Systematic and Random Searches for Compact 4-Bit and 8-Bit Cryptographic S-Boxes

Christophe Clavier, Léo Reynaud

Systematic and Random Searches for Compact 4-Bit and 8-Bit Cryptographic S-Boxes ⋆ ⋆⋆

Christophe Clavier and Léo Reynaud

University of Limoges, France

Abstract. Obtaining compact, while cryptographically strong, S-boxes is a challenging task required for hardware implementations of lightweight cryptography. Contrarily to 4-bit permutations design which is somewhat well understood, 8-bit permutations have mainly been investigated only through structured S-boxes built from 4-bit ones by means of Feistel, MISTY or SPN schemes. In this paper, we depart from this common habit and search for compact designs directly in the space of 8-bit permutations. We propose two methods for searching good and compact 8-bit S-boxes. One is derived from an adaptation to 8-bit circuits of a systematic bottom-up exploration already used in previous works for 4-bit permutations. The other is the use of a genetic algorithm that samples solutions in the 8-bit permutations space and makes them evolve toward predefined criteria. Contrarily to similar previous attempts, we chose to encode permutations by their circuits rather than by their tables, which allows to optimize non only w.r.t the cryptographic quality but also w.r.t. compactness. We obtain results which show competitive compared to structured designs and we provide an overview of the relation between quality and compactness in the range of rather small 8-bit circuits.
Beside, we also exhibit a 8-gate circuit made of only AND and XOR gates that represents a 4-bit permutation belonging to an optimal equivalence class. This shows that such optimal class can be instantiated by threshold implementation friendly circuits with no extra cost compared to previous works.

Keywords: S-box design · Hardware implementation · Threshold implementation · Genetic algorithms.

1 Introduction

The study presented in this paper has been initiated while designing the 8-bit S-box of Lilliput-TBC tweakable block cipher submitted to the NIST Lightweight Cryptography Standardization Process [18]. This 8-bit permutation has been defined as a 3-round Feistel construction based on three 4-bit functions S^1, S^2,

⋆ The research work presented in this paper has been supported by PACLIDO French funded research project.

⋆⋆ Computations needed for this research work have been realized on the CALI supercomputer funded by University of Limoges and XLIM, IPAM and GEIST institutes.

S^3. While outer round functions S^1 and S^3 are Almost Perfect Nonlinear (APN) functions borrowed from the SCREAM cipher [10], we focused on optimizing the choice of the inner round permutation S^2.

For finding compact 4-bit permutations with good cryptographic properties we chose to explore circuits systematically in a bottom-up manner by starting from the empty circuit which corresponds to the identity function, and adding gates successively in an optimized breadth-first search. Doing so, because all circuits are exhausted for increasing numbers of gates, when one finds a circuit that encodes a permutation with specific properties, then this circuit is the most compact to achieve these criteria. While this strategy has been used in previous works [21, 8], we decided to allow only gates from the restricted set {AND, XOR}. This choice corresponds to Boolean functions that are easy to share when one wants to derive a Threshold Implementation (TI) of the circuit for side-channel security. In other words, our search is directed toward finding good permutations whose associated circuits are both as most compact as possible (for a given quality and with this given set of gates) and TI-friendly for masked hardware implementations. These properties also give benefits in software if one choose to implement our S-boxes in the bit-slice manner: the reduced number of gates makes the execution time smaller, and the TI-friendly design remains a desirable property for software masked implementations also.

During our systematic search for compact 4-bit S-boxes we noticed that in most interesting circuits an AND gate was quite often followed by a XOR taking as one of its inputs the output of the AND. While we could not give any explanation for this, we thought that this structure may favor the quality/compactness goal. We thus defined a new "AND-XOR" three-input composite gate and decided to extend our search to 8-bit circuits with only gates of types AND-XOR and XOR.

Contrarily to the 4-bit case, the huge number of 8-bit permutations ($256! \approx 2^{1684}$) prevents our systematic approach to reach sufficiently long circuits to obtain interesting permutations. While finding good S-boxes directly in the set of 8-bit permutations is considered as a difficult task – it is commonly preferred to construct 8-bit permutations from 4-bit functions by means of Feistel, MISTY, Lai-Massey or SPN schemes – we still decided to undertake two kinds of guided random searches for 8-bit permutations. The first approach was to modify the systematic search tool that we used for 4-bit circuits: apart from redefining the set of gates as explained above and the number of bits to 8, we also left apart the systematic breadth-first approach and adopted a (still bottom-up) random depth-first search. The second approach was to use a genetic algorithm (GA) based on our same encoding of circuits as an ordered list of gates.

Previous Work As the S-box is the only non linear part of most encryption algorithms, finding permutations with good cryptographic properties has been widely studied. The three main principles of building such permutations are the random generation, the algebraic and the heuristic constructions. The first one fails to give good results as the exploration space is wide and suitable cryptographic properties very scarce [19, Table 9.2]. The second principle consists in finding

expressions leading to good properties. The most illustrative design concerns inversion in the finite field with the S-box of AES being a famous example. The third one uses guided search in order to evolve permutations to find even better ones. It involves notably genetic algorithms principle (see [13] for an overview on evolutionary techniques) and other techniques such as hill climbing [16], gradient descent [12] and simulated annealing [9]. All those techniques aim to find as good as possibles permutations without considering their implementations. From an hardware point of view, in order to reduce the implementation cost, these techniques must be followed by an optimization phase which tries to find a cheap implementation of a particular table. Lot of work has been done in this path, particularly to be applied to the AES S-box (e.g. [7, 5]). While there exist some works that optimize and prove the optimality on 4-bit circuits [11, 20], these techniques do not find the smallest circuit in general.
A different approach to fulfill compactness of a circuit is to build small circuits that instantiate S-boxes with sufficient cryptographic properties. Ullrich et al. [21] explore systematically all 4-bit circuits of a given number of gates before increasing this number, finding the optimal circuit for some classes of equivalence [6, 14] with their set of gates. In order to reduce the size, Canteaut et al. [8] build 8-bit S-boxes from optimal 4-bit ones using results of [21] in structures like Feistel or MISTY, until they obtain satisfactory results. Other works [3, 4] mainly reduce masking costs of such small S-boxes by decomposing cubic permutations into quadratic ones.

Our Contribution Inspired by the work of Ullrich et al. [21] we developed our own tool that builds 4-bit circuits by adding successive instructions (gates) to the empty circuit. While they used {AND, OR, XOR, NOT, MOV} as their set of instructions, we restricted our search to use only AND and XOR[1]. One can wonder whether this reduced set of instructions still allows to find a representative of an optimal class with as few instructions as them. Actually we answer positively to this question by exhibiting circuits belonging to the same optimal class with the same smallest gate count. Our first contribution thus shows that the TI-friendly property can be added to the more compact optimal 4-bit S-boxes with no penalty in term of circuit area for unmasked hardware implementations. Besides, our main contribution are two kinds of intensive search for compact circuits directly in the 8-bit permutations space. This led to a list of best S-box qualities that we can reach, ordered by the number of gates of their circuit. This list – which helps the S-box designer to answer the question *What cryptographic quality can I expect for this given number of gates?* – is provided in Table 1. These "records" are not absolute (some qualities may well be reached by smaller circuits in the future) but as far as we know this is the first attempt to provide

[1] Actually, we also implicitly allow the MOV instruction, but we do not count it for 1 gate since this can been seen as a free wiring in hardware implementations.
Regarding the NOT instruction, similarly to [21], we observed that allowing it does not result in smaller circuits. If the fixed point 0 is a concern, one can just add a NOT gate at the beginning or at the end of the circuit at a quite small cost (even negligible for threshold implementations).

a reference status of such a compactness-oriented search. It gives a large picture of the relation between quality and compactness in the range of rather small circuits for 8-bit permutations. Regarding our search by genetic algorithm, we encoded permutations by their hardware description. To the best of our knowledge, this is the first work that uses this encoding. Beside being gate count aware, it preserves structural portions of the circuit and seems to us more relevant than the encoding by table content of previous works.

Outline The paper is organized as follows. We define our cryptographic criteria, affine equivalence, and give background on threshold implementations in Section 2. Then we present our different searches for good and compact S-boxes: the systematic bottom-up search for 4-bit S-boxes is presented in Section 3, while Section 4 presents both the random bottom-up search and the GA-based search for 8-bit S-boxes, as well as a comparison with already known S-boxes. Finally Section 5 concludes this paper.

2 Preliminaries

2.1 Cryptographic Criteria to Optimize

Apart from optimizing our circuits with respect to their gate count, we also target good cryptographic quality based on the following criteria:

Algebraic Degree Given a Boolean function f defined on $\mathbb{F}_2^n$, its algebraic degree is defined as the maximal degree of the terms of its Algebraic Normal Form (ANF). The algebraic degree d of a permutation S of $\mathbb{F}_2^n$ is defined as the maximum algebraic degree of all its n component functions. We want to maximize d in order to better resist to algebraic or structural attacks.

Differential Uniformity The differential uniformity δ of a permutation S is defined as:

$$\delta(S) = \max_{a \neq 0, b} |\delta_S(a, b)|$$

where

$$\delta_S(a, b) = \#\{x \in \mathbb{F}_2^n : S(x \oplus a) = S(x) \oplus b\}$$

is related to the probability that an input difference a gives an output difference b. As we want to minimize this probability in the worst case (w.r.t. a and b) in order to make differential cryptanalysis [1] more difficult, we want to minimize $\delta(S)$.

Linearity The linearity $\mathcal{L}$ of a permutation S is defined as:

$$\mathcal{L}(S) = \max_{a, b \neq 0} |W_S(a, b)|$$

where the Walsh transform

$$W_S(a, b) = \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x \oplus b \cdot S(x)}$$

is related to the quality of the linear approximation $f_{a,b}(x) = a \cdot x \oplus b \cdot S(x)$. The higher $|W_S(a,b)|$ the better the approximation. As we want to minimize the correlation in the worst case (w.r.t. a and b) in order to make linear cryptanalysis [15] more difficult, we also want to minimize $\mathcal{L}(S)$.

2.2 Affine Equivalence Classes

Two S-boxes S_1 and S_2 are said affine equivalent if there exist two invertible linear mappings A and B and two constants a and b such that

$$S_1(x) = B \cdot (S_2(A \cdot x \oplus a) \oplus b) \quad \forall x \in \mathbb{F}_2^n \,.$$

This defines affine equivalence classes which preserve the algebraic degree, the differential uniformity and the linearity coefficient of S-boxes. De Cannière [6] has classified all 4-bit permutations in 302 classes. Leander et al. [14] define an optimal S-box as one that has optimal resistance against differential and linear cryptanalysis (minimal values of δ and $\mathcal{L}$). There exist 16 optimal classes of 4-bit S-boxes which correspond to the 16 first classes of [6]. In the sequel we borrow the notation of [3] for affine equivalence classes $(\mathcal{A}_i, \mathcal{Q}_j, \mathcal{C}_k)$ which gives the information whether the class contains affine, quadratic or cubic functions

2.3 Background on Threshold Implementations

Threshold implementation is a form of countermeasure aiming to prevent leakages provoked mainly by glitches in a hardware implementation of an algorithm [17, 2]. It consists in sharing variables and computations like in multiparty computation. A TI implementation of a function $F(x) = a$ is done as follow: the variable x is split into s_{in} input shares thanks to Boolean masking resulting in the sharing $\mathbf{x} = \{x_1, \ldots, x_{s_{in}}\}$ such that $x = \sum_{i=1}^{s_{in}} x_i$. Then, s_{out} component functions $F_j(x_1, \ldots, x_{s_{in}})$ are calculated for $j = 1, \ldots, s_{out}$, each one giving an output share such that $\mathbf{a} = \{F_1(\mathbf{x}), \ldots, F_{s_{out}}(\mathbf{x})\}$. A TI implementation must fulfill the three following properties:

- Correctness: the XOR of output shares gives the intended result $\sum_{i=1}^{s_{out}} F_i(\mathbf{x}) = F(x)$,
- Non completeness: each component function should be independent of at least one input share in order to provide first-order security,
- Uniformity: for a permutation and $s_{in} = s_{out}$, then each sharing $\mathbf{a}$ is given by exactly one sharing $\mathbf{x}$ through functions F_j.

The first property ensures that the result is correct and the second that no attacker is able to retrieve information about any variable when observing an output share. Both properties are relatively easy to implement. The last property is more difficult to fulfill but is only needed if other protected calculations use those outputs. Uniformity of sharing of input variables is mandatory, then if two calculations are done in a row, the first one must be uniform on its outputs

in order for the second to have uniformity on its inputs. It may require to rearrange each output share without breaking the first two properties, or use fresh randomness.
Notice that the TI of a function of degree d needs at least $d+1$ shares in order to fulfill these three properties.

3 Systematic Search for Compact 4-Bit S-Boxes

We describe here the study and development of a tool dedicated to the finding of compact 4-bit S-boxes with good cryptographic properties. Beside its cryptographic quality, we also require our selected S-box to provide an easy and efficient threshold implementation for side-channel security.

This search has been done by exploring 4-bit circuits with a small number of gates in a systematic way, and selecting some that satisfy some given criteria to use them as part of a Feistel structure. The resulting candidates for the LILLIPUT-TBC 8-bit S-box were checked whether they have suitable cryptographic properties. We focused on 4-bit permutations that belong to optimal classes since these are the only ones that reach $\delta = 4$ [2].

Our systematic bottom-up approach is very similar to that described in [21]. Starting from the empty circuit that encodes identity function, we progressively add gates to build more complex circuits. Since we investigate all possible ways of adding a gate before adding another one – that means exhausting all kinds of gates, and all their input and output bits –, we thus explore a tree in a breadth-first way where each node corresponds to a circuit and each level to its number of gates. This method implies that each equivalence class is found with the least gate count with certainty. Note that even if a particular representative of a class is found with a given number of gates, it is not proven that any other representative could be expressed with the same amount of gates.

During the exploration, each node may have up to $N_g * \binom{N_r}{2} * N_r$ children, where N_g is the number of gate types (in our case $N_g = 2$) and N_r is the number of registers[3]. To contain the exponential grows of the exploration, we use several tricks that either forbid some ways of adding gates, or prevent some nodes from further exploration. For instance, a gate should be added only if the following conditions are satisfied:

- its output should not overwrite a register that has not yet been used as input of the current or of a previous gate,
- the result of a gate should not be 0,
- a gate should change the value of its output register.

[2] According to [8] an 8-bit S-box derived from a 3-round Feistel construction can reach $\delta = 8$ only if both outer-round functions are APN and the inner-round function is a 4-bit permutation with differential uniformity equal to 4.

[3] The notion of register is particularly relevant for software implementations. Though, as in [21] we decided to use $N_r = 5$.

Also, we take care of the bit-permutation equivalence of circuits. We define that two circuits C_1 and C_2, represented by tables T_1 and T_2, are bit-permutation equivalent if there exist a permutation P_1 of the input bits and a permutation P_2 of the output bits such that

$$T_1(x) = P_2 \circ T_2 \circ P_1(x) \quad \forall x \in \mathbb{F}_2^n \,.$$

We keep only one representative circuit of each class of bit-permutation equivalence. To this end, we managed to define a canonical index of a circuit such that two circuits share the same index if and only if they are bit-permutation equivalent. Then when a circuit is considered, we compute its index and decide to not include it to the list of to-be-further-explored circuits if this index has already been encountered.

Results We explored up to the 8-gate level with 5 registers with this approach, resulting in a list of smallest circuits (up to the bit-permutation equivalence). We then only kept circuits resulting in a permutation. As the bit-permutation equivalence clearly does not change the affine equivalence class, we were able to find all circuits of smallest number of gates for some affine equivalence classes. In the limit of 8 gates, we encountered 62 affine equivalence classes among which all quadratic ones as well as three optimal classes – namely $\mathcal{C}_{223}$, $\mathcal{C}_{296}$ and $\mathcal{C}_{297}$ – from one of which we identified a satisfactory permutation.
As we obtained many 8-gate circuits belonging to the above three optimal classes, we decided to select one that is particularly suited for threshold implementation (beside producing a good quality 8-bit S-box). We used the same trick as in [3] to decompose a cubic permutation into two quadratics so that we can implement TI with only 3 shares. From their composition table, four optimal classes – namely $\mathcal{C}_{223}, \mathcal{C}_{266}, \mathcal{C}_{296}$ and $\mathcal{C}_{297}$ – can result from the composition of two quadratics. We hence decided to search representatives of the 6 quadratic classes with the least amount of gates to compose them and see if any optimal S-box can be built. As a result $\mathcal{Q}_4$ is very small as it needs only 2 gates. $\mathcal{Q}_{12}$ and $\mathcal{Q}_{294}$ come next with 4 gates, and $\mathcal{Q}_{293}$, $\mathcal{Q}_{299}$ and $\mathcal{Q}_{300}$ need 6 gates. A composition of two of them would result in either 4, 6, 8, 10 or 12 gates. Looking at their composition table, no optimal class can result from compositions with as few as 4 or 6 gates. All four optimal classes need a minimum of 8 gates.

Another criterion to minimize the number of gates of the TI is to make sure that the quadratic circuits have a direct uniform sharing. This particular sharing is the simplest and does not require additional gates or randomness unlike when needing correction terms or re-masking. From [3] only $\mathcal{Q}_4$, $\mathcal{Q}_{294}$ and $\mathcal{Q}_{299}$ can be directly shared. Using both previous observations, we composed 4-gate circuits of the class $\mathcal{Q}_{294}$ with no permutation in between to avoid extra gates. We obtained several permutations of the class $\mathcal{C}_{223}$, and among all these solutions we had the opportunity to choose a permutation that results from the same quadratic circuit used twice (see Figure 1).

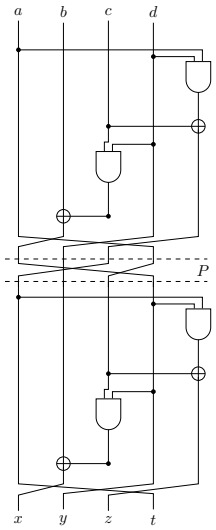

Fig. 1. The inner 4-bit S-box component of Lilliput-TBC

As the search is exhaustive, there is no smaller circuit for each quadratic class. We used two circuits of 4 gates which is the minimum according to the composition table, and each one can be directly uniform shared resulting in the minimum of additional cost for the thresholding. Note that 8 gates is the minimum that we found for an optimal class without composition, $\mathcal{C}_{223}$ being one of them. Referring to [21], they also reach this class with 8 gates (they count 9 as they include the MOV instruction) with a larger set of gates[4]. We thus demonstrate that their exist TI-friendly representatives of this $\mathcal{C}_{223}$ record class that do not need any extra gate.

Breadth-First Search with Level As we did not find all 302 classes with this method due to memory issues brought by the size of the heap of circuits to be explored, we chose to split the exploration on several parallel processes. This time, each process would not start the exploration with the empty circuit, but rather with an 8-gate portion of circuit, hence reducing the size of the heap to explore. This results in an increased number of circuits explored as these processes are independent and run in parallel. As a drawback, we lose some early-abort opportunities as circuits already explored in other parallel process are not detected. Nevertheless, this method allowed us to go as far as 10 gates (not exhaustively) and to find more classes including optimal ones.

[4] Their exemplary record circuit includes an OR gate.

4 Random Searches for Compact 8-Bit S-Boxes

Low implementation cost of 8-bit S-boxes is an important matter in lightweight cryptography. A number of already used implementations are based on Feistel, MISTY or SPN structures [8, 4]. Here we try to find interesting permutations without these structures, directly in the 8-bit permutations space. We also keep the TI-friendly set of gates. Unlike for 4-bit circuits, it is unfeasible to systematically explore 8-bit ones, even for a few gates. We therefore tried two kinds of random explorations, a bottom-up and a genetic oriented one.
Before describing those explorations, we introduce a particular gate setting that allows to speed up the search. A problem of generating circuits by adding random gates is that a very low proportion of them are permutations. This becomes particularly noticeable when changing from 4-bit to 8-bit circuits. Thanks to some observations from our 4-bit exploration, and from some S-box schemes presented by other authors, we have noticed that AND and OR gates are very likely to be followed by a XOR in low cost implementations. We thus chose to explore circuits using mostly a so-called "AND-XOR" gate that simply consists in a AND followed by a XOR. This composite gate takes three inputs – two for the AND and one for the XOR as its second input is the AND output – and gives one output. An interesting observation is that if the third input register is the same as the output register, then such a gate preserves the permutation property[5] (see Figure 2). As this is also trivially the case for the XOR, we thus have a way to systematically explore into the permutations space. A drawback of this trick is that the set of reachable permutations is reduced, but this is largely compensated by the advantage that we do not have to waste most of computation time to consider non-permutation circuits.

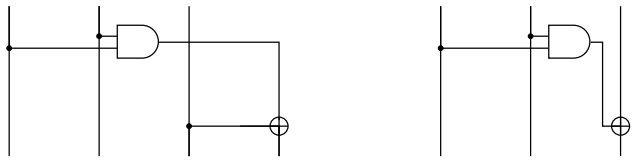

Fig. 2. Two AND-XOR composite gates: ordinary (left), preserving the permutation property (right)

4.1 Random Bottom-Up Search

We adapted the search tool described in Section 3. To this end, beside increasing the number of bits from 4 to 8, we changed the gate set to {AND-XOR, XOR}. Also, the search is no more a breadth-first systematic exploration, but an in-depth random process. It consists in repeating a huge number of times the

[5] If we add such a gate to a circuit that encodes a permutation, then the resulting circuit also encodes a permutation.

generation of a random circuit, still in the bottom-up way by adding successive gates randomly. Notice that since the AND-XOR gate already includes a XOR, we put basic XOR gates only with a marginal probability.

We randomly built circuits using eight registers[6]. Each time we add a gate, cryptographic properties are evaluated and a list of best found figures is maintained for each gate count. After some tries, we noticed that the probability that a circuit gives good cryptographic results vanishes as the number of gates grows. More precisely, consider two generated circuits of the same size, one of low quality and one of higher quality. Then if we add a same few number of random gates to each of them, it is unlikely that the first resulting circuit becomes better than the second one. This led us to proceed by stages. First we generate from scratch many circuits of some size n_1. We then select the better of them, and we pursue by adding gates to these promising circuits up to size $n_2 > n_1$. We can continue this way by progressively adding gates stage by stage.

Indeed, this strategy gives rather better results than generating long circuits from the beginning in one shot. We had to make many runs, and try different ways of selecting circuits, as well as different number of stages and stage levels. After several weeks of computations on dozens of independent computing nodes, we have obtained "record" circuits ranging from about 10 to 50 gates that are presented in Table 1 in the "random search" column. For each gate count, the cell gives the best cryptographic quality we obtained – in terms of differential uniformity (δ) and linearity ($\mathcal{L}$)[7]. For some sizes, we obtained several record circuits that are not comparable to each others. This happens when one circuit is better for some criteria while the other is better for the other criteria. In such cases, we provide all these non-comparable records.

4.2 Genetic Algorithm Search

Genetic algorithms (GA) are heuristic methods that mimic biologic processes of the evolution of life. They aim at finding as good solutions as possible to non-linear and possibly multi-dimensional problems that are not possible to solve by deterministic methods. A genetic algorithm makes evolving a population of individuals represented by chromosomes which encode a particular solution to the problem to optimize. It typically comprises the following phases :

- Generate a random population of solutions
- Randomly draw couples of individuals based on their fitness (quality) and create children by crossing parts of their parents (and eventually transform these children back into solutions if it is not the case)
- Randomly mutate children
- Mimic natural selection by preferentially selecting better solutions from the pool of parents and children and "killing" low fitness individuals

[6] Actually, using the AND-XOR gate, an extra register is implicitly required to temporarily store the AND output.

[7] It happened that our best circuits (w.r.t. δ and $\mathcal{L}$) always had maximal algebraic degree (except for quite small ones).

– Iterate until a sufficiently good solution is found

Genetic algorithms rely on the fact that crossing parts of medium or good quality individuals may give birth to a better solution by mixing together interesting parts of each parent. The mean quality of each generation is thus expected to increase over time, at least during the first generations.

A particularly important aspects of a genetic algorithm is the way chromosomes encode solutions to the problem to solve. Previous uses of GA [13] were made to search good S-boxes but only focused on their cryptographic properties and did not considered the underlying circuit. On the contrary, we have chosen to encode S-boxes by their circuit rather than by their representative table. In our case, a chromosome is thus the ordered list of gates that defines the circuit. We see two advantages to proceed this way: First, it is possible to take into account the size of the circuit in the fitness of individuals, which allows our search to be compactness oriented. Also, there may exist structural aspects of the circuit which make sense from a cryptographic point of view. These will be preserved in most cases from one generation to the next.

As solutions are encoded by circuits that give permutations, the crossing over step of two individuals is simply done by swapping parts of their circuits. Actually, the main issue of applying GA principles to circuits is that the mating of two parents that encode permutations, may well result in a child that is no more a permutation. In this case we have to find a way to transform it back into a permutation. Fortunately we did not have to mind with this issue thanks to the use of our special kind of AND-XOR gates with particular output (XOR gate with result in place) as swapping parts of two circuits always ends up in permutations.

We present the main options and parameters that we used in our GA search for permutations.

First generation The first generation is generated by random circuits of a given length. As an option, we can also insert some good solutions found from previous executions (or from circuits found with the random search).

Fitness The definition of the fitness is also an important feature of a genetic algorithm. In our case, this was not straightforward as we had to solve a multi-criteria optimization: we want to minimize the number of gates, the differential uniformity and the linearity, and to maximize the algebraic degree. We ended up using two types of fitness computation. The first is based on the very simple weighted sum of those criteria. The second consists in ranking individuals with a derivative of dominance, and then giving an equal score to all individuals of a same rank. Our derivative of dominance is defined as follow: given two individuals I_1 and I_2 and their sets of n respective criteria $\{c_1^1, \ldots, c_1^n\}$ and $\{c_2^1, \ldots, c_2^n\}$, we consider that I_1 dominates I_2 if $\#\{i : c_1^i > c_2^i\} > \frac{n}{2}$, that is the number of criteria for which I_1 is better to I_2 is more than the half. Then individuals of rank 1 are defined as those who are dominated by no others. Individuals of rank 2 are those who are dominated by no other which are not of rank 1, and so on.

Selection of parents Parents are drawn thanks to two techniques. The first one is the roulette where each individual is being attributed a probability proportional to its fitness. Parents are then drawn repeatedly according to this probability law, with or without the possibility that an individual may be parent several times. The second consists in drawing uniformly two individuals, selecting the one with best fitness to be parent, and repeat until enough parents are obtained.

Crossover The crossover is done by randomly splitting circuits in three parts. Both intermediate parts are then swapped to give two children. Note that we require they have same length, but not necessarily same starting index (position in the circuit description).

Mutations Mutations are done with small probability. Four types of mutations are considered. The first one adds a random gate at a random indice in the circuit. The second removes a random gate. The third replaces a gate by a random one. The last swaps two consecutive gates if this does not change the resulting permutation.

Next generation We use an elitist selection as the next generation is composed of the best individuals from both current generation and children. Each generation keeps the same number of individuals.

Diversity In order to keep the diversity of the population, we sometimes remove identical individuals, and can incorporate random individuals to be part of the next generation.

Results We searched by using many combinations of these options and parameters. We have made many tries and observed that like with the random bottom-up search it was a good practice to use previously found good solutions. This was done by joining some of them to random individuals for the first generation. Overall we have dedicated a bit less computation time to the genetic algorithm search than to the random bottom-up one. Though, interesting solutions have also been found as depicted in the column "genetic search" of Table 1. Here also each cell contains the best qualities (couples δ-$\mathcal{L}$) that have been reached for this number of gates. We also highlighted in bold those qualities that have been found by one method while the other did not reached it for the same number of gates. As we can see results are globally about the same for both methods. Though, genetic algorithms are sightly better in the range of medium size circuits (27 to 46 gates) while the random bottom-up search tends to show advantageous for smaller and larger ones.
Appendices A and B present two illustrative circuits alongside with their tables. They are the smallest ones we found that reach $\delta = 10$ and $\delta = 8$, with respectively 37 and 40 gates. As a curiosity, notice that the circuit of Appendix B is made of only AND-XOR gates.

# gates	algebraic degree	diff. uniformity (δ) - linearity ($\mathcal{L}$)	
		random search	genetic search
8	5	128 - 256	128 - 256
9	5	128 - 256	128 - 256
10	6	128 - 256	128 - 256
11	6	128 - 256	128 - 256
12	7	128 - 256	128 - 256
13	7	128 - 256	128 - 256
14	7	128 - 256	128 - 256
15	7	128 - 256	128 - 256
16	7	64 - 128	64 - 128
17	7	64 - 128	64 - 128
18	7	64 - 128	64 - 128
19	7	64 - 128	64 - 128
20	7	64 - 128	64 - 128
21	7	64 - 128	64 - 128
22	7	**48 - 128**	64 - 128
23	7	48 - 128	48 - 128
24	7	32 - 128	32 - 128
25	7	32 - 128	32 - 128
26	7	32 - 128	32 - 128
27	7	32 - 96	**32 - 68**
28	7	32 - 76	**32 - 64**
29	7	16 - 128 20 - 76 32 - 68	**16 - 76** **24 - 68** **32 - 64**
30	7	16 - 76 18 - 72 32 - 64	**16 - 64**
31	7	16 - 72 20 - 68 32 - 64	**16 - 64**
32	7	16 - 64	16 - 64
33	7	16 - 64	16 - 64
34	7	16 - 64	16 - 64
35	7	16 - 64	16 - 64
36	7	12 - 64	12 - 64
37	7	12 - 64	**10 - 80** 12 - 64
38	7	10 - 68 12 - 64	**10 - 64**
39	7	10 - 64	10 - 64
40	7	10 - 64	**8 - 64**
41	7	10 - 64	**8 - 64**
42	7	10 - 64	**8 - 64**
43	7	10 - 64	**8 - 64**
44	7	10 - 64	**8 - 64**
45	7	**10 - 60**	**8 - 64**
46	7	**10 - 60**	**8 - 64**
47	7	8 - 64 **10 - 60**	8 - 64 12 - 60
48	7	8 - 64 10 - 60	8 - 64 10 - 60
...	...	...	...
51	7	**10 - 56**	8 - 64 10 - 60
...	...	...	...
54	7	**8 - 60**	8 - 64 10 - 56
...	...	...	...
57	7	8 - 60	8 - 60

Table 1. Best 8-bit S-boxes w.r.t. circuit size found by random and genetic searches

4.3 Comparison with Known S-Boxes

We refer to the S-box quality as the vector $(d,\delta,\mathcal{L})$. We only compare to relevant S-boxes, discarding qualities that we did not reached, or S-boxes with very low degree.

Canteaut et al. [8] built an 8-bit S-box with quality (6,8,64) in 38 gates, counting 3×4 XOR for the 3-round Feistel structure. We found a nearly equivalent solution (see Appendix B) that provides the optimal algebraic degree $d = 7$ at the cost of only two extra gates. Note that they particularly take into account the number of non-linear gates, whereas we have an equal number of AND and XOR gates.

Compared to the S-box proposed for Lilliput-TBC the picture is about the same. It is also a 3-round Feistel that achieves the same quality as [8]. It requires 39 gates but provides an optimized TI-oriented design. Again our S-box of Appendix B gives maximal degree for about the same size.

Referring to Table 1 of [8], we can also compare to S-boxes of Robin and Fantomas. These ciphers use (6,16,64) S-boxes that cost 36 gates while we achieve (7,16,64) with 30 gates.

In [4], Boss et al. propose several structured S-boxes. Unfortunately, they provide area figures rather than gate counts. We can somehow still manage to compare with some educated guesses on their raw implementations. As we found optimal circuits for quadratic 4-bit permutations, we will use them to estimate their circuit sizes. $\mathbf{SB}_1$ uses 8 iterations of $\mathcal{Q}_{294}$ which can be built with 4 gates. It ends up requiring roughly 32 gates to achieve (6,16,64), while we obtain (7,16,64) with 30 gates only. $\mathbf{SB}_3$ uses $\mathcal{Q}_{293}$, $\mathcal{Q}_{299}$ and a matrix multiplication in a SPN like structure with 4 iterations. Only counting the two quadratics, both achievable in 6 gates, this results in more than 48 gates for a (7,8,60) permutation. We obtain the same quality with 54 gates. For $\mathbf{SB}_6$, that has quality (7,10,60), they use the same structure with 4 iterations. Only counting the quadratics $\mathcal{Q}_{293}$ and $\mathcal{Q}_{294}$, it ends up with more than 40 gates while we found the same quality with 45 ones. Again this is not counting the cost of the matrix multiplication. $\mathbf{SB}_5$ provides the same quality but without matrix multiplication, and uses 9 iterations of $\mathcal{Q}_4$ and $\mathcal{Q}_{294}$ which results in 54 gates, but on their Table 1 its area is less than that of $\mathbf{SB}_6$. We can guess that their multiplication is rather costly.

Notice that some of the above comparisons may not be totally fair. Indeed we have compared circuit sizes for unmasked implementations only. While our designs are TI-friendly (only AND and XOR gates), their size has not been optimized for a masked implementation. This sometimes results in circuits with more non-linear gates for a same gate count[8], which shows disadvantageous for protected implementations. Nevertheless we notice that in most cases, our results compare very well with – and are sometime better than – structured dedicated designs. We may expect that interesting results would have also been obtained if we had chosen to optimize the cost of a first-order protected TI implementation.

[8] Notably, S-box of Appendix B uses 20 non-linear gates compare to only 12 for the S-box of [8] or that of Lilliput-TBC.

5 Conclusion

We have implemented and studied different methods for searching good and compact 4-bit and 8-bit S-boxes.

For 4-bit S-boxes we adapted a systematic search that has been investigated in previous works [21, 8] by restricting the set of gates to only AND and XOR in order to obtain circuits for which it is easy to find a threshold implementation. We found TI-friendly circuits of an optimal class still with the same smallest number of gates. Such a circuit has been used in the design of the 8-bit permutation of Lilliput-TBC lightweight block cipher.

For 8-bit S-boxes we chose to explore in the whole space of permutations rather than to restrain ourselves to structured designs. Beside giving more opportunities to find good solutions, this allowed us to easily obtain permutations with maximal algebraic degree. We have presented a random search that builds circuits in the bottom-up manner, and a genetic algorithm designed to optimize both cryptographic quality and circuit size for unmasked implementations. It happens that both methods give results that are competitive with previously known S-boxes. While we optimized our searches for the compactness of unmasked implementations, we managed to generate only TI-friendly circuits. We think that an interesting future work will be to adapt the genetic search by modifying the definition of the fitness in order to optimize the cost of a circuit protected by threshold implementation.

We hope that this work will convince that searching for unstructured S-boxes may be a promising approach, and will motivate further works in this direction.

References

1. Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.
2. Begül Bilgin. *Threshold implementations: as countermeasure against higher-order differential power analysis.* PhD thesis, University of Twente, Netherlands, May 2015.
3. Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Natalia N. Tokareva, and Valeriya Vitkup. Threshold Implementations of Small S-Boxes. *Cryptography and Communications*, 7(1):3–33, 2015.
4. Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit Sboxes with Efficient Masking in Hardware (extended version). *J. Cryptographic Engineering*, 7(2):149–165, 2017.
5. Joan Boyar and René Peralta. New logic minimization techniques with applications to cryptology. *IACR Cryptology ePrint Archive*, 2009:191, 2009.
6. Christophe De Cannière. *Analysis and Design of Symmetric Encryption Algorithms.* PhD thesis, Katholieke Universiteit Leuven, May 2007.
7. David Canright. A Very Compact S-Box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

8. Anne Canteaut, Sébastien Duval, and Gaëtan Leurent. Construction of Lightweight S-Boxes using Feistel and MISTY structures (Full Version). *IACR Cryptology ePrint Archive*, 2015:711, 2015.
9. John A. Clark, Jeremy L. Jacob, and Susan Stepney. The Design of S-Boxes by Simulated Annealing. *New Generation Comput.*, 23(3):219–231, 2005.
10. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varıcı, François Durvaux, Lubos Gaspar, and Stéphanie Kerckhof. SCREAM v3, August 2015. Submission to the CAESAR competition.
11. Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing Implementations of Lightweight Building Blocks. *IACR Trans. Symmetric Cryptol.*, 2017(4):130–168, 2017.
12. Oleksandr Kazymyrov, Valentyna Kazymyrova, and Roman Oliynykov. A Method For Generation Of High-Nonlinear S-Boxes Based On Gradient Descent. *IACR Cryptology ePrint Archive*, 2013:578, 2013.
13. Karlo Knezevic. Combinatorial Optimization in Cryptography. In Petar Biljanovic, Marko Koricic, Karolj Skala, Tihana Galinac Grbac, Marina Cicin-Sain, Vlado Sruk, Slobodan Ribaric, Stjepan Gros, Boris Vrdoljak, Mladen Mauher, Edvard Tijan, and Filip Hormot, editors, *40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017, Opatija, Croatia, May 22-26, 2017*, pages 1324–1330. IEEE, 2017.
14. Gregor Leander and Axel Poschmann. On the Classification of 4 Bit S-Boxes. In Claude Carlet and Berk Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.
15. Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In *Advances in Cryptology - EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.
16. William Millan. How to Improve the Nonlinearity of Bijective S-Boxes. In Colin Boyd and Ed Dawson, editors, *Information Security and Privacy, Third Australasian Conference, ACISP'98, Brisbane, Queensland, Australia, July 1998, Proceedings*, volume 1438 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 1998.
17. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *Journal of Cryptology*, 24(2):292–321, Apr 2011.
18. National Institute of Standards and Technology. Lightweight Cryptography, January 2017. https://csrc.nist.gov/Projects/Lightweight-Cryptography.
19. Léo Perrin. *Cryptanalysis, Reverse-Engineering and Design of Symmetric Cryptographic Algorithms*. PhD thesis, University of Luxembourg, 2017.
20. Ko Stoffelen. Optimizing S-Box Implementations for Several Criteria Using SAT Solvers. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 140–160. Springer, 2016.
21. Markus Ullrich, Christophe De Canniere, Sebastiaan Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding Optimal Bitsliced Implementations of $4\times$ 4-bit S-Boxes. In *SKEW 2011 Symmetric Key Encryption Workshop, Copenhagen, Denmark*, pages 16–17, 2011.

A S-Box Reaching 7-10-80 Quality with 37 Gates

The 8-bit S-box of Table 2 reaches $\delta = 10$ and $\mathcal{L} = 80$ with a maximal algebraic degree $d = 7$. Figure 3 presents a 37-gate TI-friendly circuit of this S-box.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	63	02	11	06	BD	04	93	0B	09	08	5A	85	14	87	6F
10	30	13	52	21	DB	91	B8	DE	5B	39	38	0A	AD	2C	EE	16
20	20	01	64	FD	EF	07	C3	A2	6B	6A	AF	37	5E	BC	40	4E
30	31	10	ED	74	EA	92	7B	2A	1A	1B	47	FF	84	17	23	5C
40	44	BE	CC	27	22	43	E1	C2	CF	75	46	CD	A3	58	60	6D
50	54	CE	DC	77	7F	2E	49	3B	BF	65	36	DD	4A	C0	7C	41
60	EC	45	62	53	42	61	AC	A9	26	AE	28	19	C1	78	3D	86
70	55	FC	03	12	4D	3F	05	90	FE	76	29	18	E0	69	DF	15
80	99	A8	9A	1F	1E	E6	1D	9C	83	96	81	A0	8C	8A	8F	25
90	2D	1C	6E	EB	57	9F	35	A4	D7	E2	B5	94	B0	67	D2	89
A0	DA	88	3E	C4	A5	0F	4B	59	A1	D6	E5	BB	24	B1	F9	70
B0	3C	0E	D0	CA	56	8D	F5	B6	82	95	4F	F1	9D	98	E8	51
C0	7D	E4	C5	6C	D9	CB	2B	7A	C7	FB	7E	F2	68	D5	BA	50
D0	E9	F0	D1	F8	71	32	F6	B7	B3	2F	AA	A6	F4	79	73	C8
E0	A7	4C	B9	5F	FA	AB	E7	34	5D	B2	E3	80	48	F7	66	9E
F0	D8	F3	8B	0D	72	33	0C	8E	C6	C9	B4	97	3A	D4	D3	9B

Table 2. Table representation of an S-box with $d = 7$, $\delta = 10$ and $\mathcal{L} = 80$.

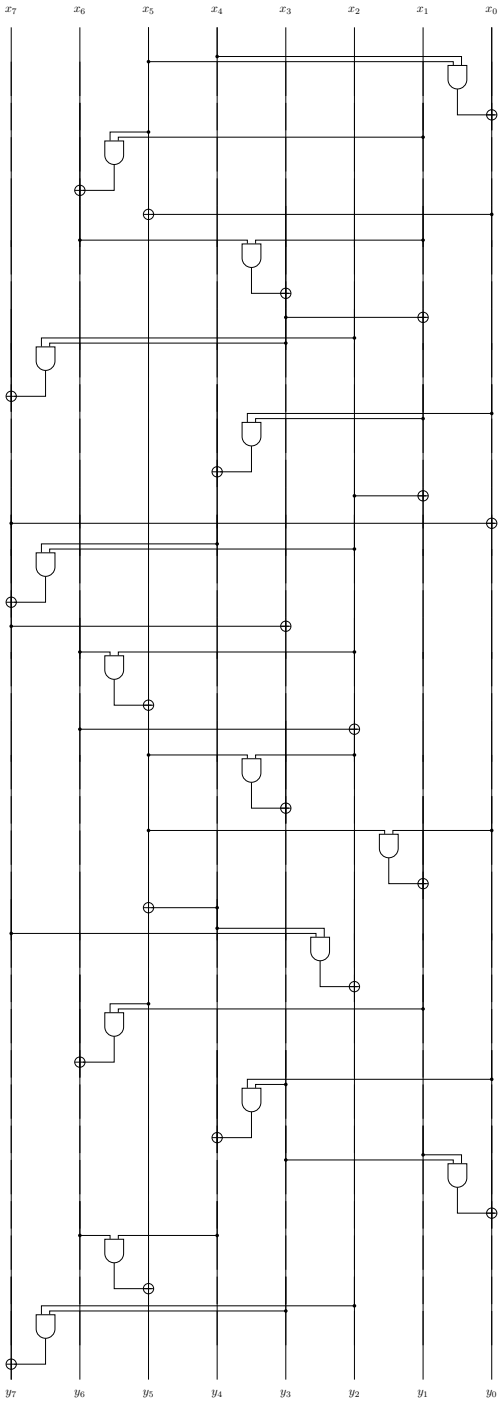

Fig. 3. A 37-gate TI-friendly circuit of the 7-10-80 S-box of Table 2

B S-Box Reaching 7-8-64 Quality with 40 Gates

The 8-bit S-box of Table 3 reaches $\delta = 8$ and $\mathcal{L} = 64$ with a maximal algebraic degree $d = 7$. Figure 4 presents a 40-gate TI-friendly circuit of this S-box.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	01	02	1A	04	05	A7	15	08	8C	0B	7B	0C	89	AE	D9
10	10	11	1B	03	BF	BE	14	A6	79	D4	77	2E	5B	D6	F4	8A
20	20	67	22	70	A5	4A	06	53	AD	CF	0F	99	28	63	2B	36
30	31	7F	3A	61	1F	FD	B4	EC	54	93	FA	C9	56	9C	58	C4
40	40	47	46	51	4C	EA	EB	F2	68	C3	6F	3C	E5	EF	42	B2
50	50	5E	5F	41	F3	5D	5C	4D	18	96	12	65	B6	B8	1C	E9
60	60	21	66	3B	ED	A4	4B	B5	44	A9	E2	F7	48	2D	4F	74
70	71	30	7E	23	52	1E	FC	07	BC	F8	16	AA	33	DA	39	8F
80	80	84	82	9F	85	81	26	B0	88	09	8B	76	8D	0D	2F	F5
90	B1	34	BA	27	9E	BB	35	83	D8	78	D7	0A	7A	5A	D5	AF
A0	A2	DC	A0	CA	87	DF	24	E6	8E	BD	2C	E3	AB	32	A8	4E
B0	9B	6D	90	72	95	E1	3E	D1	DB	45	75	17	F9	49	F6	38
C0	F1	D3	FF	CC	D2	F0	7D	C1	B9	13	B3	69	97	1D	3D	E4
D0	C0	6A	C7	7C	CD	C6	6B	FE	E8	6E	EE	19	64	43	C2	B7
E0	DE	86	D0	94	DD	A3	73	9A	37	0E	9D	55	98	2A	92	57
F0	E7	3F	E0	25	CB	91	6C	A1	62	FB	C5	AC	CE	59	C8	29

Table 3. Table representation of an S-box with $d = 7$, $\delta = 8$ and $\mathcal{L} = 64$.

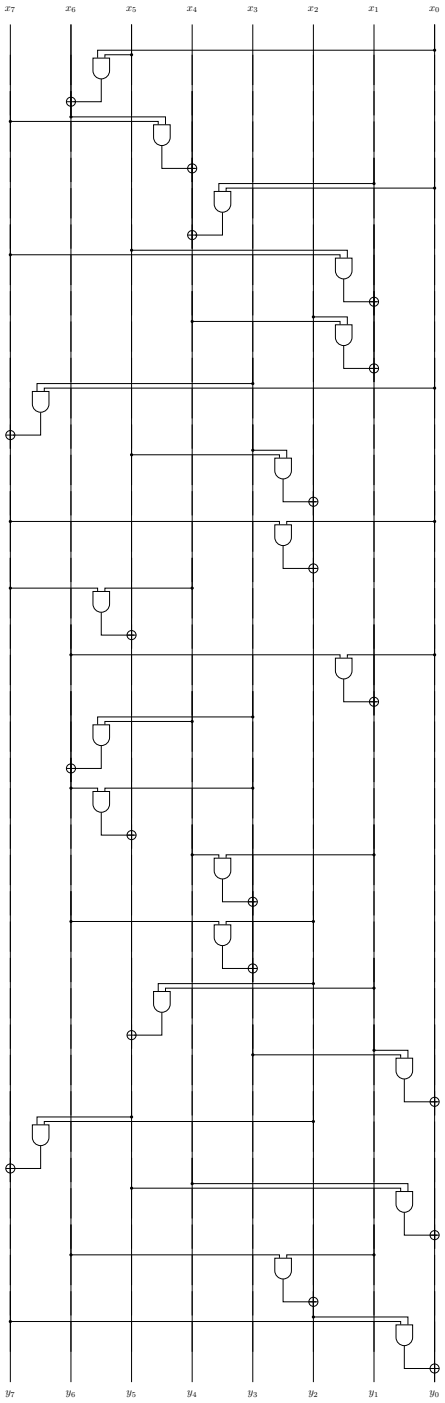

Fig. 4. A 40-gate TI-friendly circuit of the 7-8-64 S-box of Table 3