse* way, hence we start from S-boxes achieving the optimal known values for *nonlinearity*, *algebraic degree* and δ-*uniformity*. The input of the algorithm is an initial parent pool which contains a number of T S-boxes of dimensions $(n \times n)$. Some of the S-boxes in the initial pool are based on finite field inversion with respect to a polynomial basis and the others are results from applying affine transformations to the finite field inversion S-boxes. The algorithm makes use of three main functions:

1. The breeding function, denoted by $breeding(P_i, P_j, CoP_1, CoP_2, cnt)$, has five input arguments - two distinct parents from the parent pool P_i and P_j, two distinct crossover points CoP_1 and CoP_2, which are randomly generated integers $\in (1, 2^n)$, and a five-valued counter *cnt*. The two distinct crossover points CoP_1 and CoP_2 point out the positions where the breaking of the genes of P_i and P_j respectively is done. The five different values of the counter *cnt* specify the order (straight or reversed) in which the two parts with parents genes divided by the crossover point are copied into the children. Using this approach helps an additional element of randomness to be provided in the computational process by changing the direction of convergence which in turn enables the genetic process to deviate away from a point of local optimum. As an output the breeding function returns two children Ch_1 and Ch_2. The children are obtained by the following crossover scheme: All T S-boxes $P_1, P_2, \ldots, P_T$ in the parental pool are represented by their look-up tables, each containing all 2^n S-box outputs respective to all 2^n possible lexicographically ordered inputs. Thus, as a result of the breeding of the parental pair (P_i, P_j), one of all possible $\frac{T(T-1)}{2}$ pairs, (Ch_1, Ch_2) is obtained. According to the values of the counter *cnt*, the way Ch_1 and Ch_2 are obtained is different. In the case when $cnt = 1$, the rule is:
   $Ch_1(x_k) = P_1(x_k)$, where $x_k \in B$ and $k = 1, 2, \ldots, CoP_1$
   $Ch_1(x_k) = P_2(x_k)$, where $x_k \in B$ and $k = CoP_1 + 1, CoP_1 + 2, \ldots, 2^n$
   $Ch_2(x_p) = P_2(x_p)$, where $x_p \in B$ and $p = 1, 2, \ldots, CoP_2$
   $Ch_2(x_p) = P_1(x_p)$, where $x_p \in B$ and $p = CoP_2 + 1, CoP_2 + 2, \ldots, 2^n$.
   When $cnt = 2, 3, 4$ and 5, one of the two parts of one of the parents is used reversely (a kind of permutation is applied). An example for the case (8×8), $cnt = 1$, $CoP_1 = k$ and $CoP_2 = m$, where $k, m \in (1, 256)$, is given in Table 3.

Table 3: The Crossover Mechanism Example for $cnt = 1$

Parent P_1	y_1	y_2	$\ldots$	y_{k-1}	y_k	y_{k+1}	y_{k+2}	$\ldots$	y_{255}	y_{256}
Parent P_2	v_1	v_2	$\ldots$	v_{m-1}	v_m	v_{m+1}	v_{m+2}	$\ldots$	v_{255}	v_{256}
Child Ch_1	y_1	y_2	$\ldots$	y_{k-1}	y_k	v_{k+1}	v_{k+2}	$\ldots$	v_{255}	v_{256}
Child Ch_2	v_1	v_2	$\ldots$	v_{m-1}	v_m	y_{m+1}	y_{m+2}	$\ldots$	y_{255}	y_{256}

2. The mutation function, denoted by $modeling(Ch)$, has as an input argument any child born in result of the breeding of any of the parental pairs. The function is applied with two main purposes. The first one comes in response to the unwanted mutation which may have occurred in the children after the crossover, namely the loss of their bijection property. So, the bijection must be repaired. And the second purpose is some additional randomness to the computational process to be added which will help in providing a chance for deviation from a local optimum similarly to the usage of the counter cnt. The output of the mutation function is a child which is a permutation. The modeling process is described bellow:
Let $(Ch_1, Ch_2) = breeding(P_i, P_j, CoP_1, CoP_2, cnt)$ and let their Look-up tables are:

$$Ch_1 = [y_1, y_2, \ldots, y_{CoP_1-1}, y_{CoP_1}, y_{CoP_1+1}, \ldots, y_{2^n}]$$
$$Ch_2 = [v_1, v_2, \ldots, v_{CoP_2-1}, v_{CoP_2}, v_{CoP_2+1}, \ldots, v_{2^n}].$$

Clearly, each of the two parts $\{y_1, y_2, \ldots, y_{CoP_1}\}$ and $\{y_{CoP_1+1}, y_{CoP_1+2}, \ldots, y_{2^n}\}$ of Ch_1, and each of the two parts $\{v_1, v_2, \ldots, v_{CoP_2}\}$ and $\{v_{CoP_2+1}, v_{CoP_2+2}, \ldots, v_{2^n}\}$ of Ch_2, divided respectively by CoP_1 and CoP_2, consist of distinct elements because of the bijective property of the parents. So, as accurately to the order, $\{y_1, y_2, \ldots, y_{CoP_1}\}$ is part of P_i and $\{y_{CoP_1+1}, y_{CoP_1+2}, \ldots, y_{2^n}\}$ is part of P_j, some of the elements of $\{y_1, y_2, \ldots, y_{CoP_1}\}$ can be repeated in $\{y_{CoP_1+1}, y_{CoP_1+2}, \ldots, y_{2^n}\}$. The modeling of the child consists in one by one consequently checking for repetition the elements of $\{y_{CoP_1+1}, y_{CoP_1+2}, \ldots, y_{2^n}\}$ in $\{y_1, y_2, \ldots, y_{CoP_1}\}$. If so, a new element is repeatedly randomly generated, and if it is not met yet, it replaces the duplicate. The procedure is applied for both of the children. At the end, they are permutations.

3. The fitness function, denoted by $fitness(Ch)$, has as an input argument any child, which has already been modeled, and returns a value playing the role of a measure taken to ascertain whether the child will survive to the new generation or not. The fitness value is the nonlinearity value of the child N_{ch}. The fitness test is passed by the child Ch if $N_{Ch} \geq N_{thr}$, where N_{thr} is the threshold *nonlinearity* value chosen in advance. If $N_{Ch} > N_{thr}$, then the child takes its place in the offspring pool. If $N_{Ch} = N_{thr}$, then besides the child takes its place in the offspring pool, it is also saved in a file. If $N_{Ch} > N_{inv}$, then the child is saved in a file. Otherwise, the child does not survive to the next generation and it is left off. If after the breeding of all pairs from the parent pool the offspring pool is not totally full, the breeding process starts all over again with the pair (P_1, P_2). This makes sense, because the crossover points CoP_1 and CoP_2 are randomly chosen which ensures that the next pair of children of P_1 and P_2 will be different from the previous one. When the offspring pool gets full of children, if all of them have *nonlinearities* equal to the threshold value N_{thr}, the algorithm stops. Otherwise the children replace their parents in the parent pool and so on to the next generation.

4.2 Genetic Algorithm 2

The *Genetic algorithm 2 (GA2)* provided in this section is a slight modification of *GA1* in which an additional cost function is applied together with the fitness function in order to help deciding whether the respective child will survive to the next generation or not. The cost function is based on the *WHT spectrum* and is taken from the family of functions in [8]:

$$cost(S) = \sum_{c=(c_1,c_2,\ldots,c_m)\in\mathbb{B}^m\setminus\{0\}} \sum_{\omega\in\mathbb{B}^n} |\widehat{F}_{c_1f_1\oplus c_2f_2\oplus\ldots\oplus c_mf_m}(\omega) - X|^R,$$

where X and R are real-valued parameters and $\widehat{F}_{c_1 f_1 \oplus c_2 f_2 \oplus \ldots \oplus c_m f_m}(\omega)$ is the *WHT* of the *component* Boolean function $c_1 f_1 \oplus c_2 f_2 \oplus \ldots \oplus c_m f_m$ of the S-box given by c. More precisely, *GA2* uses the specific cost function for $m = n$ with $(X, R) = (21, 7)$, as proposed in [32].

The main difference between the two genetic algorithms proposed is that in addition to *GA1* in *GA2* a child, which has already been modeled, passes the fitness test not only in the case when its fitness value is good enough, i.e. $N_{Ch} \geq N_{thr}$, but also when in addition its cost value is smaller than the one of its parent, i.e. $cost(Ch_1) < cost(P_1)$. The pseudo-codes of both algorithms are presented in Appendix A.

5 Experimental Results

In this section the results of generating a set of T $(n \times n)$ bijective S-boxes using the two variants of *genetic algorithms* are provided.

5.1 Results Obtained with *GA1*

1) The case $n = 8$ The results obtained by *GA1* are closely related to the *linear redundancy* property. If it is considered as a vital one, the output of the *GA1* are S-boxes which possess *zero linear redundancy*, but their *nonlinearities* are at most 108. Otherwise, S-boxes with *nonlinearity* 110 and 112 are achieved, but they possess *some linear redundancy*.

The properties of representatives obtained with *GA1* for the case $n = 8$ with an initial pool full of 200 S-boxes and N_{thr} equal to $104, 106$ and 108 are provided in Table 4.

Table 4: S-boxes generated with *GA1* in the case $n = 8$ and $N_{thr} = 104, 106$ and 108

S-BOX	N_{thr}	N_S	$\deg(S)$	$AC(S)_{max}$	δ-uniformity	Fixed points	Linear Redundancy
AES S-BOX	–	112	7	32	4	2	*complete*
GA1 S-BOX 1	104	104	7	64	6	2	*zero*
GA1 S-BOX 2	106	106	6	56	6	2	*zero*
GA1 S-BOX 3	108	108	6	48	6	0	*zero*

The respective cryptographic properties distributions of the S-box *component* Boolean functions are provided in Table 5, where the pair $(value_1, value_2)$ means that the respective cryptographic $value_1$ has occurred exactly a $value_2$ number of times among all 255 non-trivial S-box *component* Boolean functions.

Table 5: Distributions of the *component* Boolean functions of *GA1* S-boxes in the case $n = 8$

LCBFs distributions	N_{LCBFs}	$\deg(LCBFs)$	$AC_{max}(LCBFs)$	Equivalence classes
AES S-BOX	$(112, 255)$	$(7, 255)$	$(32, 255)$	1
GA1 S-BOX 1	$(112, 14)$ $(110, 116)$ $(108, 92)$ $(106, 29)$ $(104, 4)$	$(7, 255)$	$(64, 6)$ $(56, 54)$ $(48, 141)$ $(40, 54)$	255
GA1 S-BOX 2	$(112, 60)$ $(110, 160)$ $(108, 33)$ $(106, 2)$	$(7, 254)$ $(6, 1)$	$(56, 2)$ $(48, 63)$ $(40, 180)$ $(32, 10)$	255
GA1 S-BOX 3	$(112, 99)$ $(110, 149)$ $(108, 7)$	$(7, 254)$ $(6, 1)$	$(48, 14)$ $(40, 202)$ $(32, 39)$	255

As it can be seen from Tables 4 and 5, the obtained S-boxes possess *zero linear redundancy*, that is all non-trivial 255 *component* Boolean functions belong to distinct equivalence classes. This is an improvement in terms of *linear redundancy* compared to the inversion S-boxes where all 255 *component* Boolean functions are from the same equivalence class (Table 2) and hence a *complete linear redundancy* is available. The particular S-boxes are presented in Appendix B in hexadecimal notation.

2) The case $n = 16$ The properties of representatives for $n = 16$ with an initial pool full of 50 S-boxes and N_{thr} equal to $32400, 32428$ and 32476 are provided in Table 6.

Table 6: S-boxes generated with *GA1* in the case $n = 16$ and $N_{thr} = 32400, 32428$ and 32476

S-BOX	N_{thr}	N_S	$\deg(S)$	$AC(S)_{max}$
Inversion S-BOX	–	32512	15	512
GA1 S-BOX 1	32400	32400	15	976
GA1 S-BOX 2	32428	32428	15	864
GA1 S-BOX 3	32476	32476	14	616

5.2 Results Obtained with *GA2*

GA2 is a little slower than *GA1*. Furthermore, it should be noted that its results are better only in terms of *linear redundancy*. S-boxes with *nonlinearity* 110 and *zero linear redundancy* were obtained. One reason could be the additional cost function in the fitness which slows up the offspring obtaining and thus inserts more randomness in the process. The other one could be that in all tests with *GA2*, different paths have been selected because of the randomness inserted in the breeding and the mutation step. The properties of representatives obtained with *GA2* for $n = 8$ with an initial pool of 200 S-boxes and N_{thr} equal to 106, 110 and 112 are provided in Table 7. The distributions of the S-box *component* functions are provided in Table 8. Similar to *GA1*, the improvement in terms of *zero linear redundancy* can also be observed, excluding the case $N_{trh} = 112$, where only 147 out of 255 distinct equivalence classes were achieved. The particular S-boxes can be viewed in Appendix C.

We should note that with *GA1* and *GA2* we found no S-boxes satisfying $N_{Ch} > N_{inv}$.

Table 7: S-boxes generated with *GA2* in the case $n = 8$ and $N_{thr} = 106, 110$ and 112

S-BOX	N_{thr}	N_S	$\deg(S)$	$AC(S)_{max}$	δ-uniformity	Fixed points	Linear Redundancy
AES S-BOX	–	112	7	32	4	2	*complete*
GA2 S-BOX 1	106	106	6	48	6	0	*zero*
GA2 S-BOX 2	110	110	7	40	6	0	*zero*
GA2 S-BOX 3	112	112	7	32	6	0	*some* (147 of 255)

The proposed method is fast enough in repeatedly producing of thousands of (8×8) S-boxes with *nonlinearity* up to 112. Together with the high *nonlinearity*, most of the S-boxes possess properties which are close to the finite filed inversion-based ones. At a reasonable price of small deviations from the best known values, more complex algebraic structure and *zero linear redundancy* is achieved except for $N_{thr} = 112$, where the maximum number of different equivalence classes for all *component* Boolean functions found was 147. Even for $n = 16$ the

Table 8: Distributions of the *component* Boolean functions of *GA2* S-boxes in the case $n = 8$

$LCBFs$ distributions	N_{LCBFs}	$\deg(LCBFs)$	$AC_{max}(LCBFs)$	Equivalence classes
AES S-BOX	$(112, 255)$	$(7, 255)$	$(32, 255$	1
GA2 S-BOX 1	$(112, 85)$ $(110, 150)$ $(108, 18)$ $(106, 102)$	$(7, 254)$ $(6, 1)$	$(48, 46)$ $(40, 160)$ $(32, 49)$	255
GA2 S-BOX 2	$(114, 1)$ $(112, 162)$ $(110, 92)$	$(7, 255)$	$(40, 122)$ $(32, 133)$	255
GA2 S-BOX 3	$(112, 255)$	$(7, 255)$	$(32, 255)$	147

algorithm is relatively fast if N_{thr} is chosen to be close to $N_{inv} = 32512$. It has found 50 S-boxes in 3 and a half months. We can obtain S-boxes of each possible *nonlinearity* and the closer to N_{inv} the faster the algorithm is. Considering the huge time and space resources needed for the computations when generating (16×16) S-boxes, the proposed method proves to be more applicable than any of the known *pseudo-random* and *heuristic techniques*.

References

1. E. Biham. On Matsui's linear cryptanalysis. In *Eurocrypt'94*, volume 950 of *LNCS*, pages 341–355. Springer, 1994.
2. E. Biham. Observations on the relations between bit-functions of many s-boxes. In *The 3rd NESSIE conference*, November 2002.
3. E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. In *Advances in Cryptology CRYPTO'90*, volume 537 of *LNCS*, pages 2–21. Springer Verlag, 1991.
4. E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, 4:3–72, 1991.
5. C. Carlet. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, chapter Boolean Functions for Cryptography and Error Correcting Codes, pages 257–397. Cambridge University Press, 2010.
6. C. Carlet. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, chapter Vectorial Boolean Functions for Cryptography, pages 257–397. Cambridge University Press, 2010.
7. F. Chabaud and S. Vaudenay. Links between differential and linear cryptanalysis. In *Advances in Cryptology EUROCRYPT'94*, volume 950 of *LNCS*, pages 356–365. Springer Verlag, 1995.
8. J.A. Clark, J.L. Jacob, and S. Stepney. The design of s-boxes by simulated annealing. *New Generation Computing Archive*, 23(3), September 2005.
9. N. T. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology - ASIACRYPT'02*, volume 2501 of *LNCS*, pages 267–287. Springer Verlag, 2002.
10. J. Daeman and V. Rijmen. *The design of Rijndael: AES The advanced Encryption Standard*. Springer Verlag, 2002.
11. J. Daemen, R. Govaerts, and J. Vandewalle. Correlation matrices. In *FSE'94*, volume 1008 of *LNCS*, pages 275–285. Springer, 1995.
12. J. Fuller and W. Millan. Linear redundancy in s-boxes. In *FSE'03*, volume 2887 of *LNCS*, pages 74–86. Springer, 2003.
13. B. Gerard, V. Grosso, M. Naya-Plasencia, and F.-X. Standaert. Block ciphers that are easier to mask: How far can we go? In *CHES'03*, volume 8086 of *LNCS*, pages 383–399. Springer, 2003.
14. J. Dj. Golić. Fast low order approximation of cryptographic functions. In *Advances in Cryptology EUROCRYPT'96*, volume 1070 of *LNCS*, pages 268–282. Springer Verlag, 1996.
15. O. Kazymyrov, V. Kazymyrova, and R. Oliynykov. A method for generation of high-nonlinear s-boxes based on gradient descent. *IACR Cryptology ePrint Archive (2013), http://eprint.iacr.org/2013/578*.
16. L.Goubin, A.Martinelli, and M.Walle. Impact of s-boxes size upon side channel resistance and block cipher design. In *AFRICACRYPT'13*, volume 7918 of *LNCS*, pages 240–259. Springer, 2013.
17. M. Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397. Springer Verlag, 1994.
18. F. J. McWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1978.
19. W. Meier and O. Staffelbach. Nonlinearity criteria for cryptographic functions. In *Advances in Cryptology EUROCRYPT'89*, volume 434 of *LNCS*, pages 549–562. Springer Verlag, 1990.
20. W. Millan. How to improve the nonlinearity of bijective s-boxes. In *Australian Conference on Information Security and Privacy 1998*, volume 1438, pages 181–192. Springer Verlag, 1998.
21. W. Millan, L. Burnett, G. Carter, A. Clark, and E. Dawson. Evolutionary heuristics for finding cryptographically strong s-boxes. In *ICICS'99*, volume 1726 of *LNCS*, pages 263–274. Springer, 1999.
22. W. L. Millan. Low order approximation of cipher functions. In *Cryptography: Policy and Algorithms Conference, Proceedings*, volume 1029 of *LNCS*, pages 144–155. Springer Verlag, 1996.
23. K. Nyberg. Perfect nonlinear s-boxes. In *Advances in Cryptology EUROCRYPT'91*, volume 547 of *LNCS*, pages 378–386. Springer Verlag, 1992.
24. K. Nyberg. Differentially uniform mappings for cryptography. In *Advances in Cryptology EUROCRYPT'93*, volume 765 of *LNCS*, pages 55–64. Springer Verlag, 1994.
25. B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, KU Leuven, 1994.
26. L. Qu, Y. Tan, C. Li, and G. Gong. More constructions of differentially 4-uniform permutations on $\mathbb{F}_{2^{2k}}$. In *arxiv.org/pdf/1309.7423*, 2013.
27. L. Qu, Y. Tan, C. Tan, and C. Li. Constructing differentially 4-uniform permutations over $\mathbb{F}_{2^{2k}}$ via the switching method. *IEEE Transactions on Inform. Theory*, 59(7):4675–4686, 2013.
28. O. S. Rothaus. On bent functions. *Journal of Combinatorial Theory*, 20(3):300–305, May 1976.
29. J. Seberry, X. M. Zhang, and Y. Zheng. Systematic generation of cryptographically robust s-boxes. In *Proceedings of the first ACM Conference on Computer and Communications Security*, pages 171–182. The Association for Computing Machinery, Fairfax, VA, 1993.

30. J. Seberry, X. M. Zhang, and Y. Zheng. Relationships among nonlinearity criteria. In *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *LNCS*, pages 376–388. Springer Verlag, 1995.
31. C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.
32. P. Tesař. A new method for generating high non-linearity s-boxes. *Radioengineering*, 19(1):23–26, 2010.
33. A.M. Youssef and S.E. Tavares. On some algebraic structures in the aes round function. Technical Report 2002/144, Cryptology ePrint Archive, 2002.
34. X. Zhang, Y. Zheng, and H. Imai. Relating differential distribution tables to other properties of substitution boxes. *Designs, Codes and Cryptography*, 19:45–63, 2000.

A Appendix A

A.1 Genetic Algorithm 1 pseudo code

1. Step 1 (Initializing step)
   In this step the parameters of the algorithm are defined:
   - Define an integer n representing the dimension of the $(n \times n)$ S-box.
   - Define an integer T representing the number of the S-boxes in the parent pool (PP).
   - Define an even integer $N_{thr} \leq N_{inv}$ representing the nonlinearity threshold value.
   - Generate a number of T $(n \times n)$ S-boxes and put them in the (PP). Some based on inversion in the finite field $GF[2^n]$ and some obtained in result of application of affine transformations to the generated algorithmic S-boxes.
   - Create an empty offspring pool (OP) of size T.
   - Set the counter cnt value to be 0.
   - Set the parents indexes t and r to be 1.
2. Step 2 (Breeding step)

   if $(cnt = 6)$ then
   $cnt = 0$
   $r = r + 1$
   if $(r \leq T)$ then
   $cnt = cnt + 1$
   $(Ch_1, Ch_2) = breeding(P_t, P_r, CoP_1, CoP_2, cnt)$
   else
   if $(t < T)$ then
   $t = t + 1$
   $r = t + 1$
   $cnt = cnt + 1$
   $(Ch_1, Ch_2) = breeding(P_t, P_r, CoP_1, CoP_2, cnt)$
   else
   if $(|OP| = T)$
   if $(N_{Ch_j} = N_{thr})$ for each child Ch_j in the OP $(j = 1$ to $T)$ then
   STOP the algorithm
   go to Step 5
   else
   move and replace OP into PP
   $cnt = 0$
   $t = 1$
   $r = 1$
   go to Step 2
   else
   $cnt = 0$
   $t = 1$
   $r = 1$
   go to Step 2
   go to Step 3

3. Step 3 (Mutation step)
   $Ch_1 = modeling(Ch_1)$ and $Ch_2 = modeling(Ch_2)$
   go to Step 4

4. Step 4 (Fitness step)

   $N_{Ch_1} = fitness(Ch_1)$ and $N_{Ch_2} = fitness(Ch_2)$
   if $(N_{Ch_1} > N_{inv})$ then
     save Ch_1 in a file
   if $(N_{Ch_1} < N_{thr})$ then
     delete Ch_1
   if $(N_{Ch_1} \geq N_{thr}$ and $|OP| < T)$ then
     save Ch_1 in the OP
     $|OP| = |OP| + 1$
     if $(N_{Ch_1} = N_{thr})$ then
       save Ch_1 in a file
   if $(N_{Ch_2} > N_{inv})$ then
     save Ch_2 in a file
   if $(N_{Ch_2} < N_{thr})$ then
     delete Ch_2
   if $(N_{Ch_2} \geq N_{thr}$ and $|OP| < T)$ then
     save Ch_2 in the OP
     $|OP| = |OP| + 1$
     if $(N_{Ch_2} = N_{thr})$ then
       save Ch_2 in a file
   if $(|OP| < T)$ then
     go to Step 2
   else
     if $(N_{Ch_j} = N_{thr})$ for each child Ch_j in the OP $(j = 1$ to $T)$ then
       STOP the algorithm
       go to Step 5
     else
       move and replace OP into PP
       $cnt = 0$
       $t = 1$
       $r = 1$
       go to Step 2

5. Step 5 (Solution pool)
   In the offspring pool OP are the all desired T S-boxes with $N = N_{thr}$.

A.2 Genetic Algorithm 2 pseudo code

The pseudo code of *GA2* is exactly the same as the pseudo code of *GA1* with a few exceptions. Only lines 6 and 15 of *GA1* fitness step, namely
"if $(N_{Ch_1} \geq N_{thr}$ and $|OP| < T)$ then" and
"if $(N_{Ch_2} \geq N_{thr}$ and $|OP| < T)$ then",
are replaced with
"if $(N_{Ch_1} \geq N_{thr}$ and $C_{Ch_1} < C_{P_1}$ and $|OP| < T)$ then" and
"if $(N_{Ch_2} \geq N_{thr}$ and $C_{Ch_2} < C_{P_2}$ and $|OP| < T)$ then" respectively,
where $C_{P_1} = cost(P_1)$, $C_{P_2} = cost(P_2)$, $C_{Ch_1} = cost(Ch_1)$ and $C_{Ch_2} = cost(Ch_2)$.

B S-boxes generated with *GA1*

B.1 S-box No 1 ($N_S = 104, deg(S) = 7, AC(S)_{max} = 64, \delta = 6$)

{0x52 0x53 0xDF 0xA4 0x99 0x00 0x29 0x83 0xBA 0x1D 0x7B 0x92 0xE2 0xB3 0xB7 0x95
0x26 0xE6 0xF8 0x19 0xCB 0x79 0x32 0x0D 0x0A 0x6D 0xAF 0x9E 0xAD 0x12 0xBC 0xE0
0x68 0x3C 0x08 0xA3 0x07 0x1F 0xFA 0x9B 0x93 0x58 0xCA 0x47 0x62 0x16 0xF0 0x90
0x7E 0x17 0xC0 0x3E 0xA1 0x6B 0x34 0x10 0xA0 0x67 0x72 0x3D 0x25 0xE9 0x0B 0x4B
0x4F 0xAC 0x65 0x35 0x7F 0x63 0xA7 0x3B 0xF5 0x36 0xF9 0x41 0x06 0x77 0xBB 0x5B
0xBF 0x0E 0x57 0x98 0x1E 0x76 0xD5 0xED 0x4A 0x6C 0x70 0xA2 0x03 0xBE 0x33 0x45
0x44 0x0C 0xFD 0x81 0x1B 0xF4 0x64 0x11 0xA6 0x15 0xC3 0x8D 0x61 0xC1 0x73 0x69
0x2B 0xE5 0xC5 0xD7 0x42 0xE7 0xE8 0x6E 0xE4 0x22 0x82 0x54 0xF3 0xA8 0xD3 0xD0
0xD1 0x2C 0x2D 0xD2 0xC4 0x21 0xEC 0x04 0xC9 0xCC 0xC7 0x8B 0xA5 0x50 0xEB 0xF6
0x8C 0x38 0x60 0x3F 0x8A 0xD8 0xD6 0x20 0x78 0x46 0xCD 0xDA 0xAB 0x8E 0xDB 0xC8
0xA9 0x2E 0x7C 0x91 0xDD 0xEA 0x37 0x1A 0x74 0x9A 0x40 0x18 0x9C 0xB5 0x80 0x30
0x5E 0xB2 0x4D 0xBD 0x43 0x27 0x2A 0x23 0xF7 0xDC 0x24 0x6F 0xEF 0xEE 0xD4 0x05
0x59 0x7A 0x7D 0xF1 0x88 0x86 0xB6 0x5D 0xFB 0x75 0x01 0x56 0x49 0xAE 0xFE 0xB4
0x28 0x55 0xFC 0x31 0x97 0x89 0xB0 0xB8 0xC6 0xD9 0x96 0x87 0xCF 0xAA 0xC2 0x39
0xE3 0x5F 0x84 0xB9 0x94 0x5C 0x9D 0xFF 0x5A 0x1C 0x85 0xB1 0x0F 0x02 0x4C 0xF2
0x4E 0x71 0x48 0x9F 0x14 0xCE 0x09 0x51 0x66 0xE1 0x6A 0x3A 0xDE 0x13 0x8F 0x2F}

B.2 S-box No 2 ($N_S = 106, deg(S) = 6, AC(S)_{max} = 56, \delta = 6$)

{0x50 0x51 0x93 0xD2 0xF2 0x2E 0x11 0x0A 0x01 0x66 0x6F 0xFC 0xB3 0x38 0x7D 0x7A
0xBB 0xCB 0x4B 0x65 0x8C 0x4E 0x06 0xF5 0xE2 0x24 0x64 0x42 0x85 0x34 0x45 0x8D
0xE6 0x1B 0xDE 0xAB 0x9E 0xB9 0x89 0xF1 0x3E 0x8B 0x5F 0x7C 0x7B 0x5E 0xC1 0xA1
0x09 0x87 0x6A 0xA4 0x4A 0x43 0x59 0x00 0xF9 0x33 0x62 0xA5 0x99 0x9C 0xFD 0x5A
0x0B 0x56 0xB6 0xA7 0x17 0xEF 0xEE 0x14 0x37 0x2B 0xE7 0x71 0xFF 0x03 0xC3 0xAF
0x67 0x58 0xFE 0x1D 0x94 0x81 0x46 0xF4 0x86 0x60 0x57 0x10 0xDB 0xCD 0xEB 0xDC
0xBF 0xD1 0xF8 0x69 0x4D 0x84 0x2A 0x18 0x5D 0xB2 0x9A 0xE0 0x97 0x8E 0x78 0x8A
0xC7 0x82 0xA2 0xD4 0x49 0xE3 0xE9 0xD7 0xF7 0xB4 0x36 0x19 0xC5 0xC9 0x55 0xF3
0xBE 0x31 0x53 0x92 0x23 0xA3 0xE8 0x27 0xB0 0xA8 0xCC 0x0C 0x0F 0xEA 0x72 0xAA
0xA0 0x7E 0xAE 0x1E 0xC8 0x2C 0x83 0x20 0xC4 0x2D 0xBA 0x41 0xDA 0x0D 0xEC 0xBC
0x88 0x77 0x54 0x2F 0x07 0x47 0xB5 0x28 0x32 0x68 0xFB 0xFA 0x5B 0x6E 0x02 0x1C
0x3B 0x9B 0x48 0x25 0x90 0xAD 0x70 0x1A 0xD6 0x26 0xDD 0x0E 0xCE 0xBD 0x16 0x15
0xE4 0xAC 0xD3 0x52 0x04 0x80 0x8F 0x3C 0x9D 0x6C 0x3A 0xE1 0x6D 0x98 0x74 0xB8
0x95 0x05 0x21 0xC6 0x35 0x4C 0x08 0x61 0xF0 0x76 0x3F 0x79 0x44 0x4F 0x3D 0x96
0xD8 0xA9 0x39 0x5C 0x29 0xF6 0x12 0xA6 0x9F 0x75 0xCA 0x40 0xCF 0xED 0xD0 0x30
0xC0 0x7F 0x22 0xD5 0x63 0x6B 0xB7 0x13 0xC2 0xDF 0xE5 0x73 0xB1 0x91 0x1F 0xD9}

B.3 S-box No 3 ($N_S = 108, deg(S) = 6, AC(S)_{max} = 48, \delta = 6$)

{0x97 0x96 0x1A 0x26 0x1B 0x82 0xAB 0x01 0x38 0x9F 0xF9 0x10 0x60 0x31 0x35 0x17
0xA4 0x64 0x7A 0x9B 0x49 0xFB 0xB0 0x8F 0x88 0xEF 0x2D 0x1C 0x2F 0x90 0x3E 0x62
0xEA 0xBE 0x8A 0x21 0x85 0x9D 0x78 0x19 0x11 0xDA 0x48 0xC5 0xE0 0x94 0x72 0x12
0xFC 0x95 0x42 0xBC 0x23 0xE9 0xB6 0x92 0x22 0xE5 0xF0 0xBF 0xA7 0x6B 0x89 0xC9
0xCD 0x2E 0xE7 0xB7 0xFD 0xE1 0x25 0xB9 0x77 0xB4 0x7B 0xC3 0x84 0xF5 0x39 0xD9
0x3D 0x8C 0xD5 0xD1 0x9C 0xF4 0x57 0x6F 0xC8 0xEE 0xF2 0x20 0x81 0x3C 0xB1 0xC7
0xC6 0x8E 0x7F 0x03 0x99 0x76 0xE6 0x93 0x24 0x5D 0x41 0x0F 0xE3 0x43 0xF1 0xEB
0xA9 0x67 0x47 0x55 0xC0 0x65 0x6A 0xEC 0x66 0xA0 0x00 0xD6 0x71 0x2A 0x51 0x52
0x53 0xAE 0xAF 0x50 0x46 0xA3 0x6E 0x86 0x4B 0x4E 0x45 0x09 0x27 0xD2 0x69 0x74
0x0E 0xBA 0xE2 0xBD 0x08 0x5A 0x54 0xA2 0xFA 0xC4 0x4F 0x58 0x29 0x0C 0x59 0x4A
0x2B 0xAC 0xFE 0x13 0x5F 0x68 0xB5 0x98 0xF6 0x18 0xC2 0x9A 0x1E 0x37 0x02 0xB2
0xDC 0x30 0xCF 0x3F 0xC1 0xA5 0xA8 0xA1 0x75 0x5E 0xA6 0xED 0x6D 0x6C 0x56 0x87
0xDB 0xF8 0xFF 0x73 0x0A 0x04 0x34 0xDF 0x79 0xF7 0x83 0xD4 0xCB 0x2C 0x7C 0x36
0xAA 0xD7 0x7E 0xB3 0x15 0x0B 0x32 0x3A 0x44 0x5B 0x14 0x05 0x4D 0x28 0x40 0xBB
0x61 0xDD 0x06 0x3B 0x16 0xDE 0x1F 0x7D 0xD8 0x9E 0x07 0x33 0x8D 0x80 0xCE 0x63
0x8B 0xF3 0xE8 0xE4 0xB8 0xD0 0xD3 0x5C 0x0D 0x4C 0xAD 0x70 0x1D 0xCA 0x91 0xCC}

C S-boxes generated with *GA2*

C.1 S-box No 1 ($N_S = 106, deg(S) = 6, AC(S)_{max} = 48, \delta = 6$)

{0x82 0xD3 0x21 0x1F 0x95 0xDC 0x4E 0x86 0x5A 0x68 0x8D 0x47 0xC4 0x31 0xA0 0x5E
0xCE 0x20 0xD7 0xB6 0x56 0x2C 0x33 0x05 0x81 0x8F 0x08 0x32 0xB3 0x8A 0xCC 0x58
0x84 0x22 0xF3 0x5C 0x7B 0x1E 0xB8 0x6C 0xC8 0x71 0xF5 0x6F 0x09 0x04 0x12 0xC5
0x50 0xD4 0x57 0x0A 0xE7 0x78 0xFA 0x4D 0x49 0xB4 0xA6 0x97 0x85 0x3E 0xCF 0x0E
0xA1 0x10 0xF2 0x3C 0x69 0x17 0xCD 0x00 0x2D 0x0B 0xA2 0xDB 0xBF 0x67 0xD5 0x2F
0x87 0x19 0x28 0xFB 0x6A 0xB1 0x27 0xB5 0x14 0x8C 0xE1 0xD9 0xEA 0x9C 0x72 0x9F
0xCB 0xEE 0x89 0xA9 0x3B 0x83 0xE6 0x2A 0x63 0x93 0xDF 0xC1 0x9E 0x41 0x36 0xC9
0x34 0xF4 0xB9 0x38 0xB0 0x4B 0x5B 0x16 0x52 0xBE 0xFC 0x98 0x77 0x92 0xE4 0xAB
0x40 0x73 0xEB 0x42 0x9A 0x9B 0xFD 0x64 0x24 0xA7 0x1B 0xDD 0x76 0x62 0xE3 0xEC
0x06 0x80 0x15 0x46 0xB2 0x02 0x7D 0xA8 0x4F 0x18 0x23 0x3F 0x7A 0x3A 0x07 0x8E
0x53 0x66 0x1C 0xED 0xF7 0xD0 0x6D 0x39 0xD6 0x0C 0x48 0x26 0x03 0x8B 0x4A 0x6B
0xE9 0xF0 0xA5 0xC7 0x60 0xE5 0x7C 0x88 0x96 0x25 0xAD 0xC6 0xDA 0x55 0x5F 0x11
0x75 0xC0 0x94 0x1A 0x54 0xA3 0x44 0xE0 0x0D 0xB7 0x51 0x2E 0x90 0xE2 0xF6 0xBA
0xBD 0x0F 0x59 0x01 0x7F 0xEF 0x70 0x37 0xAC 0xA4 0x30 0x13 0xF8 0xFE 0x74 0xDE
0xF9 0xC2 0x99 0x65 0x4C 0x29 0xFF 0xC3 0xBB 0xD1 0x35 0x6E 0x3D 0x5D 0xE8 0xAE
0xCA 0x7E 0xBC 0x1D 0x9D 0x43 0xAF 0xF1 0x2B 0x79 0xAA 0x61 0x91 0xD2 0x45 0xD8}

C.2 S-box No 2 ($N_S = 110, deg(S) = 7, AC(S)_{max} = 40, \delta = 6$)

{0x1D 0x2B 0xD9 0x88 0xA0 0xE9 0x7B 0xB3 0x6F 0x5D 0xB8 0x72 0xF1 0x04 0x95 0x6B
0xFB 0x15 0xE2 0x83 0x63 0x19 0x06 0x30 0xB4 0xBA 0x3D 0x07 0x86 0xBF 0xF9 0x6D
0xB1 0x17 0xC6 0x69 0x4E 0xB7 0x8D 0x59 0xFD 0x44 0xC0 0x5A 0x3C 0x31 0x27 0xF0
0x65 0xE1 0x62 0x3F 0xD2 0x4D 0xCF 0x78 0x7C 0x81 0x93 0xA2 0xB0 0x0B 0xFA 0x3B
0x94 0x25 0xC7 0x09 0x5C 0x22 0xF8 0x35 0x18 0x3E 0x97 0xEE 0x8A 0x52 0xE0 0x1A
0xB2 0x2C 0xD5 0xCE 0x5F 0x84 0x12 0x80 0x21 0xB9 0xD4 0xEC 0xDF 0xA9 0x47 0xAA
0xFE 0xDB 0xBC 0x9C 0x0E 0xB6 0xD3 0x1F 0x56 0xA6 0xEA 0xF4 0x2A 0x74 0x03 0xFC
0x01 0xC1 0x8C 0x0D 0x85 0x7E 0x6E 0x23 0x67 0x8B 0xC9 0xAD 0x42 0xA7 0xD1 0x9E
0x75 0x46 0xDE 0x77 0xAF 0xAE 0xC8 0x51 0x11 0x92 0x2E 0xE8 0x43 0x57 0xD6 0xE6
0x33 0xB5 0x20 0x73 0x87 0x37 0x48 0x9D 0x7A 0x2D 0x16 0x0A 0x4F 0x0F 0x32 0xBB
0x66 0x53 0x29 0xD8 0xC2 0xE5 0x58 0x0C 0xE3 0x39 0x7D 0x13 0x36 0xBE 0x7F 0x5E
0xDC 0xC5 0x90 0xF2 0x55 0xD0 0x49 0xBD 0xA3 0x10 0x98 0xF3 0xEF 0x60 0x6A 0x24
0x40 0xF5 0xA1 0x2F 0x61 0x96 0x71 0xAB 0x38 0x82 0x64 0x1B 0xA5 0xD7 0xC3 0x8F
0x14 0x3A 0x6C 0x34 0x4A 0xDA 0x45 0x02 0x99 0x91 0x05 0x26 0xCD 0xCB 0x41 0xEB
0xCC 0xF7 0xAC 0x50 0x79 0x1C 0xCA 0xF6 0x8E 0xE4 0x00 0x5B 0x08 0x68 0xDD 0x9B
0xFF 0x4B 0x89 0x28 0xA8 0x76 0x9A 0xC4 0x1E 0x4C 0x9F 0x54 0xA4 0xE7 0x70 0xED}

C.3 S-box No 3 ($N_S = 112, deg(S) = 7, AC(S)_{max} = 32, \delta = 6$)

{0x1E 0x2B 0xD9 0x88 0xA0 0xE9 0x7B 0xB3 0x6F 0x5D 0xB8 0x72 0xF1 0x04 0x95 0x6B
0xFB 0x15 0xE2 0x83 0x63 0x19 0x06 0x30 0xB4 0xBA 0x3D 0x07 0x86 0xBF 0xF9 0x6D
0xB1 0x17 0xC6 0x69 0x4E 0xB7 0x8D 0x59 0xFD 0x44 0xC0 0x5A 0x3C 0x31 0x27 0xF0
0x65 0xE1 0x62 0x3F 0xD2 0x4D 0xCF 0x78 0x7C 0x81 0x93 0xA2 0xB0 0x0B 0xFA 0x3B
0x94 0x25 0xC7 0x09 0x5C 0x22 0xF8 0x35 0x18 0x3E 0x97 0xEE 0x8A 0x52 0xE0 0x1A
0xB2 0x2C 0x1D 0xCE 0x5F 0x84 0x12 0x80 0x21 0xB9 0xD4 0xEC 0xDF 0xA9 0x47 0xAA
0xFE 0xDB 0xBC 0x9C 0x0E 0xB6 0xD3 0x1F 0x56 0xA6 0xEA 0xF4 0xAB 0x74 0x03 0xFC
0x01 0xC1 0x8C 0x0D 0x85 0x7E 0x6E 0x23 0x67 0x8B 0xC9 0xAD 0x42 0xA7 0xD1 0x9E
0x75 0x46 0xDE 0x77 0xAF 0xAE 0xC8 0x51 0x11 0x92 0x2E 0xE8 0x43 0x57 0xD6 0xE6
0x33 0xB5 0x20 0x73 0x87 0x37 0x48 0x9D 0x7A 0x2D 0x16 0x0A 0x4F 0x0F 0x32 0xBB
0x66 0x53 0x29 0xD8 0xC2 0xE5 0x58 0x0C 0xE3 0x39 0x7D 0x13 0x36 0xBE 0x7F 0x5E
0xDC 0xC5 0x90 0xF2 0x55 0xD0 0x49 0xBD 0xA3 0x10 0x98 0xF3 0xEF 0x60 0x6A 0x24
0x40 0xF5 0xA1 0x2F 0x61 0x96 0x71 0xD5 0x38 0x82 0x64 0x1B 0xA5 0xD7 0xC3 0x8F
0x14 0x3A 0x6C 0x34 0x4A 0xDA 0x45 0x02 0x99 0x91 0x05 0x26 0xCD 0xCB 0x41 0xEB
0xCC 0xF7 0xAC 0x50 0x79 0x1C 0xCA 0xF6 0x8E 0xE4 0x00 0x5B 0x08 0x68 0xDD 0x9B
0xFF 0x4B 0x89 0x28 0xA8 0x76 0x9A 0xC4 0x2A 0x4C 0x9F 0x54 0xA4 0xE7 0x70 0xED}