Cellular Automata Based S-boxes

Luca Mariot[1], Stjepan Picek[2], Alberto Leporati[1], and Domagoj Jakobovic[3]

[1]Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336, 20126 Milano, Italy ,
{luca.mariot, leporati}@disco.unimib.it

[2]Cyber Security Research Group, Delft University of Technology, Mekelweg 2, Delft, The Netherlands ,
stjepan@computer.org

[3]University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, 10000, Zagreb, Croatia ,
domagoj.jakobovic@fer.hr

February 17, 2018

Abstract

Cellular Automata (CA) represent an interesting approach to design Substitution Boxes (S-boxes) having good cryptographic properties and low implementation costs. From the cryptographic perspective, up to now there have been only ad-hoc studies about specific kinds of CA, the best known example being the χ nonlinear transformation used in Keccak. In this paper, we undertake a systematic investigation of the cryptographic properties of S-boxes defined by CA, proving some upper bounds on their nonlinearity and differential uniformity. Next, we extend some previous published results about the construction of CA-based S-boxes by means of a heuristic technique, namely Genetic Programming (GP). In particular, we propose a "reverse engineering" method based on De Bruijn graphs to determine whether a specific S-box is expressible through a single CA rule. Then, we use GP to assess if some CA-based S-box with optimal cryptographic properties can be described by a smaller CA. The results show that GP is able to find much smaller CA rules defining the same reference S-boxes up to size 7×7, suggesting that our method could be used to find more efficient representations of CA-based S-boxes for hardware implementations. Finally, we classify up to affine equivalence all 3×3 and 4×4 CA-based S-boxes.

Keywords Cellular Automata S-box Cryptographic properties Heuristics

1 Introduction

A frequent direction adopted in the design of block ciphers is the *Substitution-Permutation Network* (SPN) cipher. Such ciphers usually consist of an XOR operation with the key/subkeys, a linear layer, and a substitution layer [21]. To build the substitution layer, a common option in today's designs is to use one or more *Substitution Boxes* (S-boxes, also known as vectorial Boolean functions). In order for an S-box to be useful, it needs to fulfill a number of cryptographic properties. In his seminal work on the design of block ciphers, Shannon introduced the concept of *confusion* that an S-box needs to have [36]. Here, confusion can be defined as the property that the ciphertext statistics should depend on the plaintext statistics in a manner too complicated to be exploited by an attacker. This concept is connected with the cryptographic property of *nonlinearity*. Finding an S-box that is resilient against various attacks is not easy and this problem becomes even more complicated if we consider various sizes of S-boxes that are of practical relevance. As examples, some common occurring S-box sizes are 4×4 (PRESENT [5]), 5×5 (Keccak [4]), and 8×8 (AES [18]). Note that the examples have the same input and output sizes as with the S-boxes we consider in this paper – i.e., mappings from n bits to n bits.

From the cryptographic properties perspective, the minimum set of criteria one would need to consider when designing S-boxes includes bijectivity, high nonlinearity, and low differential uniformity. To obtain such properties, there are several options to consider ranging from mathematical constructions to various heuristics. When discussing mathematical constructions, a typical choice is to use power mappings, as in the case of the AES S-box (where the inverse power function and an affine transformation are used). Conversely, in heuristic approaches the designer has at his disposal a number of techniques that in general cannot compete with mathematical constructions, but which can offer an interesting alternative in specific scenarios (see Section 6 for details).

In this paper, we focus on S-boxes constructed with *Cellular Automata* (CA). More precisely, a CA-based S-box can be considered as a particular type of vectorial Boolean function where each coordinate function corresponds to the CA rule applied on a local neighborhood. The best known example of such an S-box is the χ nonlinear transformation used in the *Keccak* sponge construction, which is now part of the SHA-3 standard [4]. There, the authors use a CA rule affecting only three neighborhood positions for each bit, which results in an extremely lightweight definition of the S-box with a small implementation cost, but which also yields suboptimal cryptographic properties. To the best of our knowledge, all the other ciphers using CA rules for defining S-boxes actually use that same rule. This is the case of Panama [15], RadioGatún [3], Subterranean [13], and 3Way [17] ciphers. Besides those S-boxes, there are also designs using an S-box that is an affine transformation of the Keccak S-box, such as Ascon [12].

This paper extends earlier work on the subject published by Picek et al. [30, 29]. In those papers, we covered the construction of CA-based S-boxes of different dimensions using a specific heuristic technique, namely *Genetic Programming*

(GP). We showed it is possible to construct optimal S-boxes with respect to the nonlinearity and differential uniformity properties (except for dimension 6×6, which is anyway not achievable with a single CA rule, see Section 5.2 for details). Beside that, those papers addressed the construction of CA-based S-boxes that are additionally optimized with respect to the area requirements. On the other hand, the main contributions of the present paper are the following:

1. We theoretically prove some upper bounds for the nonlinearity and differential uniformity properties of S-boxes constructed by CA. In particular, we relate those cryptographic properties to the corresponding properties of the underlying local rules. Interestingly, our findings also show why the CA used in Keccak cannot have a better nonlinearity and differential uniformity by adding more cells.

2. We present a "reverse-engineering" method that is able to find CA rules resulting in specific S-boxes. In this context, we address two main questions: the first one is whether a generic S-box can be expressed through a single CA rule, for which we devise a procedure based on the *De Bruijn graph representation* of CA. Next, given an S-box that can be represented with a single CA rule, the second question we address is whether there exists a shorter rule resulting in the same S-box. Our reverse engineering approach, which is still based on GP, shows that it is possible to obtain such shorter rules for all optimal CA-based S-boxes reported in [30, 29].

3. We conduct an exhaustive search for 3×3, 4×4, and 5×5 CA-based S-boxes. Further, we provide a complete classification up to affine equivalence for the 3×3 and 4×4 sizes.

The rest of the paper is organized as follows. In Section 2, we discuss necessary information about S-boxes and cryptographic properties we consider. Section 3 gives background on cellular automata and their connection with S-boxes. Section 4 gives theoretical results – specifically, upper bounds for nonlinearity and differential uniformity attainable by CA-based S-boxes. Section 5 offers experimental results where we investigate how to use heuristics to construct shorter rules for CA-based S-boxes. Additionally, we provide enumerations of affine classes for several S-box sizes and discuss possible future research directions. Section 6 gives a short overview of related work, both from the perspective of CA and of *Evolutionary Computation* (EC) approaches in the design of S-boxes. Finally, in Section 7 we summarize the main points of the paper.

2 Cryptographic Properties of S-boxes

Let n, m be positive integers, i.e., $n, m \in \mathbb{N}^+$. We denote by $\mathbb{F}_2^n$ the n-dimensional vector space over the finite field $\mathbb{F}_2$. Further, for any set S, we denote $S \setminus \{0\}$ by S^*. The usual inner product of $a, b \in \mathbb{F}_2^n$ equals $a \cdot b = \bigoplus_{i=1}^{n} a_i b_i$.

The *Hamming weight* $w_H(a)$ of a vector a, where $a \in \mathbb{F}_2^n$, is the number of non-zero positions in the vector. An (n, m)-function is any mapping F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. An (n, m)-function F can be defined as a vector $F = (f_1, \cdots, f_m)$, where the Boolean functions $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ for $i \in \{1, \cdots, m\}$ are called the *coordinate functions* of F. Given $v \in (\mathbb{F}_2^m)^*$, the *component function* $v \cdot F : \mathbb{F}_2^n \to \mathbb{F}_2$ is the Boolean function defined for all $x \in \mathbb{F}_2^n$ as the inner product between v and $F(x)$. In other words, the component functions of F represent the non-trivial linear combinations of its coordinate functions.

2.1 S-box Representations

A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a truth table (TT), which is a vector $\Omega_f \in \mathbb{F}_2^{2^n}$ defined as

$$\Omega_f = (f(0, \cdots, 0), \cdots, f(1, \cdots, 1)) \ .$$

More precisely, Ω_f contains the output values of f in lexicographical order with respect to the input entries, i.e., for $a, b \in \mathbb{F}_2^n$, it holds $a \leq b$ if and only if $a_i \leq b_i$ where $i \in \{1, \cdots, n\}$ is the first index such that $a_i \neq b_i$ [9]. An (n, m) S-box can be represented in the truth table form as a matrix of dimension $2^n \times m$ where each of the m columns represents a coordinate function.

The *Walsh-Hadamard* transform of an (n, m)-function F is defined as (see [10]):

$$W_{v \cdot F}(\omega) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus \omega \cdot x}, \ v \in (\mathbb{F}_2^m)^*, \ \omega \in \mathbb{F}_2^n \ . \tag{1}$$

In particular, the quantity $W_{v \cdot F}(\omega)$ measures the *correlation* between the component function $v \cdot F$ and the linear function $\omega \cdot x$. The maximum absolute value of the Walsh transform of F for a given $v \in (\mathbb{F}_2^m)^*$ is also called the *linearity* of the component function $v \cdot F$.

2.2 S-box Properties

In order to resist linear and differential cryptanalysis attacks, a balanced S-box should ideally have high nonlinearity and low differential uniformity. An (n, m)-function F is balanced if it takes every value of $\mathbb{F}_2^m$ the same number 2^{n-m} of times. Balanced (n, n)-functions correspond to bijective S-boxes.

The *linearity* of F (also called the *spectral radius*) is defined as the maximum linearity of all its component functions $v \cdot F$, where $v \in \mathbb{F}_2^{m*}$ [26, 10]:

$$\mathcal{L}(F) = \max_{\substack{\omega \in \mathbb{F}_2^n \\ v \in \mathbb{F}_2^{m*}}} |W_{v \cdot F}(\omega)|. \tag{2}$$

The *nonlinearity* N_F of an (n, m)-function F equals:

$$N_F = 2^{n-1} - \frac{1}{2}\mathcal{L}(F). \tag{3}$$

Let F be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$ with $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^m$. We define the *difference distribution table* of F with respect to a and b as:

$$D_F(a,b) = \left\{ x \in \mathbb{F}_2^n : F(x) \oplus F(x \oplus a) = b \right\}. \tag{4}$$

The entry at position (a,b) corresponds to the cardinality of the difference distribution table $D_F(a,b)$ and is denoted as $\delta_F(a,b)$. The *differential uniformity* δ_F is then defined as [27]:

$$\delta_F = \max_{\substack{a \in \mathbb{F}_2^{n*} \\ b \in \mathbb{F}_2^m}} \delta_f(a,b). \tag{5}$$

2.3 S-box Bounds

The nonlinearity of any (n,m) function F is bounded above by the so-called *covering radius bound*:

$$N_F \leq 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{6}$$

Functions satisfying the above bound are called *bent*, and they exist only when n is even. Further, for $m = n$ a better bound exists. The nonlinearity of any (n,n) function F is bounded above by the so-called *Sidelnikov-Chabaud-Vaudenay bound* [11]:

$$N_F \leq 2^{n-1} - 2^{\frac{n-1}{2}}. \tag{7}$$

Bound (7) is an equality if and only if F is an *Almost Bent* (AB) function, by definition of AB functions [10].

Functions that have differential uniformity equal to 2 are called *Almost Perfect Nonlinear* (APN) functions. Every AB function is also APN, but the converse does not hold in general. AB functions exist only in an odd number of variables, while APN functions also exist for an even number of variables. When discussing the differential uniformity parameter for permutations, the best possible (known) value is 2 for any odd n and also for $n = 6$. For n even and larger than 6, this is an open question. The differential uniformity value for the *inverse function* equals 4 when n is even and 2 when n is odd.

2.4 Affine Equivalence

Two S-boxes S_1 and S_2 of dimension $n \times n$ are *affine equivalent* if the following equation holds [10]:

$$S_1(x) = B \cdot (S_2(A \cdot x \oplus a)) \oplus b, \tag{8}$$

where A and B are invertible $n \times n$ matrices in $\mathbb{F}_2$ and $a, b \in \mathbb{F}_2^n$.

Both nonlinearity and differential uniformity are affine invariant, meaning that applying an affine transformation to an S-box will not change its values of those properties.

3 Cellular Automata

Cellular Automata (CA) are parallel computational models that have been used to simulate and analyze a wide variety of discrete complex systems in different application domains. A CA is characterized by a lattice of *cells*. During a single time step, each cell in the lattice synchronously updates its state according to a *local rule*, which is applied to the *neighborhood* of the cell. In what follows, we focus on *one-dimensional Boolean cellular automata*, meaning that the lattice is a one-dimensional array, and the state of each cell is binary. The following definition formalizes the two models of CA we address in this work:

Definition 1. *Let $f : \mathbb{F}_2^d \to \mathbb{F}_2$ and $n \geq d$. We define the following two models of* one-dimensional Boolean *CA with n input cells and* local rule *f:*

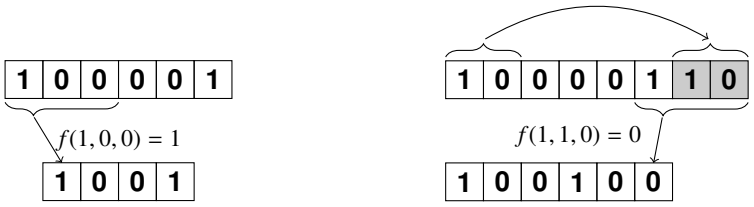
- No Boundary CA *(NBCA): $F : \mathbb{F}_2^n \to \mathbb{F}_2^{n-d+1}$ is defined for all $x \in \mathbb{F}_2^n$ as:*

$$F(x_1, x_2, \cdots, x_n) = (f(x_1, \cdots, x_d), f(x_2, \cdots, x_{d+1}), \cdots, f(x_{n-d+1}, \cdots, x_n)) \ . \tag{9}$$

- Periodic Boundary CA *(PBCA): $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is defined for all $x \in \mathbb{F}_2^n$ as:*

$$F(x_1, x_2, \cdots, x_n) = (f(x_1, \cdots, x_d), \cdots, f(x_{n-(d-2)}, \cdots, x_1), \cdots, f(x_n, \cdots, x_{d-1}) \ . \tag{10}$$

Figure 1 reports an example respectively of NBCA and PBCA based on the local rule $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ with $n = 6$ cells. Thus, a CA can be seen as



(a) No Boundary CA – NBCA (b) Periodic Boundary CA – PBCA

Figure 1: Examples of NBCA and PBCA with local rule 150, defined as $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$

a vectorial Boolean function where each coordinate function f_i corresponds to the local rule f applied to the *neighborhood* $(x_i, \cdots, x_{i+d-1})$. In the no boundary case, this rule is applied just up to the coordinate $n - d + 1$, meaning that the size of the input array shrinks by $d - 1$ cells. In the periodic setting, the CA array is seen as a ring, so that the first cell follows the last one. The remaining $d - 1$ cells are updated by using the first $d - 1$ as their right neighbors. In Figure 1b, this is depicted by appending a grey-shaded copy of the first $d - 1 = 2$ cells to the right of the CA array.

Notice that, since the local rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$ is a Boolean function, it can be represented by a truth table of 2^d bits. In the CA literature, another common way to

identify a local rule is by means of its *Wolfram code*, which is basically the decimal representation of the truth table. As an example, the Wolfram code for the CA local rule applied in Figure 1 is 150, since its truth table is $\Omega_f = (1, 0, 0, 1, 0, 1, 1, 0)$. The vectorial Boolean function F of a CA is also called the CA *global rule*.

Example 1. *The nonlinear transformation χ used in Keccak [4] is a PBCA with $n = 5$ cells and local rule $f : \mathbb{F}_2^3 \to \mathbb{F}_2$ defined as:*

$$f(x_1, x_2, x_3) = x_1 \oplus x_2 x_3 \oplus x_3 \quad . \tag{11}$$

The Wolfram code for such rule is 210.

Remark 1. *Note that if the rule adopted in Keccak is used with a cellular array of even length, then the resulting S-box is not bijective. In particular, the S-box is bijective if and only if the size of the CA is odd [16]. Since the Keccak rule has diameter $d = 3$, it results in an optimal S-box of size 3×3, while for the size 5×5 (adopted in the design of Keccak) the resulting S-box has suboptimal cryptographic properties. Naturally, as one would extend the size of this S-box by adding new cells to the CA, the cryptographic properties would become increasingly worse. In Section 4 we formalize this observation by analyzing how the nonlinearity and differential uniformity of a CA are affected by adding new cells, deriving upper bounds for these two cryptographic properties.*

Remark 2. *PBCA with $d = n$ actually correspond to* rotational symmetric S-boxes*, originally introduced in [34].*

We conclude this section by observing that, in the CA literature, the focus is usually on the *iterated behavior* of CA. In particular, the local rule is applied to all cells in parallel for multiple time steps, in order to study the long-term properties of the resulting dynamical system. On the other hand, in this work we only consider the situation where the CA is evolved for just one time step, and investigate the cryptographic properties of the resulting vectorial Boolean functions. This is the same approach used by the designers of the CA-based nonlinear transformation χ used in Keccak [4]. In general, we remark that studying the iterated behavior in a CA would correspond to determining the cycles of the corresponding S-box.

4 Theoretical Findings

In this section, we prove some bounds on the nonlinearity and differential uniformity of S-boxes defined by CA, relating them to the corresponding properties of the underlying local rules. To prove our results, we make use of the following theorem proved by Nyberg, concerning how the nonlinearity and the differential uniformity of an S-box are affected by adding a coordinate function while maintaining fixed the number of input variables [27].

Theorem 1. *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an S-box defined by m coordinate functions $f_1, \cdots, f_m : \mathbb{F}_2^n \to \mathbb{F}_2$, and let $g : \mathbb{F}_2^n \to \mathbb{F}_2$. Define $\tilde{F} : \mathbb{F}_2^n \to \mathbb{F}_2^{m+1}$ as follows:*

$$\tilde{F}(x_1, \cdots, x_{n+1}) = (f_1(x_1, \cdots, x_n), \cdots, f_m(x_1, \cdots, x_n), g(x_1, \cdots, x_n)) \ . \quad (12)$$

Then, the following upper bounds hold:

$$N_F(\tilde{F}) \leq \min\{N_F(F), N_F(g)\} \ . \quad (13)$$

$$\frac{1}{2}\delta_F \leq \delta_{\tilde{F}} \leq \delta_F \ . \quad (14)$$

Consider now a CA (either with no boundary or periodic boundary conditions) with n cells and local rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$. How do the nonlinearity and differential uniformity of F change by adding a new cell, thus obtaining a new CA $\tilde{F}$ of $n+1$ cells? Observe that Theorem 1 cannot be directly applied here, because we need to address the case where both a coordinate function *and* an input variable are added to the original CA. We first address this situation for generic S-boxes (i.e., not necessarily defined by a CA rule) in the following result:

Theorem 2. *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an S-box defined by m coordinate functions $f_1, \cdots, f_m : \mathbb{F}_2^m \to \mathbb{F}_2$, and let $g : \mathbb{F}_2^{n+1} \to \mathbb{F}_2$ be a Boolean function defined on $n+1$ variables. Define $\tilde{F} : \mathbb{F}_2^{n+1} \to \mathbb{F}_2^{m+1}$ as follows:*

$$\tilde{F}(x_1, \cdots, x_{n+1}) = (f_1(x_1, \cdots, x_n), \cdots, f_m(x_1, \cdots, x_n), g(x_1, \cdots, x_n, x_{n+1})) \ . \quad (15)$$

Then, $\tilde{F}$ satisfies the following bounds:

$$N_{\tilde{F}} \leq \min\{2 \cdot N_F, N_g\} \ , \quad (16)$$

$$\delta_F \leq \min\{2 \cdot \delta_F, \delta_g\} \ . \quad (17)$$

Proof. We begin by addressing the bound on nonlinearity. We are going to analyze the Walsh-Hadamard transform of $\tilde{F}$ by classifying its component functions as follows:

(i) The $2^m - 1$ component functions that do not select the new coordinate g, i.e., those described by the vectors $\tilde{v} = (v, 0) \in \mathbb{F}_2^{m+1}$, where $v \in \mathbb{F}_2^{m*}$.

(ii) The single component function that just selects g, defined by the vector $(\underline{0}, 1)$ where $\underline{0} \in \mathbb{F}_2^m$.

(iii) Finally, the $2^m - 1$ component functions that select g and whose first m coordinates are not all zeros, defined by the vectors $\tilde{v} = (v, 1) \in \mathbb{F}_2^{m+1}$, where $v \in \mathbb{F}_2^{m*}$.

Consider the component functions of type (i). Let $\tilde{v} = (v, 0) \in \mathbb{F}_2^{m+1}$, where $v \in \mathbb{F}_2^{m*}$. Then, the Walsh-Hadamard transform of $\tilde{v} \cdot \tilde{F}$ computed on $\omega \in \mathbb{F}_2^{n+1}$ equals

$$W_{\tilde{v}\cdot\tilde{F}}(\tilde{\omega}) = \sum_{\tilde{x} \in \mathbb{F}_2^{n+1}} (-1)^{\tilde{v}\cdot\tilde{F}(\tilde{x}) \oplus \tilde{\omega}\cdot\tilde{x}} = \sum_{(x, x_{n+1}) \in \mathbb{F}_2^{n+1}} (-1)^{(v,0)\cdot(F(x), g(x_{n+1})) \oplus (\omega, \omega_{n+1})\cdot(x, x_{n+1})} =$$

$$= \sum_{(x, x_{n+1}) \in \mathbb{F}_2^{n+1}} (-1)^{v\cdot F(\tilde{x}) \oplus \omega\cdot x} \cdot (-1)^{\omega_{n+1}\cdot x_{n+1}} \ . \quad (18)$$

Let us rewrite the right hand side of Eq. (18) by dividing the sum with respect to the value of x_{n+1}:

$$\begin{aligned} W_{\tilde{v}\cdot\tilde{F}}(\tilde{\omega}) &= \sum_{(x,0)\in\mathbb{F}_2^{n+1}} (-1)^{v\cdot F(x)\,\oplus\,\omega\cdot x} + \sum_{(x,1)\in\mathbb{F}_2^{n+1}} (-1)^{v\cdot F(x)\,\oplus\,\omega\cdot x\,\oplus\,\omega_{n+1}} = \\ &= \sum_{x\in\mathbb{F}_2^n} (-1)^{v\cdot F(x)\,\oplus\,\omega\cdot x} + (-1)^{\omega_{n+1}} \cdot \sum_{x\in\mathbb{F}_2^n} (-1)^{v\cdot F(x)\,\oplus\,\omega\cdot x} \quad . \end{aligned} \tag{19}$$

Notice that the two sums in Eq. (19) correspond to the Walsh-Hadamard coefficient $W_{v\cdot F}(\omega)$. Thus, it holds that

$$W_{\tilde{v}\cdot\tilde{F}}(\tilde{\omega}) = \begin{cases} 2 \cdot W_{v\cdot F}(\omega) & , \text{if } \omega_{n+1} = 0 \\ 0 & , \text{if } \omega_{n+1} = 1 \end{cases} \tag{20}$$

Hence, by Eq. (20) we have that the linearity of $\tilde{F}$ will be at least twice the linearity of F, from which it follows that

$$N_{\tilde{F}} \leq 2 \cdot N_F \quad . \tag{21}$$

Let us now consider the component of type (ii), i.e., the one defined by $\tilde{v} = (\underline{0}, 1)$. In this case, it is easy to see that $N_{\tilde{v}\cdot\tilde{F}} = N_g$, which yields

$$N_{\tilde{F}} \leq N_g \quad . \tag{22}$$

Since the nonlinearity of $\tilde{F}$ is defined as the minimum nonlinearity among all its component functions, by combining Eqs. (21) and (22) we get

$$N_{\tilde{F}} \leq \min\{2 \cdot N_F, N_g\} \quad . \tag{23}$$

Remark that, since we are considering an upper bound on the minimum nonlinearity among all component functions, even if the components of type (iii) yielded a lower nonlinearity the upper bound would still stand. Hence, we can safely ignore those components in this proof.

We now address the differential uniformity bound. Given $\tilde{a} = (a, a_{n+1}) \in \mathbb{F}_2^{n+1}$ and $\tilde{b} = (b, b_{m+1}) \in \mathbb{F}_2^{m+1}$, the difference distribution table of $\tilde{F}$ with respect to $\tilde{a}$ and $\tilde{b}$ is

$$\begin{aligned} D_{\tilde{F}}(a,b) &= \{\tilde{x} = (x, x_{n+1}) \in \mathbb{F}_2^{n+1} : \tilde{F}(\tilde{x} \oplus \tilde{a}) \oplus \tilde{F}(\tilde{x}) = \tilde{b}\} = \\ &= \{\tilde{x} \in \mathbb{F}_2^{n+1} : (F(x \oplus a), g(\tilde{x} \oplus \tilde{a})) \oplus (F(x), g(\tilde{x})) = (b, b_{m+1})\} = \\ &= \{\tilde{x} \in \mathbb{F}_2^{n+1} : [F(x \oplus a) \oplus F(x) = b] \wedge [g(\tilde{x} \oplus \tilde{a}) \oplus g(\tilde{x}) = b_{m+1}]\} = \\ &= \{\tilde{x} \in \mathbb{F}_2^{n+1} : [x \in D_F(a,b)] \wedge [\tilde{x} \in D_g(\tilde{a}, \tilde{b})]\} = \\ &= \{\tilde{x} \in \mathbb{F}_2^{n+1} : x \in D_F(a,b)\} \cap \{(x, x_{n+1}) \in \mathbb{F}_2^{n+1} : \tilde{x} \in D_g(\tilde{a}, \tilde{b})\} = A \cap B \quad . \end{aligned} \tag{24}$$

Clearly, from Eq. (24) we have that $B = D_g(\tilde{a}, \tilde{b})$, and thus $|B| = \delta_g(\tilde{a}, \tilde{b})$. On the other hand, for A we obtain $|A| = 2 \cdot |D_F(a, b)| = 2 \cdot \delta_F(a, b)$, since the vectors $\tilde{x}$ in A are constructed by taking all vectors x belonging to $D_F(a, b)$ and by appending to their right a 0 and a 1. Consequently, it holds that

$$\delta_{\tilde{F}}(\tilde{a}, \tilde{b}) = |A \cap B| \leq \min\{2 \cdot \delta_F(a, b), \delta_g(\tilde{a}, \tilde{b})\} \ . \tag{25}$$

Finally, observe that one can construct the delta difference tables of maximum cardinality of $\tilde{F}$ by taking all possible intersections between the difference distribution tables of maximum cardinality of F and g. Hence, the differential uniformity of $\tilde{F}$ satisfies

$$\delta_{\tilde{F}} \leq \min\{2 \cdot \delta_F, \delta_g\} \ . \tag{26}$$

□ □

Of course, the upper bounds given in Eq. (16) are not tight. In fact, the component functions of type (iii) could yield a lower nonlinearity and differential uniformity than those featured by the components of types (i) and (ii) considered in the proof of Theorem 2.

Before turning our attention to the CA case, we still need one more preliminary result about how the nonlinearity and differential uniformity of a Boolean function change by adding dummy variables:

Lemma 1. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function with nonlinearity N_f and differential uniformity δ_f. Given $t \in \mathbb{N}$, define $\tilde{f} : \mathbb{F}_2^{n+t} \to \mathbb{F}_2$ as follows:*

$$\tilde{f}(x_1, \cdots, x_n, x_{n+1}, \cdots, x_{n+t}) = f(x_1, \cdots, x_n) \ . \tag{27}$$

Then, the following equalities hold:

$$N_{\tilde{f}} = 2^t \cdot N_f \ , \delta_{\tilde{f}} = 2^t \cdot \delta_f \ . \tag{28}$$

Proof. We proceed by induction on t.

For $t = 1$, one can easily see that $\tilde{f}$ is a special case of the vectorial function $\tilde{F}$ considered in Theorem 2 with $m = 1$, with the difference that no new output coordinates are added. Hence, the Walsh-Hadamard transform of $\tilde{f}$ is described by Eq. (20), which yields $N_{\tilde{f}} = 2 \cdot N_f$. On the other hand, for $\tilde{a} = (a, a_{n+1}) \in \mathbb{F}_2^{n+1}$ and $b \in \mathbb{F}_2$, the difference distribution table of $\tilde{f}$ is

$$D_{\tilde{f}}(\tilde{a}, b) = \{(x, x_{n+1}) \in \mathbb{F}_2^{n+1} : f(x \oplus a) \oplus f(x) = b\} \ ,$$

from which it follows that $\delta_{\tilde{f}}(\tilde{a}, b) = 2 \cdot \delta_f(a, b)$, and thus $\delta_{\tilde{f}} = 2 \cdot \delta_f$.

Next, assume that $t > 1$, and consider the case $t + 1$, with $f' : \mathbb{F}_2^{n+t} \to \mathbb{F}_2$ indicating the function truncated at $n + t$ variables. Then, by induction hypothesis the following equalities are satisfied:

$$N_F(f') = 2^t \cdot N_F(f) \ ,$$
$$\delta_{f'} = 2^t \cdot \delta_f \ .$$

Similarly to the case $t = 1$, the Walsh-Hadamard coefficients of $\tilde{f} : \mathbb{F}_2^{n+t+1} \to \mathbb{F}_2$ are as in Eq. (20), from which one obtains

$$N_{\tilde{f}} = 2 \cdot N_{f'} = 2 \cdot 2^t \cdot N_f = 2^{t+1} \cdot N_f \ . \tag{29}$$

Finally, the difference distribution table $D_{\tilde{f}}(\tilde{a}, b)$ is again constructed by appending a 0 and a 1 to all vectors in $D_{f'}(a, b)$. Hence, the equality $\delta_{\tilde{f}}(\tilde{a}, b) = 2 \cdot \delta_{f'}(a, b)$ holds for all $\tilde{a} = (a, a_{n+t+1}) \in \mathbb{F}_2^{n+t+1}$, from which it finally follows that

$$\delta_{\tilde{f}} = 2 \cdot \delta_{f'} = 2 \cdot 2^t \delta_f = 2^{t+1} \cdot \delta_f \ .$$

□ □

Leveraging on the above results, we can now prove upper bounds on the nonlinearity and differential uniformity of S-boxes defined by CA, both in the no boundary and the periodic settings:

Theorem 3. *Let $f : \mathbb{F}_2^d \to \mathbb{F}_2$ and $n \geq d$. Then, the NBCA and PBCA $\tilde{F}$ with n input cells and local rule f satisfy the following bounds:*

$$N_{\tilde{F}} \leq 2^{n-d} \cdot N_f \tag{30}$$

$$\delta_{\tilde{F}} \leq 2^{n-d} \cdot \delta_f \ . \tag{31}$$

Proof. We first address the no boundary case. Let $\tilde{F} : \mathbb{F}_2^n \to \mathbb{F}_2^{n-d+1}$ be a NBCA with local rule f. We proceed by induction on $m = n - d + 1$.

For $m = 2$, we can apply Theorem 2 by setting $F = f$ and $g : \mathbb{F}_2^{d+1} \to \mathbb{F}_2$ defined as follows:

$$g(x_1, x_2, \cdots, x_{d+1}) = f(x_2, \cdots, x_{d+1}) \ .$$

Thus, Theorem 2 yields that

$$N_{\tilde{F}} \leq \min\{2 \cdot N_f, N_g\} \ ,$$
$$\delta_{\tilde{f}} \leq \min\{2 \cdot \delta_f, \delta_g\} \ .$$

Additionally, by Lemma 1 we know that

$$N_g = 2 \cdot N_f \ ,$$
$$\delta_g = 2 \cdot \delta_f \ .$$

Since $m - 1 = n - d + 1 - 1 = 1$, the three bounds are satisfied in the base case.

Next, let us assume that $m > 2$ and consider the case $m+1$, with $\tilde{F} : \mathbb{F}_2^{n+1} \to \mathbb{F}_2^{m+1}$ being the NBCA with $n + 1$ cells. In particular, define $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ as the NBCA with n cells, and $g : \mathbb{F}_2^{n+1} \to \mathbb{F}_2$ as $g(x_1, \cdots, x_{n+1}) = f(x_{n-d}, \cdots, x_{n+1})$. Again, Theorem 2 gives us that

$$N_{\tilde{F}} \leq \min\{2 \cdot N_F, N_g\} \ ,$$
$$\delta_{\tilde{F}} \leq \min\{2 \cdot \delta_F, \delta_g\} \ .$$

(a) PBCA S-Boxes

CA size n \ Rule size d	3	4	5	6	7
3	2	–	–	–	–
4	4	4	–	–	–
5	8	8	12	–	–
6	16	16	24	24	–
7	32	32	48	48	56

(b) Generic (n, n)-functions

$n \times n$	N_F
3×3	2
4×4	4
5×5	12
6×6	24
7×7	56

Table 1: Best attainable nonlinearity values for PBCA S-boxes and generic bijective S-boxes up to $n = 7$ variables.

while by Lemma 1 we obtain

$$N_g = 2^m \cdot N_f \ ,$$
$$\delta_g = 2^m \cdot \delta_f \ .$$

Remarking that $m + 1 = n - d + 1$, by induction hypothesis one finally gets

$$N_{\tilde{F}} \leq 2^m \cdot N_f = 2^{n-d} \cdot N_f \ ,$$
$$\delta_{\tilde{F}} \leq 2^m \cdot \delta_f = 2^{n-d} \cdot \delta_f \ ,$$

which concludes the proof for the NBCA case. Finally, for the periodic case it just suffices to observe that the PBCA is constructed by adding $n - d$ coordinate functions to the NBCA $\tilde{F}$ without extending the number of input variables, where the new coordinates always coincide with the local rule f applied on the rightmost and leftmost $d - 1$ cells. Hence, Theorem 1 can be applied here, from which one deduces that the same bounds for nonlinearity and differential uniformity also hold for the PBCA case. □ □

Tables 1a and 1b report the best nonlinearity values respectively reachable by PBCA as given by Theorem 3, for various values of d and n, and by generic bijective (n, n)-functions. Table 1a is lower triangular because the bound of Theorem 3 is meaningful only if $n \geq d$. For the maximum nonlinearity of N_f of the local rule we considered the quadratic bound, since it is known to be optimal for balanced Boolean functions of sizes up to $d = 7$ variables [9]. By comparing Tables 1a and 1b one can see that the only case where CA are able to reach the same best values as generic (n, n)-functions is when $d = n$, i.e., the rotation-symmetric case which corresponds to the diagonal of Table 1a. This also explains from a theoretical point of view why the nonlinearity of the CA χ used in Keccak is suboptimal with respect to the Sidelnikov-Chabaud-Vaudenay bound, since the neighborhood size of the rule is $d = 3$ while $n = 5$. The χ rule is, however, optimal with respect to the nonlinearity bound given in Theorem 3.

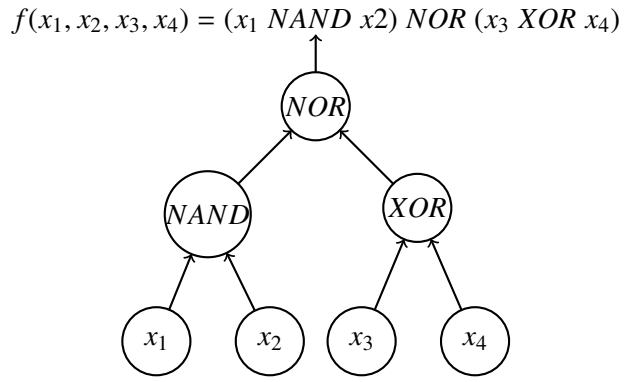


Figure 2: Example of GP tree encoding a Boolean function of 4 variables.

5 Experimental Results

5.1 Genetic Programming Approach

Genetic Programming (GP) is an *Evolutionary Algorithm* (EA) in which the data structures that undergo optimization are computer programs (i.e., executable expressions) [2]. Although GP has a history longer than 50 years, its full acceptance is due to the work of John Koza at the beginning of the 1990s, in which he formalized the idea of employing chromosomes based on tree data structures. Since the aim of GP is to automatically generate new programs, each individual in a population represents a computable expression, whose most common form are symbolic expressions corresponding to parse trees. A *parse tree* (syntax tree) is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. A tree can represent a mathematical expression, a rule set or a decision tree, for instance. The building elements in a tree-based GP are *functions* (inner nodes) and *terminals* (leaves, problem variables); both functions and terminals are known as *primitives*. For further information about GP, we refer interested readers to [22, 32].

In our experiments, the function set consists of several Boolean primitives that enable representation of any Boolean function: NOT, which inverts its single argument, XOR, NAND, NOR, each of which takes two input arguments. Additionally, we use the function IF, which takes three arguments and returns the second one if the first one evaluates to *true*, and the third one otherwise. This function corresponds to the multiplexer gate (MUX). In our setting, GP evolves a Boolean function of n variables in the form of a tree which represents a CA local rule.

Figure 2 depicts an example of GP tree which represents a Boolean function of 4 variables $f : \mathbb{F}_2^4 \to \mathbb{F}_2$. The leaves in this case stand for the 4 input variables, while the internal nodes correspond to the *NAND* and *XOR* respectively combining x_1, x_2 and x_3, x_4. Finally, the root node gives the output value of the function by combining the results of the *NAND* and *XOR* nodes through a *NOR*.

Throughout our experiments, we assume the following: the state of a CA is

represented by a periodic one-dimensional binary array of size n. The elements of the binary array are used as Boolean variables in a GP tree (GP terminals), where the variable c_0 denotes the value that is being updated. The variables $c_1, \ldots, c_{n-1}$ denote the cells to the right of the current cell. The neighborhood of a cell is formed by the cell itself and the $n-1$ cells to its right, so each value in the current state can be used in a local update rule, which corresponds to the case of rotation-symmetric S-boxes (i.e., $d = n$). A candidate Boolean function obtained with GP is evaluated in the following manner: all the possible 2^n input states are considered, and for each state the same rule is applied in parallel to each of the variables to determine the next state. The obtained global rule represents a candidate S-box.

In the evolution process, GP uses a 3-tournament selection, where the worst of three randomly selected individuals is eliminated. A new individual is then created by applying crossover to the remaining two individuals from the tournament. The new individual is then mutated with a probability of 0.5. We use the mutation probability to select whether an individual would be mutated or not, and the mutation operator is executed only once on a given individual; e.g., if the mutation probability is 0.5, then on average 5 out of every 10 new individuals will be mutated and one mutation will be performed on each of those 5 individuals. This procedure is illustrated in Algorithm 1.

The variation operators are simple tree crossover, uniform crossover, size fair, one-point, and context preserving crossover (selected at random), and subtree mutation [33]. All our experiments suggest that having a maximum tree depth equal to the size of S-box is sufficient (i.e., tree depth equals n, which is the number of Boolean variables). The initial population is created at random and every experiment is repeated 50 times.

We emphasize that not all bijective S-boxes can be represented by CA rules and consequently the number of S-boxes expressible through CA is smaller than the total number of S-boxes of a certain size. Considering the AES S-box as an example, it is possible to see that this S-box cannot be obtained with a single CA rule. Still, this does not mean there are no S-boxes of that size with the same properties that cannot be constructed with a single CA rule. On the other hand, there are infinitely many ways how one can represent an S-box with CA rules. For example, with the tree representation for the rule, it suffices to consider the trivial approach where one adds subexpressions that cancel themselves out. Accordingly, the number of CA rules representations is much larger than the number of S-boxes and it is impossible to exhaustively visit them even for small sizes.

We note that Picek et al. showed that genetic programming can be used to design CA-based S-box with optimal cryptographic properties up to size 7×7 (not counting the APN in dimension 6) [30, 29]. Besides the cryptographic properties, they demonstrated how it is possible to use the same approach to reduce the size of the CA rules (which consequently reduces the area of S-boxes). Finally, they discussed the power consumption of such CA-based S-boxes and they found them to be comparable or better than several S-boxes used in modern ciphers [30].

Algorithm 1 Genetic Programming evolution

repeat
  randomly select 3 individuals;
  remove the worst of 3 individuals;
  child = crossover (remaining two individuals);
  perform mutation on *child*, with given individual mutation probability;
  generate S-box using *child* Boolean function
  evaluate S-box
  assign fitness to *child*
  insert *child* into population;
until stopping criteria reached

5.2 Reverse Engineering of CA-based S-boxes

Here, we assume that we already have an S-box and we want to obtain its CA rule representation. There are two obvious reasons why one would want to do this. The first reason is to check whether a certain S-box is expressible with a CA rule. The second reason is to obtain a combinatorial circuit representation of an S-box (in the case that the S-box can be represented with a CA rule). The first objective can be reached with another technique that we briefly explain.

Given the truth table description of an S-box, the task of determining the local rule of the corresponding CA can be determined using the De Bruijn graph representation [37]. The *De Bruijn graph* associated to a CA with local rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$ is a directed graph $G = (V, E)$ where $|V| = 2^{d-1}$. In particular, each vertex in G is labeled with a binary vector of length $d - 1$. An edge from vertex $a \in V$ to $b \in V$ exists if and only if a and b *overlap* respectively on the last and the first $d - 2$ coordinates. For example, for $d = 3$ the De Bruijn graph has an edge from $a = 01$ to $b = 10$ since a and b have a 1 respectively in the last and in the first position. A CA local rule is represented over the De Bruijn graph as a labeling of the edges, i.e., a function $l : E \to \{0, 1\}$. Hence, in the example above the labeling of $(01, 10)$ would be the result of the local rule applied to the input 010. Figure 3 depicts the De Bruijn graph representation of the CA rule χ used in Keccak.

To check if a given S-box of length n can be expressed using a CA rule with diameter $d < n$, one could start from a De Bruijn graph with 2^{d-1} vertices and iteratively label the edges by reading the entries in the truth table of the S-box. As soon as an inconsistency is found (i.e., an edge gets more than one label), one knows that the S-box is not representable with a CA of diameter d. On the other hand, if after reading the whole S-box each edge has a unique label, then the De Bruijn graph of a CA rule implementing that S-box is obtained.

As an example, we consider the APN function in dimension 6 [6]. Considering the last occurring value that equals 22, we see that this S-box cannot be generated with a CA rule. This is due to the fact that for the input 63 (111111 in binary) the output equals 22 (010110), which means that the local rule is not consistent because

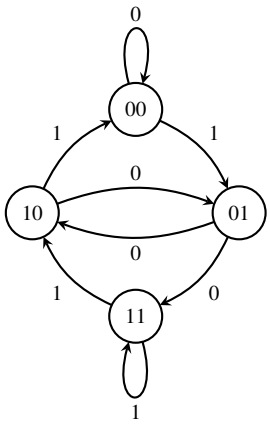


Figure 3: De Bruijn graph associated to CA rule of Keccak.

it assigns different values to the cells 1, 3, and 6 (value 0) and to the cells 2, 4, and 5 (value 1).

We note that the above procedure cannot help us to reach the second objective, i.e., finding a combinatorial representation of a given S-box. Additionally, this problem is much more difficult since there exist many circuits mapping to the same truth table, and there is no easy way to determine the smallest circuit. Consequently, we use a regression process based on GP in order to find an efficient combinatorial circuit for a given S-box defined by a CA.

More formally, let $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ be a CA-based S-box of n-bit defined by a local rule $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$. Given a GP tree T_f encoding the local rule, we define the *size* $S(T_f)$ of T_f as the number of primitives composing the tree. The optimization objective is thus to find a second GP tree T'_f encoding the same local rule f and such that $S(T'_f) < S(T_f)$.

As shown in [30], the main idea behind this approach is that the size of the GP tree encoding a CA rule gives a good approximation of its *implementation cost* as expressed by the *GE* measure, which stands for *Gate Equivalent* (i.e., the number of equivalent *NAND* gates in the specified technology). In particular, to properly evaluate the tree size and the influence of its elements to the implementation cost, we define an *implementation weight* using the *GE* measure. This weight reflects the relative area of those functions as follows: the weights of *NAND* and *NOR* gates are set to 1, the *XOR* weight is 2, the weight of *IF* is 2.33 and the weight of *NOT* equals 0.667 (note that the weights can be easily modified to reflect different hardware properties).

The GP evolution process used in our experiments is guided by a fitness function that describes *the difference* between the S-box obtained by a CA rule, and the one given as an input parameter. The design of the objective function is such that the truth table output of the current CA rule is compared with the truth table of the given S-box.

Table 2: Reverse engineering approach, Eq. (32)

S-box size	Original size	New size			
		Max	Min	Avg	Std dev
4×4	77	26	**11**	13.96	3.36
5×5	27	30	**9**	15.32	6.13
6×6	26	31	**13**	20.11	5.34
7×7	23	42	**13**	22.19	8.99

Rather than only counting all the bits in which the two differ, we employ a two-stage fitness. In the first stage, the number of differing bits is minimized, which is the primary objective. Only if the difference is zero, we add a term devoted to minimizing the size of the resulting CA rule (enforcing parsimony). This term is defined so that it is inversely proportional to the size of the GP tree representing an individual.

$$fitness_{re} = nErrors + \Delta_{nErrors,0}\left(\frac{treeSize}{maxTreeSize}\right), \tag{32}$$

where *nErrors* denotes the number of differing bits in the truth tables, while *treeSize* and *maxTreeSize* are respectively the actual tree size and the maximum size that the tree may assume given the maximum tree depth and the number of arguments of the GP functions. Note that this fitness measure is minimized; the correct CA rule will have a fitness in the range [0, 1], which in that case depends only on the tree size.

In our experiments, we used as inputs the S-boxes obtained in [30, 29], which were evolved through GP. By doing so, we can be sure that the S-box can be represented with a CA rule, while trying to find an implementation with a smaller complexity. In Table 2, we give results for each S-box size. Column *Original size* gives the size of the target S-box used in the regression, and the other columns give statistics for the obtained results (here, column *Min* represents the best obtained solution). Remark that all columns refer to the number of primitives in the GP individuals *not* multiplied by their implementation weights. Note also that we randomly selected the target input S-boxes among those with the best obtained properties from [30, 29].

We notice that for all the presented sizes, our procedure is able to find CA rules encoded by much smaller GP trees than those used in the original cases. More precisely, up to the 6×6 size, we have 100% success rate in obtaining the correct rules. For the 7×7 size, that percentage equals 96.7%, which is still an excellent result. This makes our methodology a viable option when the goal is to implement the S-box obtained via a CA rule in hardware, since a rule with smaller GP tree will mean a smaller gate count (measured in *GE*), and consequently a smaller area. Since smaller S-boxes are used in lightweight cryptography, where one common objective is to have as small as possible areas, we deem our approach useful in the design phase of lightweight ciphers. As an interesting fact, we note that we also tried this approach with the Keccak S-box, and among the obtained solutions there

Table 3: Results for exhaustive search

n	Number of CA-based S-boxes	Number of bijective CA-based S-boxes	Number of optimal CA-based S-boxes
3	256	36	12
4	65 536	1 536	512
5	4 294 967 296	22 500 002	2 880

Table 4: Equivalence classes of bijective 3×3 CA-based S-boxes

Class	Representative	Number of S-boxes	Optimal
0	0,1,2,3,4,5,6,7	6	No
1	0,1,2,3,4,5,7,6	6	No
2	0,1,2,3,4,6,7,5	12	No
3	0,1,2,4,3,6,7,5	12	Yes

were several occurrences of the exact same CA rule as used in Keccak.

When working with the 8×8 S-box size, our regression technique was unable to find any correct rule corresponding to the given S-box. We experimented with an S-box originally obtained with a CA rule consisting of 177 primitives, which is a much longer rule when compared with the sizes where our approach found correct rules.

5.3 Equivalence Classes

In this section, we concentrate on S-boxes of sizes up to 5×5, i.e., those that can be exhaustively checked when considering CA-based S-boxes. First, in Table 3 we give results for sizes 3×3, 4×4, and 5×5. As it can be seen, from the corpus of possible CA-based S-boxes, only a fragment is bijective. Additionally, from the bijective S-boxes again only a small part is optimal with regards to the nonlinearity and differential uniformity properties. We emphasize that this is the total number of CA-based S-boxes since other S-boxes that are affine equivalent to these cannot be obtained with a single CA rule.

For sizes larger than 5×5, an exhaustive search is not possible. Still, a simple estimation can be made. The total number of CA-based S-boxes equals the number of Boolean functions of the corresponding size, i.e., 2^{2^n}. Next, the number of balanced Boolean functions of size n equals $\binom{2^n}{2^{n-1}}$, which also represents a trivial upper bound on the number of bijective CA-based S-boxes. As an example, for size 5×5, the number of bijective S-boxes obtainable with a single CA rule forms only 26.7% of possible balanced Boolean functions of size 5.

When considering 3×3 size, we give details in Table 4. In Table 5, we give details about equivalence classes of 4×4 S-boxes that are CA-based and bijective. For the 4×4 S-box size, Leander and Poschmann defined optimal S-boxes as those being bijective, with maximal nonlinearity (equal to 4), and minimal differential

Table 5: Equivalence classes of bijective 4×4 CA-based S-boxes. Note that we give class representatives for each class but that does not necessarily mean it is possible to construct them with a single CA rule.

Class	Representative	Number of S-boxes	Optimal
0	F,D,B,9,7,5,3,1,E,C,A,8,6,4,2,0	16	No
1	0,1,2,3,4,5,6,7,8,9,A,B,C,D,F,E	32	No
3	0,1,2,3,4,5,6,7,8,9,A,B,D,E,F,C	32	No
4	0,1,2,3,4,5,6,7,8,9,A,B,D,C,F,E	16	No
6	0,1,2,3,4,5,6,7,8,9,A,C,B,D,F,E	32	No
9	0,1,2,3,4,5,6,7,8,9,A,C,D,E,B,F	64	No
41	0,1,2,3,4,5,7,6,8,A,9,C,B,F,E,D	128	No
193	0,1,2,3,4,5,8,A,6,C,7,F,D,B,9,E	128	No
270	0,1,2,3,4,6,8,B,5,C,9,D,E,A,7,F	128	Yes (G_4)
272	0,1,2,3,4,6,8,B,5,C,D,7,9,F,A,E	128	Yes (G_6)
273	0,1,2,3,4,5,8,A,6,C,7,F,E,B,9,D	128	No
278	0,1,2,3,4,6,8,B,5,C,D,7,A,F,9,E	128	Yes (G_5)
279	0,1,2,3,4,5,8,A,6,B,C,7,D,F,E,9	128	No
281	0,1,2,3,4,5,7,8,6,9,A,C,F,B,D,E	128	No
282	0,1,2,3,4,6,8,B,5,C,D,7,F,9,E,A	128	Yes (G_3)
288	0,1,2,3,4,5,6,7,8,9,C,E,F,B,D,A	32	No
289	0,1,2,3,4,5,6,7,8,9,C,E,B,F,D,A	64	No
291	0,1,2,3,4,5,7,6,8,A,9,B,C,F,E,D	64	No
294	0,1,2,3,4,5,6,7,8,9,B,A,E,F,D,C	32	No

uniformity (again equal to 4) [23]. There are in total 16 non equivalent classes of S-boxes with such properties (denoted $G_0, \ldots, G_{15}$).

5.4 Future Work

There are several options for future developments, the most obvious one being focusing on the 8×8 case. In particular, we remark that the 8×8 CA-based S-boxes evolved through GP in [30, 29] had suboptimal cryptographic values. Hence, besides applying the reverse engineering approach on these CA (which we tried without much success in this paper), a first direction for future work would be to improve the GP performance to evolve optimal 8×8 CA-based S-boxes. Since the main reason GP failed in the 8×8 case could be the enormous size of the resulting search space, a possible idea to overcome this obstacle would be to reduce this space by either experimenting with the GP parameters (such as set of primitives and tree depth) or by designing specific genetic operators preserving some basic cryptographic properties (such as bijectivity). In this way, the GP heuristic would explore a smaller set of candidate solutions and could have better possibilities at locating S-boxes with optimal cryptographic properties.

Naturally, in this paper we concentrated only on a small set of cryptographic properties and one could include in the fitness function other relevant properties

like the *algebraic degree*. As it can be seen in Table 1a, for the sizes 4×4 and 6×6 the best obtainable nonlinearity still equals the quadratic bound respectively when $d = 3$ and $d = 5$. Hence, it would be interesting to investigate whether the GP heuristic adopted in [30, 29] is still able to evolve S-boxes with optimal nonlinearity when $d = n - 1$, i.e., when the local rule depends on all input variables except one. From the theoretical side, another possible direction for future research is to investigate lower bounds on the nonlinearity and differential uniformity of CA S-boxes based on specific subclasses of local rules, such as *plateaued Boolean functions*. We note that this question has already been investigated in Mariot et al. [24] for permutive local rules. A local rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$ is called *permutive* if it is defined as $f(x_1, \cdots, x_{d-1}, x_d) = g(x_1, \cdots, x_{d-1}) \oplus x_d$, where g is a function of $d - 1$ variables. Computer searches performed on small input size suggest that permutive rules always satisfy with equality the bound on nonlinearity given in Theorem 3. An example of permutive rule is the function χ used in the Keccak S-box. However, the authors of [24] later observed a mistake in the proof of this fact, and they are currently investigating either how to fix it or to disprove it.

Another interesting venue for future research is to extend our reverse engineering approach through affine equivalence. As a matter of fact, the fitness function used in this paper aimed at evolving CA rules which resulted in the *same* S-boxes given as input to GP. An interesting question to investigate is whether it is possible to reverse engineer a CA-based S-box through GP in order to obtain an affine equivalent CA, thus having the same cryptographic properties. We remark that the only affine transformations preserving the CA property (i.e., such that the affine equivalent versions of a CA are CA themselves) are those defined by *circulant matrices* [20]. Since these matrices have an easy combinatorial characterization, a possible idea to address this question could be to generate all CA-based S-boxes which are affine equivalent to a given input CA, and then try apply our GP reverse engineering approach on them, to investigate if even smaller rules can be found.

More in general, one could extend the above line of research by considering whether a generic S-box that is not expressible by a CA admits an affine equivalent S-box defined by a CA. As we mentioned in Section 5.1, not all S-boxes of a specified size can be expressed by a CA of the same diameter, due to a simple combinatorial argument: while the number of (n, n)-functions is $n \times 2^{2^n}$, the number of PBCA of size n and diameter $d = n$ is just 2^{2^n}. However, we note that applying a generic affine transformation (i.e., not necessarily defined by a circulant matrix) to a CA does not yield in general a CA. It would be interesting to find a procedure that is able to solve the inverse problem, that is, starting from a (n, n)-function which is not a CA, determine whether there is an affine (non-circulant) transformation which is defined by a CA. A straightforward method to perform this task would be generate all S-boxes which are equivalent to the starting one, and then determine if some of them can be expressed by a single CA rule using the De Bruijn graph representation. However, we note that as the size of the S-box increases, exhaustively enumerating the affine equivalent version of an S-box becomes computationally unfeasible.

The experiments presented in this paper focused on evolving CA as nonlinear

elements for the confusion phase of a block cipher. Another interesting perspective would be to investigate the use of CA also for the diffusion phase. Since linear diffusion layers are often implemented in the literature using MDS linear codes, a possible venue for future research in this context is to optimize through GP the implementation cost of the MDS matrices arising from linear CA.

Recall that a (n, k, t) binary linear cyclic code C is a k-dimensional subspace of the vector space $\mathbb{F}_2^n$ such that each pair of vectors (called *codewords*) is at the Hamming distance at least t, which is also closed under cyclic shifts (so that $x \in C$ implies $\tilde{\sigma}(x) \in C$). On the other hand, a CA is called *linear* if its local rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$ is defined as an XOR of a subset of cells in the neighborhood.

Consider now a NBCA of length $n = m + d - 1$ equipped with a linear rule $f : \mathbb{F}_2^d \to \mathbb{F}_2$. In this case, the global rule of F is defined by a $m \times (m + d - 1)$ *transition matrix* M_F of the following form:

$$M_F = \begin{pmatrix} a_1 & \cdots & a_d & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & a_1 & \cdots & a_d & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & a_1 & \cdots & a_d \end{pmatrix}. \tag{33}$$

The vectorial Boolean function defined by such CA is determined by the matrix-vector multiplication $y = M_F x^\top$.

One can notice that the transition matrix in Eq. (33) actually has the same form as the generator matrix of a cyclic code (see [25]). Hence, an interesting idea would be to use linear CA to implement MDS cyclic codes for diffusion layers in block ciphers. This would require first to characterize the systematic generator matrix S_F of the cyclic codes induced by a linear CA with transition matrix M_F as defined in Eq. (33). Consequently, one could employ the non-systematic part of S_F as a MDS matrix to implement a linear diffusion layer. It is known that these MDS matrices are not sparse [1]. Thus, a possible future work in this context could be to optimize through GP the implementation cost of the MDS matrices arising from linear CA.

Finally, it would be interesting to see how CA rules can be integrated into unbalanced MISTY constructions as presented by Canteaut et al. [8]. Since the aim there is to construct lightweight S-boxes of larger sizes, a procedure to obtain building blocks (i.e., smaller S-boxes) could be beneficial.

6 Related Work

Most of the block ciphers based on the dynamics of cellular automata focus on the use of *reversible* CA (RCA). A CA is reversible if its global rule $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is bijective and the inverse $G = F^{-1}$ is again the global rule of a CA. In a CA-based block cipher, the idea is to represent a block of plaintext as the initial configuration of the CA. The global rule is then applied for a certain number of steps to obtain the encrypted block. For decryption, the inverse global rule is applied for the same number of steps starting from the ciphertext block to recover the plaintext.

The first block cipher based on cellular automata was proposed by Gutowitz [19]. In particular, for the substitution phase *block CA* were used to ensure the invertibility of the resulting S-box. In a block CA, the local rule does not determine the next state of a single cell, but rather the state of a block of adjacent cells. The cellular array is partitioned in blocks of equal length, and then a permutation is applied to each block in parallel. In the next step, the partition is shifted one cell to the right with periodic boundary conditions.

A second type of CA which has been used for block ciphers are *second-order CA*, where the state of a cell is determined by XORing its previous state with the result of the local rule. Hence, the configuration at time $t-1$ can be computed by knowing both the configurations at time t and $t+1$. Seredynsky et al. investigated second-order CA as S-boxes, by assessing the *avalanche properties* of several rules with diameter $d = 5, 7$ and array lengths $n = 32, 64$ [35].

Another interesting kind of CA for block ciphers are the so-called *complementing landscapes cellular automata* (CLCA), where the state of a cell is flipped if and only if a pattern belonging to a specific landscape occurs in the surrounding cells. Daemen et al. studied CLCA for designing block ciphers, discovering the rule χ used in Keccak [16, 4]. In particular, this rule induces an invertible CA if the length n of the cellular array is odd.

From the EC perspective, we mention only several characteristic approaches, all of which use the permutation encoding. Clark et al. used the principles from the evolutionary design of Boolean functions to evolve S-boxes with the desired cryptographic properties for sizes up to 8×8 [14]. Burnett et al. used a heuristic method to generate MARS-like S-boxes [7]. With their approach, they were able to generate a number of S-boxes of appropriate sizes that satisfy all the requirements placed on a MARS S-box. Picek et al. used Cartesian Genetic Programming and Genetic Programming to evolve S-boxes and discussed how to obtain permutation based encoding with those algorithms [31]. Picek et al. presented an improved fitness function with which EC is able to find higher nonlinearity values for a number of S-box sizes [28]. Picek et al. also discussed how to use genetic programming to evolve cellular automata rules that in turn can be used to generate S-boxes with good cryptographic properties [30, 29]. Finally, Picek et al. used the same genetic paradigm to evolve CA rules to be used in S-boxes but where the goal is not only cryptographic properties but also implementation perspective [30]. Interestingly, the results obtained in these two papers, where GP is used to evolve CA rules, outperform any other solutions obtained with heuristics for sizes 5×5 up to 7×7.

7 Conclusions

In this paper, we approach the problem of designing S-boxes with good cryptographic properties with cellular automata rules that are then mapped to S-boxes. We first show upper bounds for the nonlinearity and differential uniformity achievable by CA, both in the no boundary and periodic boundary settings.

Next, we use GP in order to "reverse-engineer" an S-box. There, we use the regression approach to find the shortest CA rule resulting in a specific S-box. This approach has interesting ramifications from two aspects: fast checking whether an S-box is expressible through CA rules and obtaining different rules (and consequently their sizes) resulting in a specific S-box. Finally, we conduct an exhaustive search of CA-based S-boxes of sizes 3×3, 4×4, and 5×5. For the first two dimensions, we also classify them with respect to the affine equivalence notion.

References

[1] D. Augot and M. Finiasz. Direct construction of recursive MDS diffusion layers using shortened BCH codes. In *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, pages 3–17, 2014.

[2] T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, 2000.

[3] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Radiogatún, a belt-and-mill hash function. *IACR Cryptology ePrint Archive*, 2006:369, 2006.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The Keccak reference, January 2011. `http://keccak.noekeon.org/`.

[5] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 450–466, Berlin, Heidelberg, 2007. Springer-Verlag.

[6] K. A. Browning, J. F. Dillon, M. T. McQuistan, and A. J. Wolfe. An APN permutation in dimension six. *Finite Fields: theory and applications*, pages 33–42, 2010.

[7] L. Burnett, G. Carter, E. Dawson, and W. Millan. Efficient Methods for Generating MARS-Like S-Boxes. In *Proceedings of the 7th International Workshop on Fast Software Encryption*, FSE '00, pages 300–314, London, UK, UK, 2001. Springer-Verlag.

[8] A. Canteaut, S. Duval, and G. Leurent. Construction of Lightweight S-Boxes Using Feistel and MISTY Structures. In O. Dunkelman and L. Keliher, editors, *Selected Areas in Cryptography - SAC 2015: 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 373–393, Cham, 2016. Springer International Publishing.

[9] C. Carlet. Boolean Functions for Cryptography and Error Correcting Codes. In Y. Crama and P. L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 257–397. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[10] C. Carlet. Vectorial Boolean Functions for Cryptography. In Y. Crama and P. L. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 398–469. Cambridge University Press, New York, USA, 1st edition, 2010.

[11] F. Chabaud and S. Vaudenay. Links between differential and linear cryptanalysis. In A. De Santis, editor, *Advances in Cryptology — EUROCRYPT '94: Workshop on the Theory and Application of Cryptographic Techniques Perugia, Italy, 1994 Proceedings*, pages 356–365. Springer Berlin Heidelberg, 1995.

[12] F. M. Christoph Dobraunig, Maria Eichlseder and M. Schläffer. Ascon, 2014. CAESAR submission, `http://ascon.iaik.tugraz.at/`.

[13] L. Claesen, J. Daemen, M. Genoe, and G. Peeters. Subterranean: A 600 Mbit/sec cryptographic VLSI chip. In *Computer Design: VLSI in Computers and Processors, 1993. ICCD '93. Proceedings., 1993 IEEE International Conference on*, pages 610–613, Oct 1993.

[14] J. A. Clark, J. L. Jacob, and S. Stepney. The design of S-boxes by simulated annealing. *New Generation Computing*, 23(3):219–231, Sept. 2005.

[15] J. Daemen and C. S. K. Clapp. Fast Hashing and Stream Encryption with PANAMA. In *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, pages 60–74, 1998.

[16] J. Daemen, R. Govaerts, and J. Vandewalle. Invertible shift-invariant transformations on binary arrays. *Applied Mathematics and Computation*, 62(2):259 – 277, 1994.

[17] J. Daemen, R. Govaerts, and J. Vandewalle. A new approach to block cipher design. In R. Anderson, editor, *Fast Software Encryption: Cambridge Security Workshop Cambridge, U. K.,1993 Proceedings*, pages 18–32, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[18] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[19] H. Gutowitz. Cryptography with dynamical systems. In *Cellular Automata and Cooperative Systems*, pages 237–274. Springer, 1993.

[20] S. Kavut. Results on rotation-symmetric s-boxes. *Inf. Sci.*, 201:93–113, 2012.

[21] L. R. Knudsen and M. Robshaw. *The Block Cipher Companion.* Information Security and Cryptography. Springer, 2011.

[22] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[23] G. Leander and A. Poschmann. On the Classification of 4 Bit S-Boxes. In C. Carlet and B. Sunar, editors, *Arithmetic of Finite Fields*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer Berlin Heidelberg, 2007.

[24] L. Mariot and A. Leporati. A cryptographic and coding-theoretic perspective on the global rules of cellular automata. *Natural Computing*, 2017.

[25] R. J. McEliece. *Theory of Information and Coding*. Cambridge University Press, New York, NY, USA, 2nd edition, 2001.

[26] K. Nyberg. On the construction of highly nonlinear permutations. In R. Rueppel, editor, *Advances in Cryptology - EUROCRYPT' 92*, volume 658 of *Lecture Notes in Computer Science*, pages 92–98. Springer Berlin Heidelberg, 1993.

[27] K. Nyberg. S-boxes and round functions with controllable linearity and differential uniformity. In *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, pages 111–130, 1994.

[28] S. Picek, M. Cupic, and L. Rotim. A New Cost Function for Evolution of S-boxes. *Evolutionary Computation*, 2016.

[29] S. Picek, L. Mariot, A. Leporati, and D. Jakobovic. Evolving S-boxes Based on Cellular Automata with Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 251–252, New York, NY, USA, 2017. ACM.

[30] S. Picek, L. Mariot, B. Yang, D. Jakobovic, and N. Mentens. Design of S-boxes Defined with Cellular Automata Rules. In *Proceedings of the Computing Frontiers Conference*, CF'17, pages 409–414, New York, NY, USA, 2017. ACM.

[31] S. Picek, J. F. Miller, D. Jakobovic, and L. Batina. Cartesian Genetic Programming Approach for Generating Substitution Boxes of Different Sizes. In *GECCO Companion '15*, pages 1457–1458, New York, NY, USA, 2015. ACM.

[32] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[33] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[34] V. Rijmen, P. S. L. M. Barreto, and D. L. G. Filho. Rotation symmetry in algebraically generated cryptographic substitution tables. *Inf. Process. Lett.*, 106(6):246–250, 2008.

[35] M. Seredynski and P. Bouvry. Block encryption using reversible cellular automata. In *Cellular Automata, 6th International Conference on Cellular Automata for Research and Industry, ACRI 2004, Amsterdam, The Netherlands, October 25-28, 2004, Proceedings*, pages 785–792, 2004.

[36] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[37] K. Sutner. De bruijn graphs and linear cellular automata. *Complex Systems*, 5(1):19–30, 1991.