A 10-Bit S-Box Generated by Feistel Construction from Cellular Automata

Thomas Prévost
Bruno Martin
I3S, Université Côte d'Azur
Euclide B, 2000 Rte des Lucioles
06900 Sophia Antipolis, France

We propose a new 10-bit S-box generated from a Feistel construction. The subpermutations are generated by a five-cell cellular automaton (CA) based on a unique, well-chosen, local transition rule and bijective affine transformations. In particular, the CA rule is chosen based on empirical tests of its ability to generate good pseudorandom output on a ring CA. Similarly, the Feistel network layout is based on empirical data regarding the quality of the output S-box.

We perform cryptanalysis of the generated 10-bit S-box: testing the properties of algebraic degree, algebraic complexity, nonlinearity, strict avalanche criterion, bit independence criterion, linear approximation probability, differential approximation probability, differential uniformity and boomerang uniformity. We relate the properties to those of the AES S-box. We find security properties comparable to or sometimes even better than those of the standard AES S-box. We believe that our S-box could be used to replace the five-bit substitution of ciphers like ASCON.

Keywords: S-box; block cipher; cellular automata; Feistel permutation; Boolean functions

1. Introduction

Cryptography today plays a leading role in the development of telecommunications. Symmetric encryption is an important part of modern cryptography. It must allow two parties who share a common secret key to exchange enciphered data. The encrypted data should look like random bits from the perspective of an external attacker who does not have the key.

There are two main families of symmetric ciphers: stream ciphers and block ciphers. Stream ciphers encrypt data on the fly, bit by bit, while block ciphers treat data as a series of blocks of a specific size. It is the latter that is most used today. Advanced encryption standard (AES) and Blowfish are the best-known block cipher algorithms.

Substitution boxes (abbreviated S-boxes) are the most important nonlinear component of many block ciphers. They play the role of

https://doi.org/10.25088/ComplexSystems.35.1.89

input bits mixer and are essential for the security of the cipher. This is the part that must be designed with the greatest attention, since it is on its weakness that most attacks focus.

The S-box designer must in particular ensure that the S-box is resistant against linear [1], differential [2] and boomerang [3] attacks, which are today the main threats to the security of S-boxes.

However, it is impossible to study all possible n-bit S-boxes, starting from a certain n. Indeed, an S-box can be seen as a permutation on the discrete set $0, 2^n - 1$. So there is a total of $2^n!$ possible S-boxes. For eight-bit S-boxes like AES, this represents approximately $8.58 \cdot 10^{506}$ possible S-boxes. For 10-bit S-boxes like the one we propose, there are $5.42 \cdot 10^{2639}$ possible permutations. It is therefore absolutely unthinkable to study them all exhaustively.

Current constructions of S-boxes rely on algebraic approaches by using properties of finite fields, like AES. We propose here a combinatorial construction with an S-box based on a Feistel construction of depth 11. This construction consists of three layers of bijective affine transformation and eight layers of permutations by a uniform binary cellular automaton (CA) of dimension 1, with a well-chosen local function considered as a Boolean function. This function is used as a pseudorandom permutation.

First, we discuss the related work in this area. In Section 3, we recall the definitions of Boolean functions and uniform cellular automata (CAs). We give some important properties, which will be useful in the rest of this paper. Next, we recall Feistel constructions and how they can be used to generate cryptographically secure random permutations in Section 4 (in particular, thanks to the Luby–Rackoff theorem). In Section 5, we explain how we generate a 10-bit S-box from a Feistel construction based on uniform CA permutations. Subsequently, we carry out the cryptanalysis of the S-box thus obtained, according to different criteria (Section 6). Finally, we conclude with the possible use of this type of construction and suggestions to generate larger S-boxes.

2. Related Work

There are many research papers that propose new methods for generating S-boxes. Most focus on eight-bit S-boxes, although some offer smaller S-boxes.

The generation of $2n$-bit S-boxes from n-bit subpermutations has already been considered in the literature, either by Feistel or MISTY constructions [4, 5].

Many other methods have also been considered. Burnett et al. proposed a heuristic method to generate MARS-like S-boxes [6].

Methods based on genetic programming were used to successively select the S-boxes with the best cryptographic properties [7, 8]. Many stochastic methods have been proposed to generate S-boxes with the best possible properties [9]. Others have already thought about using chaotic functions to generate S-boxes [10].

A method to generate S-boxes of arbitrary size allowing to maximize their degree, their nonlinearity and to minimize their differential uniformity was proposed in [11].

The AES S-box [12] nevertheless remains to this day the security reference for eight-bit S-boxes. This S-box is a finite field polynomial construction (and the security standard). Many other scientific papers also propose S-boxes based on polynomial constructions [13, 14].

A further property currently sought in S-box design is energy efficiency, characterized by the ability to perform the required permutations while minimizing computational resource utilization. This involves designing the S-box according to the internal functioning of the processor's logic gates. Such S-boxes have been proposed on eight bits [15, 16].

Other eight-bit S-boxes have been proposed to be easily adaptable to field programmable gate array (FPGA) [17].

Other ciphers, on the contrary, rely on smaller S-boxes to further reduce energy consumption [18, 19]. However, such designs cannot be done without a reduction in the cryptographic quality of the S-box, and therefore in the security of the cipher [20].

To our knowledge, no paper has been published on the generation of 10-bit S-boxes.

As indicated in Section 5, the construction of S-boxes from functions validating the NIST FIPS 140-2 test has already been explored by [21], but CAs were not tested then.

The search for Boolean functions with chaotic behavior was first studied by Wolfram in 1983 [22]. He discovered that the CA rule 30 presents the best chaotic evolution. However, the Siegenthaler bound states that functions with three variables are not suitable for cryptography [23].

The classification of Boolean functions has already been done according to multiple criteria [24, 25]; we do not bring much new in this area except perhaps their selection based on a random test.

The use of CAs for cryptography is not recent [26]. Gutowitz proposed in 1993 the use of CAs for the block cipher [27]. Several papers have already been proposed to construct S-boxes or hash-functions from such automata [28, 29]. However, to our knowledge, no one has yet designed an S-box from a CA-based Luby–Rackoff construction.

https://doi.org/10.25088/ComplexSystems.35.1.89

3. Definitions and Notation

3.1 Uniform Cellular Automata

CAs form a model of discrete parallel computation, composed of cells. Each cell is a finite state machine. At each discrete time step, all the cells of the CA update their state synchronously according to the states of their neighbors and their current state, following a local rule. The best-known CA is Conway's Game of Life, which is two dimensional.

We can formally define one-dimensional CAs as triples (Q, δ, N) where:

- Q is a finite set of states; here the set of states is $\{0, 1\}$, the Boolean values.
- δ is the local transition function $Q^n \longrightarrow Q$, called the *rule*. n is the *arity* of the rule. Here we use Boolean functions as local transition rules.
- $N \subseteq \mathbb{Z}$ is the finite neighborhood, $\sharp N = a$ being the size of the CA.

Here we are interested in one-dimensional CAs. They are a finite ring of cells, each containing a Boolean value. At each discrete time step, each cell is updated according to itself and its neighbors. It is possible to have several local rules within the same CA: this type of CA is called *nonuniform*.

A CA is said to be *uniform* if it applies the same local transition rule for all its cells.

Here we consider uniform CAs, with an n-variable Boolean local transition function (or rule) δ. At each discrete time step, each cell is modified according to the result of δ on itself and its $n - 1$ neighbors. In the case of cells at the edge of the automaton, we arbitrarily choose to count the cells on the other edge as neighbors, thus forming a ring, as shown in Figure 1.

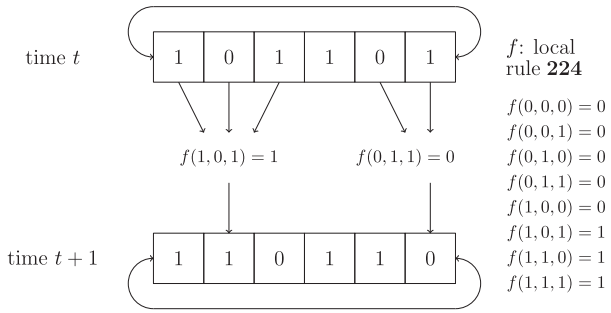

Figure 1. Example of a one-dimensional uniform CA with the single three-bit local rule 224. To compute the next value of a cell on the border, we consider the cell on the other edge as neighboring this one.

By choosing a good local rule, it is possible to create a pseudorandom bit generator, as shown in [30]. Generally speaking, certain local rules are capable of producing a chaotic effect on the states taken by designated cells at successive time steps. The best-known rule is the three-bit rule 30, as shown by Wolfram in 1983 [22]. Most pseudorandom bit generators, however, use linear-feedback shift registers (LFSRs) [31].

However, uniform automata can have short and therefore nonchaotic cycles, depending on the inputs. For example, a uniform automaton filled only with ones will only be able to take successive ones or zeros as the value at the next time step. It is therefore advisable to be careful with these particular inputs when using a uniform CA for cryptographic applications.

3.2 Boolean Functions

A Boolean function is a function that takes n Boolean values as input and returns a single Boolean value as output, n being the number of variables in the function. The local transition functions of CAs can be viewed as Boolean functions.

3.2.1 Truth Table

According to Wolfram's numbering, Boolean functions are characterized by their truth table, which lists the outputs corresponding to the inputs, unique in the set of functions with a given number of variables.

Example 1. In the set of three-bit functions, rule 30 is expressed as 00 011 110 in binary. Starting with the rightmost least significant bit, this means that $f(0, 0, 0) = 0$, $f(0, 0, 1) = 1$, $f(0, 1, 0) = 1$ and so on.

There are 2^{2^n} n-variable Boolean functions. A convenient way to represent them is given by the algebraic normal form, as presented next.

3.2.2 Algebraic Normal Form

Definition 1. Any n-variable Boolean function f can be expressed by a unique binary polynomial, called the algebraic normal form (ANF):

$$f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u \left(\prod_{i=1}^{n} x_i^{u_i} \right), \; a_u \in \mathbb{F}_2,$$

u_i is the i^{th} projection of u, and x_i is bit i of input x.

Example 2. The ANF of rule 30 is $x_0 \oplus x_1 \oplus x_2 \oplus x_1 \cdot x_2$.

https://doi.org/10.25088/ComplexSystems.35.1.89

Example 3. The ANF of the three-variable function $\chi : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2$ used by Keccak [32] is $\chi(x_0, x_1, x_2) = x_0 \oplus x_1 \cdot x_2 \oplus x_2$. Its rule number is 210.

Definition 2. The algebraic degree of a function f counts the number of variables in the largest monomial $x_0^{u_1} \ldots x_{n-1}^{u_n}$ of its ANF.

Example 4. The largest monomial of rules 30 and 210 is $x_1 \cdot x_2$; their degree is 2.

A function f is said to be *nonlinear* if and only if its degree is at least 2.

3.2.3 Hamming Weight

Definition 3. The Hamming weight of a Boolean function f, written $w_h(f)$, is the number of $x \in \mathbb{F}_2^n$ such that $f(x) = 1$.

Definition 4. An n-variable Boolean function f is balanced if and only if $w_h(f) = 2^{n-1}$ (it returns as many ones as zeros).

3.2.4 Correlation Immunity

Definition 5. An n-variable Boolean function f is k-correlation immune, $1 \leq k \leq n$, if and only if for any binary random input $x = x_0, \ldots, x_{n-1}$, $f(x)$ is statistically independent from any subset of size k of x.

The Walsh–Hadamard transform [33] is an essential tool for analyzing the statistical properties of a Boolean function. The Walsh–Hadamard transform of a Boolean function f is defined by:

$$\hat{f}(\omega) = \sum_{x=0}^{2^n-1} (-1)^{f(x) \oplus x \cdot \omega} \tag{1}$$

where $x \cdot \omega = \sum_{i=0}^{n-1} x_i \cdot \omega_i$ denotes the dot product of the two binary vectors.

Theorem 1. An n-variable Boolean function f is k-order correlation immune, $1 \leq k \leq n$ if and only if for every $\omega \in \mathbb{F}_2^n$ such that $1 \leq w_h(\omega) \leq k$, $\hat{f}(\omega) = 0$.

Xiao and Massey proved Theorem 1 in [34]. A Boolean function that is both balanced and correlation immune at order k is said to be *resilient at order k*.

3.2.5 Strict Avalanche Criterion

Definition 6. An n-variable Boolean function f satisfies the strict avalanche criterion (SAC) if and only if $\forall\, i \in 1, n$; flipping bit i of the input x results in the output $f(x)$ being changed for exactly half of the inputs x.

The strict avalanche criterion is particularly interesting in the cryptographic context since it makes it difficult to infer input from output. It makes the Boolean function "chaotic."

4. Feistel Constructions

The Feistel construction [35] is a method for constructing secure pseudorandom bijective permutations from pseudorandom functions. It is also possible to use a Feistel network to construct pseudorandom functions that are not permutations, but here we are looking to construct permutations.

The Feistel network, from a certain depth, guarantees the computational indistinguishability of its pseudorandom permutation from a random permutation.

Definition 7. A function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ is said to be a pseudorandom function (PRF) if its output is computationally difficult to distinguish from a random output.

Definition 8. A pseudorandom function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ is called a pseudorandom permutation (PRP) if and only if it is bijective.

As shown in Figure 2, the Feistel construction creates a block permutation function of size n. It is made up of a stack of layers, each composed of PRP f_i of input and output size $n/2$. We call *depth* the number of subpermutations f_i.

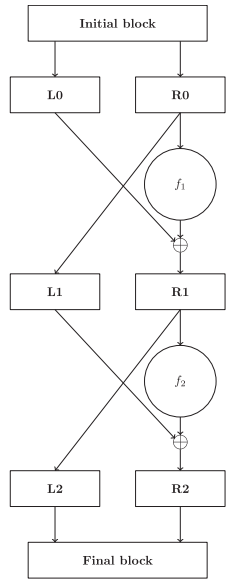

Figure 2. Example of Feistel construction of depth 2. f_1 and f_2 are PRPs.

https://doi.org/10.25088/ComplexSystems.35.1.89

Luby and Rackoff proved in [36] that the output of the Luby–Rackoff function is computationally indistinguishable from a random output as long as the depth of the network is at least four, even for an adversary who knows the input (i.e., known-plaintext-attack (KPA)).

As shown in [37], a Feistel construction with a depth of at least seven returns an output that is computationally indistinguishable from a random output for an adversary able to choose the input value (i.e., chosen-plaintext-attack (CPA)); that is to say, there is no probabilistic algorithm that is capable of making the distinction in polynomial time.

5. Our 10-Bit S-Box from a Cellular Automaton–Based Feistel Construction

5.1 Architecture of the Feistel Construction

The permutation function generated by the Feistel construction allows us to construct an S-box. We pass the $2^{10} = 1024$ possible inputs to the function; the output then gives us the S-box. The latter must validate several security requirements, explained in Section 6. Another important property to respect is bijectivity, which makes it possible to invert the S-box and therefore to proceed with decryption. In short, it is necessary that for the S-box $S : \mathbb{F}_2^{10} \longrightarrow \mathbb{F}_2^{10}$:

$$\forall\, x, y \in \mathbb{F}_2^{10}, \quad S(x) = S(y) \Longrightarrow x = y. \tag{2}$$

In our network, we use a five-cell CA as a pseudorandom permutation f_i; its output is evaluated after a single time step on the input. The automaton has only one local transition function with five variables, which we detail in Section 5.2.

However, as explained in Section 3.1, a uniform CA will fail to return chaotic output for some particular inputs that are regular. If we used only this type of CA for intermediate permutation functions f_i, some inputs would still return a predictable result, for example, $S(0)$ or $S(1023)$, which would return 0 or 1023 (0b1111111111).

Fortunately, [38] tells us that it is possible to replace certain pseudorandom permutations by "almost" pairwise independent permutations, that is, permutations whose output is almost uniformly distributed for any two given inputs. An affine function $f_{a,b}(x) = a \cdot x + b$ satisfies these requirements: its values on any given two inputs is almost uniformly distributed. So that the permutation is bijective, we chose a such that $\gcd(a, 2^{10}) = 1$.

All affine functions are expressed modulo $2^{10} = 1024$. Figure 3 schematizes our Luby–Rackoff construction. Table 1 shows the algebraic degree after each round.

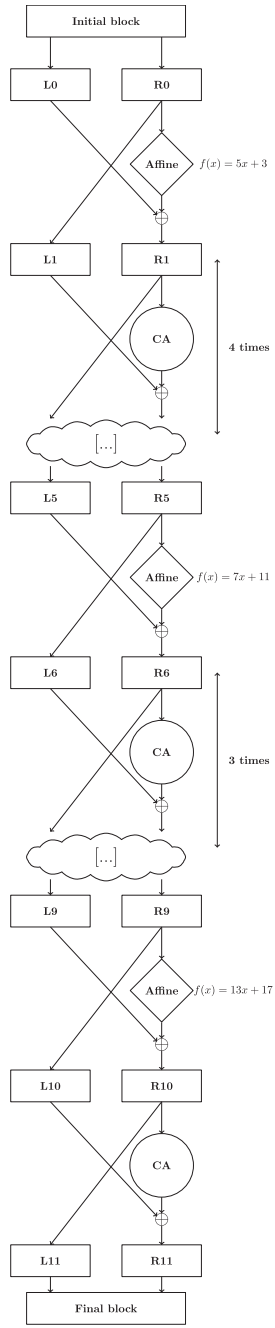

Figure 3. Selected Feistel construction, with 11 layers. The "CA" functions correspond to the output of the five-cell CA after one time step. Affine functions are expressed modulo $2^{10} = 1024$.

https://doi.org/10.25088/ComplexSystems.35.1.89

Round	1	2	3	4	5	6	7	8	9	10	11
Minimum degree	1	4	4	4	4	8	8	8	8	8	8
Maximum degree	9	9	9	9	9	9	9	9	9	9	9

Table 1. Minimum and maximum algebraic degree after each of our Feistel network rounds.

The Luby–Rackoff construction we chose to generate our 10-bit S-box consists of the following 11 layers:

1. A first layer uses the affine function $f_{5,3}(x) = 5.\,x + 3 \bmod 2^{10}$.
2. Next come four layers using the five-bit CA that will be defined in Section 5.2 as the pseudorandom permutation.
3. The next layer uses the affine function $f_{7,11}(x) = 7.\,x + 11 \bmod 2^{10}$.
4. Next we have three CA layers.
5. We have another affine layer, $f_{13,17}(x) = 13.\,x + 17 \bmod 2^{10}$.
6. Finally, a last layer reuses the five-bit CA.

5.2 Construction of the Local Pseudorandom Permutation with a Cellular Automaton

5.2.1 Construction of the Cellular Automaton

We are looking for a CA that takes as input the value to be permuted and that returns the result of the permutation. For this, we build a CA in a ring of five cells. To perform the permutation, we assign to the cells of the ring the value to be permuted, then we return the value of the CA after a single time step. We chose this construction because there are no four-variable Boolean functions that have the right cryptographic properties and that allow us to create a bijective CA.

5.2.2 Basic Properties of Local Transition Rule

There are a total of $2^{2^5} = 2^{32} = 4\,294\,967\,296$ five-bit local transition Boolean functions. The papers [22, 23, 30] fortunately give us some ideas for selecting the Boolean functions most likely to introduce "chaos" into the output of the CA.

Let us start by keeping only balanced Boolean functions, as explained by Definition 4. There are then $\binom{32}{16} = 601\,080\,390$ functions left.

We then eliminate the functions that are not first-order correlation immune, as explained in Definition 5, to keep only 807 980 rules.

We also eliminate linear functions, to keep 807 928 functions. We are satisfied with the nonlinearity property here: [39] proves that

there cannot exist bent functions (maximally nonlinear) with an odd number of variables.

Finally, we eliminate all functions that do not respect the strict avalanche criterion (SAC), explained in Definition 6, to keep 7080 local rules.

5.2.3 Selection over NIST FIPS 140-2 Randomness Test

We were inspired by [21], which gives us an original method to search for the most "chaotic" local rules. We try to create a pseudorandom generator from a uniform CA, as explained by [30], then we only keep the rules that allow us to create "good" pseudorandom bit generators.

We start by creating a ring of CAs, as explained in Section 3, of size 1024 bits. Indeed, [40] informs us that the ring must have strictly more than 1000 cells to produce a secure pseudorandom generator. Having a size equal to a power of two simply speeds up the computations.

For the seed value, we fill the ring with 1024 truly random bits, downloaded from the website www.random.org. In order to have a reproducible experience, you can find the seed at the following address: github.com/thomasarmel/cellular_automata_prng (although the seed value should not influence the results). You will also find the pseudorandom bit generator that uses a 1024-cell CA.

Next, we test each of the 7080 local rules: at each time step, we update all cells in the ring with the rule under test. We then extract the 512^{th} bit from the ring. We repeat the operation for as many bits as we wish to extract.

For each of the 7080 rules, we evaluate the pseudorandom bit generator with the NIST FIPS 140-2 test [41]. The generator must pass all tests ("Monobit," "Poker," "Runs," "Long run" and "Continuous run") out of 100 000 bits generated, which is equivalent to passing each test 39 times. This threshold of 100 000 bits is arbitrary, but more than sufficient to eliminate any statistical bias.

There then remain 53 rules that allow the ring CA to validate the NIST FIPS 140-2 pseudorandom generation test.

5.2.4 Cellular Automaton Bijectivity

Finally, we must ensure that the five-bit ring CA of the Luby–Rackoff construction is bijective. To do this, we eliminate all the local rules that do not allow us to create a bijective CA.

Let f be the local rule. Let f_a denote, with $0 \leq a < 5$, the rule corresponding to a rotations of the input variables of f. For example, $f_1(x_0, x_1, x_2, x_3, x_4) = f(x_1, x_2, x_3, x_4, x_0)$. Then the local rule f allows us to construct a bijective automaton if and only if

f is balanced, and the set of $2^5 - 1$ XOR combinations of the rules f_a are balanced, as proved by [42]. That is, if $f_0 \oplus f_1$, $f_0 \oplus f_1 \oplus f_2 \ldots$ are balanced.

Finally, only one rule remains, whose truth table numbering is 1 438 886 595 (in decimal), or

$$x_0 \cdot x_3 + x_1 \cdot x_3 + x_2 \cdot x_3 + x_3 \cdot x_4 + x_1 + x_2 + x_3 + 1$$

in its ANF form.

Table 2 shows the results of advanced tests using the dieharder suite. The generated S-box can be found in Appendix List. The source code to generate the S-box can be found at: github.com/thomasarmel/luby_rackoff_sbox_finder.

birthdays	0.8236 (P)	**operm5**	0.0044 (W)
rank_32x32	0.7126 (P)	**rank_6x8**	0.0000 (F)
bitstream	0.0000 (F)	**opso**	0.0000 (F)
oqso	0.0000 (F)	**dna**	0.0000 (F)
count_1s_str	0.0000 (F)	**count_1s_byt**	0.0000 (F)
parking_lot	0.0003 (W)	**2dsphere**	0.4735 (P)
3dsphere	0.8615 (P)	**squeeze**	0.0000 (F)
sums	0.0996 (P)	**runs**	0.3734 (P)
craps	0.0000 (F)	**mars_tsang_gcd**	0.0000 (F)
sts_monobit	0.0000 (F)	**sts_runs**	0.0000 (F)
sts_serial	0.0000 (F)	**rgb_bitdist**	0.0000 (F)
rgb_min_dist	0.7074 (P)	**rgb_permutations**	0.3524 (P)
rgb_lagged_sum	0.0000 (F)	**rgb_kstest_test**	0.0000 (F)
dab_bytedistrib	0.0000 (F)	**dab_dct**	0.0000 (F)
dab_filltree	0.4775 (P)	**dab_filltree2**	0.0000 (F)
dab_monobit2	1.0000 (F)		

Table 2. We conducted advanced tests on the 1024-cell CA random bit generator with rule 1 438 886 595 using the dieharder suite. For the same test using different ntup (tuple length for test), we kept the best p-value result. All results are rounded to four digits after the decimal point. If this generator can produce "chaotic" output, it is, however, not well suited for cryptographic random bit generation, due to the temporal dependence between each successive bit (P = passed, W = weak, F = failed).

6. Cryptanalysis

Here we propose the cryptanalysis of the specific S-box that we generated from the construction in Section 5. In order to have a more systematic analysis of the security of S-boxes generated from Feistel networks, the reader can refer to [5].

6.1 S-Box Analysis Criteria

The main properties expected of an S-box are given in [43]. Since an S-box is the only nonlinear component of a block cipher algorithm, it must be able to resist linear [1] and differential [2] cryptanalysis. We will also discuss resistance to the boomerang attack [3].

An S-box must also be bijective, but this was already addressed in Section 5.

To quantify the security of our S-box, we will compare it to the AES S-box [12], which is the industry standard for security today. The latter has a dimension of eight bits, but we have not found any 10-bit S-box that is used in practice. When necessary, we will therefore adapt our metrics to compare them to an S-box of a different size. We will also compare certain metrics to other S-boxes presented in the literature. We particularly used the SageMath language to compute certain metrics, the latter offering a library dedicated to the cryptographic properties of S-boxes (doc.sagemath.org/html/en/reference/cryptography/sage/crypto/sbox.html). Sage notably allowed us to compute algebraic complexity, nonlinearity, linear and differential approximation probabilities, differential uniformity and boomerang uniformity.

6.2 Algebraic Degree

It is possible to represent an S-box by its component functions. For an S-box of size n, $S : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$, then it is possible to write $S(x_0, x_1, \ldots, x_{n-1}) = (y_0, y_1, \ldots, y_{n-1})$.

The component functions are then the $2^n - 1$ Boolean functions $f_i : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ corresponding to the Boolean functions $f_i(x_0, x_1, \ldots, x_{n-1}) = y_i$, for $i \in 1, n$ and all their XOR combinations (e.g., $f_{10}(x_0, x_1, \ldots, x_9) = y_0 \oplus y_1$).

As explained by Definition 1, any Boolean function can be expressed in its ANF form. A possible attack against an S-box consists of trying to approximate its value by component functions of low degree; this is a low-order approximation attack [44]. A high minimum algebraic degree, as explained in Definition 9, makes it possible to protect against this type of attack.

Definition 9. The minimum algebraic degree of an S-box is the minimum degree of its component functions. The maximum algebraic degree is the maximum degree of the component functions.

Our S-box has a minimum algebraic degree of 8 and a maximum degree of 9. In comparison, the minimum and maximum degree of the AES S-box is 7 (having a larger S-box gives us an advantage).

Table 3 provides the ANF monomial count for each of the S-box output bits.

https://doi.org/10.25088/ComplexSystems.35.1.89

Bit	0	1	2	3	4	5	6	7	8	9
Deg	9	9	9	9	9	9	9	9	9	9
♯ mon.	509	528	498	506	505	479	494	495	527	499

Table 3. Algebraic degree and ANF count for S-box output bits (excluding constant monomial).

6.3 Algebraic Complexity

The algebraic complexity of an S-box defines its ability to resist interpolation attacks [45].

Definition 10. The algebraic complexity (AC) of an n-bit S-box S is the number of monomials in the univariate polynomial representation of S such that

$$S(x) = a_0 + a_1 \cdot x + \cdots + a_{2^n-1} \cdot x^{2^n-1}. \tag{3}$$

The algebraic complexity of our S-box is **1023**, which is the maximum possible. The algebraic complexity of the AES S-box is 255, which is also the highest possible value.

6.4 Nonlinearity

Strong nonlinearity [46] allows the S-box to resist linear cryptanalysis. It is defined as the minimum nonlinearity of each of the component functions.

Definition 11. For an n-variable Boolean function f_i, the nonlinearity N_{f_i} is given by

$$N_{f_i} = 2^{n-1} - \frac{1}{2} \max_{\omega \in \mathbb{F}_2^n} \left| \hat{f}_i(\omega) \right| \tag{4}$$

with $\hat{f}_i$ the Walsh–Hadamard transform of f_i, as defined in Section 3.2.4.

The nonlinearity of our S-box is **434**. It is difficult to compare with the nonlinearity of the AES S-box (112), because the two S-boxes do not have the same size.

To provide a meaningful comparison, we recall that the theoretical upper bound for nonlinearity of an n-bit Boolean function is $2^{n-1} - 2^{(n/2)-1}$ for even n, which equals 496 when $n = 10$ and 120 when $n = 8$. The AES S-box, which is 8×8, achieves a nonlinearity of 112, corresponding to about $112/120 = 93.3\%$ of the optimal value (unreachable for a bijective S-box). Our S-box achieves $434/496 = 87.5\%$ of the theoretical optimum for $n = 10$, which is a little less than AES but better than many S-boxes presented in the literature. For both the AES S-box and ours, however, it is not possible to express the value of one of the output bits as a function of a linear

combination of the input bits with a probability $\geq 60\%$ (56.25% for the AES S-box and 57.62% for our S-box). We used the GitHub repository github.com/PoustouFlan/SUnbox to obtain these values.

6.5 Strict Avalanche Criterion

The notion of SAC for the design of S-boxes was first introduced in 1985 by [47]. To satisfy the SAC, half of the output bits must be modified when a single input bit is modified. For an S-box, the bits of the SAC dependency matrix must be close to the ideal value of 0.5.

Table 4 gives the SAC dependency matrix of the proposed S-box. We used the GitHub repository github.com/abrari/block-cipher-testing to compute these values.

	0	1	2	3	4	5	6	7	8	9
0	0.51	0.44	0.48	0.48	0.48	0.45	0.47	0.48	0.48	0.50
1	0.54	0.52	0.50	0.51	0.53	0.53	0.48	0.48	0.53	0.50
2	0.52	0.48	0.54	0.48	0.53	0.52	0.48	0.50	0.54	0.49
3	0.51	0.54	0.50	0.50	0.50	0.53	0.46	0.51	0.50	0.51
4	0.51	0.52	0.51	0.46	0.48	0.52	0.52	0.54	0.54	0.54
5	0.46	0.48	0.48	0.50	0.52	0.51	0.48	0.48	0.50	0.47
6	0.51	0.49	0.49	0.54	0.50	0.49	0.50	0.52	0.51	0.57
7	0.46	0.50	0.47	0.50	0.51	0.48	0.48	0.50	0.54	0.50
8	0.49	0.50	0.48	0.50	0.47	0.49	0.55	0.48	0.48	0.52
9	0.48	0.49	0.50	0.47	0.50	0.52	0.52	0.55	0.53	0.49

Table 4. SAC dependency matrix of the constructed S-box. Each row represents the modified input bit, and each column the impact on the output bit. For example, flipping the first input bit will change the first output bit 51% of the time.

Table 5 compares the average value as well as the extreme values of our dependency table to those of the AES S-box.

	Average	Minimum	Maximum
AES	0.50	0.45	0.56
Proposed	0.50	0.44	0.57

Table 5. SAC dependency matrix comparison between AES and proposed S-box.

Our average value is good, and the extreme values are almost as good as those of the AES S-box.

6.6 Bit Independence Criterion Parameter

The concept of bit independence criterion (BIC) was first introduced in [47].

Definition 12. We say that an n-bit S-box S satisfies the BIC if $\forall\, i, j, k \in 1, n$, with $i \neq j$; inverting the k^{th} bit of the input changes the i^{th} and the j^{th} output bits independently.

The metric we use for S-boxes is called the *bit independence criterion parameter*, which measures how far an S-box is from validating the BIC. This distance is between 0 and 1, the closer to 0 being the better.

To compute this parameter, we need to know the BIC parameter between two output bits i and j. The latter is defined as the maximum correlation coefficient between output bits i and j after inversion of input bit k, for all k.

The BIC parameter of an S-box is the maximum value of the BIC parameter of output bits i and j, for all combinations of i and j such that $i \neq j$.

The BIC parameter of our S-box is **0.124**. For comparison, the one of the AES S-box is 0.134. Our BIC parameter is therefore better than that of the AES S-box, and also better than other S-boxes proposed in the scientific literature. For example, the S-box of the block cipher PRESENT [18] has a BIC parameter of 1.

We thank the author of github.com/abrari/block-cipher-testing for the code that allowed us to compute the BIC of our S-box.

6.7 Linear Approximation Probability

The linear approximation probability (LAP) gives us an indication of how resistant our S-box is to linear cryptanalysis [1]. It is computed, for an n-bit S-box S, by the maximum correlation between $x.\alpha$ and $S(x).\beta$, $\forall\, \alpha, \beta \in 0, 2^n - 1$ (except when $\alpha = \beta = 0$).

The LAP can be derived from the linear approximation table (LAT) as described in [48]. We have

$$\mathrm{LAT}(\alpha, \beta) = \frac{\sharp\{x \in \mathbb{F}_2^n \mid \alpha \cdot x = \beta \cdot S(x)\}}{2^{n-1}} - 1. \tag{5}$$

The maximum magnitude of this correlation is defined as the maximum correlation value from the table, except at the coordinate (0, 0) (for which the correlation is logically 1):

$$\epsilon = \max_{\alpha, \beta \neq (0,0)} \left|\hat{S}(\alpha, \beta)\right|. \tag{6}$$

The linear approximation probability is then given by the correlation potential expression:

$$\mathrm{LAP} = (2 \cdot \epsilon)^2. \tag{7}$$

The LAP of our S-box is **9.28%**, which is comparable to or even better than other S-boxes proposed in the scientific literature [49]. However, the LAP is a little bit worse than that of the AES S-box, which is 6.25%.

6.8 Differential Approximation Probability

The differential approximation probability (DAP) is determined by the XOR distribution between the input and output of an S-box. The lowest possible value guarantees the security of the S-box against differential cryptanalysis [2].

The DAP is given by the maximum value of the differential probability table.

Denote an input of the S-box by $\Delta x \in \mathbb{F}_2^n$ and an output by $\Delta y \in \mathbb{F}_2^m$, with m the output size in bits of the S-box. The probability for each Δx, Δy of an n-bit S-box differential probability table is computed by:

$$\mathrm{DP}(\Delta x \to \Delta y) = \frac{\sharp\{x \in \mathbb{F}_2^n \mid S(x) \oplus S(x \oplus \Delta x) = \Delta y\}}{2^n} \tag{8}$$

and so

$$\mathrm{DAP} = \max_{\Delta x, \Delta y} \mathrm{DP}(\Delta x \to \Delta y). \tag{9}$$

The DAP of our S-box is **1.37%**, which is better than the AES S-box, which has a DAP of 1.56%. This DAP is also better than many S-boxes presented in the literature [50].

6.9 Differential Uniformity

The differential uniformity of an S-box defines its proximity to perfect nonlinearity [51]. For an n-bit S-box S, its differential uniformity δ_S is defined by

$$\begin{aligned} \delta_S = &\max_{a, b \in \mathbb{F}_2^n, a \neq 0} \delta(a, b) = \max_{a, b \in \mathbb{F}_2^n, a \neq 0} \\ &\sharp\{x \in \mathbb{F}_2^n \mid S(x \oplus a) \oplus S(x) = b\}. \end{aligned} \tag{10}$$

Definition 13. An n-bit S-box is called almost perfect nonlinear (APN) if its differential uniformity is less than or equal to 2.

There is no known APN n-bit S-box for $n > 6$ [52]. For such S-boxes, the best known reachable differential uniformity is four.

The differential uniformity of our 10-bit S-box is **14**. Hence, our S-box demonstrates reasonably low differential uniformity and remains competitive with state-of-the-art constructions [21, 53]. However, the differential uniformity of our S-box is higher than the one of AES, which is four.

https://doi.org/10.25088/ComplexSystems.35.1.89

6.10 Boomerang Uniformity

The boomerang uniformity $\mathcal{BU}$ defines the resistance of an S-box to the boomerang attack [3], which is an improvement of differential cryptanalysis. A small $\mathcal{BU}$ value provides better resistance to the boomerang attack.

To compute the $\mathcal{BU}$ of an n-bit S-box, we start by computing the boomerang connectivity table (BCT), an $n \times n$ matrix whose entry in the $\Delta_i \in \mathbb{F}_2^n$ row and in the $\Delta_o \in \mathbb{F}_2^n$ column is given by:

$$\begin{aligned} &\mathrm{BCT}(\Delta_i, \Delta_o) = \\ &\quad \sharp\left\{x \in \mathbb{F}_2^n \,\middle|\, S^{-1}(S(x) \oplus \Delta_o) \oplus S^{-1}(S(x \oplus \Delta_i) \oplus \Delta_o) = \Delta_i\right\}. \end{aligned} \tag{11}$$

The $\mathcal{BU}$ is given by the maximum entry of the BCT, ignoring the first row and first column. The $\mathcal{BU}$ of an S-box cannot be less than its differential uniformity [54], which corresponds to $\mathcal{BU} = 4$.

Our 10-bit S-box has a $\mathcal{BU}$ of **24**, which is higher than AES ($\mathcal{BU} = 6$). Thus, our S-box's $\mathcal{BU}$ represents six times the practical optimum for even n, which remains anyway within the range observed for secure large-dimension S-box constructions [20, 21].

7. Discussion

This method of constructing 10-bit S-boxes showed security results comparable to the AES standard, now widely used in the industry. However, we believe that it might be possible to improve the quality of our S-box by modifying parameters of the Feistel network. In particular, the quality of the S-box might be improved by increasing the network depth. Changing the parameters of the affine functions would result in different S-boxes, but we expect the security parameters to be roughly equivalent.

We can also consider building even larger S-boxes, provided that the number of variables n is even and $n/2$ is odd. For example, we could consider building S-boxes with 14 or even 18 variables. However, this would require selecting "good" Boolean functions with seven or nine variables, which represents a significant computational challenge. Indeed, the number of possible n-bit Boolean functions is 2^{2^n}. So, for example, there exist $1.34 \cdot 10^{154}$ 9-bit Boolean functions.

Our 10-bit S-box could, for example, be used in an ASCON-type sponge network [19]. This cipher performs permutations on 320-bit blocks, and for this purpose uses a 5-bit S-box on 64 sub-blocks. We believe that the quality and therefore the security of the permutation would be improved, but the algorithmic complexity would be increased. New security bounds should be computed in order to determine the number of rounds required [55]. An example of a possible ASCON implementation with our S-box (in Rust) is available at:

github.com/thomasarmel/sponges/blob/sbox_10/ascon/src/lib.rs#L100. On our 13th Gen Intel Core i7-13700H 5 GHz CPU, the modified **round**() function is, however, 10 to 15 times slower than the original one using ASCON's original S-box but has superior cryptographic quality. It would also be more complex to propose a constant-time implementation of the modified cipher.

We provide a performance comparison of the execution time of a constant-time implementation of our S-box versus ASCON's S-box. With all compilation optimizations, our constant-time S-box is 343 times slower than ASCON's S-box (22.97 ns versus 0.067 ns): github.com/thomasarmel/ascon_10_bits_sbox_comparator. These results illustrate that our S-box will not perform well for a constant-time implementation.

8. Conclusion

In this paper, we proposed a new 10-bit S-box from a Feistel construction based on uniform cellular automata permutations and carried out the cryptanalysis. In particular, we evaluated its robustness against linear, differential and boomerang attacks. We showed that our S-box has security that is comparable to or even better than other S-boxes presented in the scientific literature. In particular, the security evaluations were comparable to and sometimes even better than those of the AES S-box, which is today the widely used standard.

To our knowledge, no method for constructing 10-bit S-boxes has ever been proposed in the scientific literature. Our method can be extended for the construction of n-bit S-boxes, given that n is even and $n/2$ is odd.

Acknowledgments

We would like to thank Jean-Charles Regin (www.constraint-programming.com/people/regin) for kindly lending us his computing machines.

This work has been supported by a government grant managed by the Agence Nationale de la Recherche under the Investissement d'avenir program, reference ANR-17-EURE-004.

Appendix A. Generated 10-Bit S-Box

Table A.1. Generated S-box input/output table (*continues*).

0x000	0x090	**0x100**	0x2d8	**0x200**	0x238	**0x300**	0x05e
0x001	0x15c	**0x101**	0x354	**0x201**	0x137	**0x301**	0x12a
0x002	0x1c0	**0x102**	0x2fa	**0x202**	0x005	**0x302**	0x04a
0x003	0x292	**0x103**	0x36b	**0x203**	0x147	**0x303**	0x15a
0x004	0x101	**0x104**	0x0c6	**0x204**	0x0a4	**0x304**	0x1e7
0x005	0x309	**0x105**	0x2c7	**0x205**	0x3cc	**0x305**	0x18f
0x006	0x228	**0x106**	0x3b9	**0x206**	0x384	**0x306**	0x293
0x007	0x36f	**0x107**	0x10a	**0x207**	0x3c1	**0x307**	0x180
0x008	0x20f	**0x108**	0x35d	**0x208**	0x071	**0x308**	0x0b9
0x009	0x28e	**0x109**	0x042	**0x209**	0x023	**0x309**	0x070
0x00a	0x2fc	**0x10a**	0x240	**0x20a**	0x303	**0x30a**	0x094
0x00b	0x379	**0x10b**	0x1b9	**0x20b**	0x118	**0x30b**	0x188
0x00c	0x214	**0x10c**	0x2da	**0x20c**	0x366	**0x30c**	0x251
0x00d	0x085	**0x10d**	0x0d6	**0x20d**	0x208	**0x30d**	0x167
0x00e	0x2be	**0x10e**	0x0bc	**0x20e**	0x215	**0x30e**	0x37f
0x00f	0x269	**0x10f**	0x06e	**0x20f**	0x177	**0x30f**	0x3c9
0x010	0x392	**0x110**	0x192	**0x210**	0x172	**0x310**	0x09e
0x011	0x358	**0x111**	0x356	**0x211**	0x38e	**0x311**	0x00e
0x012	0x361	**0x112**	0x237	**0x212**	0x01a	**0x312**	0x0e0
0x013	0x2c9	**0x113**	0x0f6	**0x213**	0x07c	**0x313**	0x2e6
0x014	0x23b	**0x114**	0x020	**0x214**	0x372	**0x314**	0x009
0x015	0x2ec	**0x115**	0x069	**0x215**	0x2fd	**0x315**	0x02f
0x016	0x1b8	**0x116**	0x0d7	**0x216**	0x0aa	**0x316**	0x38c
0x017	0x055	**0x117**	0x24a	**0x217**	0x1c4	**0x317**	0x011
0x018	0x091	**0x118**	0x225	**0x218**	0x3a6	**0x318**	0x33e
0x019	0x1a8	**0x119**	0x322	**0x219**	0x1f5	**0x319**	0x170
0x01a	0x08a	**0x11a**	0x07f	**0x21a**	0x181	**0x31a**	0x015
0x01b	0x282	**0x11b**	0x220	**0x21b**	0x1b0	**0x31b**	0x342
0x01c	0x34f	**0x11c**	0x0ed	**0x21c**	0x035	**0x31c**	0x030
0x01d	0x36e	**0x11d**	0x216	**0x21d**	0x35c	**0x31d**	0x09c
0x01e	0x32e	**0x11e**	0x16e	**0x21e**	0x186	**0x31e**	0x000
0x01f	0x0af	**0x11f**	0x2e5	**0x21f**	0x247	**0x31f**	0x2bb
0x020	0x212	**0x120**	0x1fc	**0x220**	0x0b1	**0x320**	0x0c4
0x021	0x37c	**0x121**	0x380	**0x221**	0x140	**0x321**	0x202
0x022	0x0f8	**0x122**	0x22f	**0x222**	0x2a2	**0x322**	0x2a6
0x023	0x113	**0x123**	0x32d	**0x223**	0x3e7	**0x323**	0x2ad
0x024	0x2b0	**0x124**	0x145	**0x224**	0x3f4	**0x324**	0x3cd
0x025	0x1d9	**0x125**	0x0e1	**0x225**	0x211	**0x325**	0x3e9
0x026	0x362	**0x126**	0x0de	**0x226**	0x3d2	**0x326**	0x14e
0x027	0x2cb	**0x127**	0x03b	**0x227**	0x320	**0x327**	0x3c2
0x028	0x099	**0x128**	0x06d	**0x228**	0x025	**0x328**	0x364
0x029	0x1e8	**0x129**	0x04d	**0x229**	0x27a	**0x329**	0x179
0x02a	0x29b	**0x12a**	0x067	**0x22a**	0x270	**0x32a**	0x00b

Table A.1. Generated S-box input/output table (*continues*).

0x02b	0x26f	**0x12b**	0x1ee	**0x22b**	0x08b	**0x32b**	0x2a0
0x02c	0x11c	**0x12c**	0x1e3	**0x22c**	0x1f9	**0x32c**	0x319
0x02d	0x075	**0x12d**	0x0ef	**0x22d**	0x331	**0x32d**	0x00d
0x02e	0x26c	**0x12e**	0x0b7	**0x22e**	0x0ec	**0x32e**	0x11d
0x02f	0x1bb	**0x12f**	0x285	**0x22f**	0x3fb	**0x32f**	0x3b4
0x030	0x3ec	**0x130**	0x2a3	**0x230**	0x218	**0x330**	0x2b2
0x031	0x027	**0x131**	0x1b7	**0x231**	0x1d5	**0x331**	0x24b
0x032	0x393	**0x132**	0x084	**0x232**	0x033	**0x332**	0x3e1
0x033	0x2e8	**0x133**	0x2c0	**0x233**	0x29a	**0x333**	0x3ef
0x034	0x3e4	**0x134**	0x391	**0x234**	0x3c7	**0x334**	0x22e
0x035	0x24e	**0x135**	0x274	**0x235**	0x04f	**0x335**	0x25f
0x036	0x29e	**0x136**	0x258	**0x236**	0x1a0	**0x336**	0x115
0x037	0x353	**0x137**	0x3b1	**0x237**	0x0ae	**0x337**	0x3a8
0x038	0x175	**0x138**	0x318	**0x238**	0x265	**0x338**	0x126
0x039	0x34e	**0x139**	0x30a	**0x239**	0x3bc	**0x339**	0x2ed
0x03a	0x19a	**0x13a**	0x223	**0x23a**	0x111	**0x33a**	0x3fd
0x03b	0x03e	**0x13b**	0x002	**0x23b**	0x185	**0x33b**	0x15e
0x03c	0x0d3	**0x13c**	0x357	**0x23c**	0x3be	**0x33c**	0x25d
0x03d	0x326	**0x13d**	0x0d2	**0x23d**	0x3d6	**0x33d**	0x1b3
0x03e	0x2b6	**0x13e**	0x1cb	**0x23e**	0x026	**0x33e**	0x365
0x03f	0x226	**0x13f**	0x1dc	**0x23f**	0x0c9	**0x33f**	0x0e4
0x040	0x351	**0x140**	0x16b	**0x240**	0x2a5	**0x340**	0x191
0x041	0x2d4	**0x141**	0x1d2	**0x241**	0x38a	**0x341**	0x3ca
0x042	0x3d8	**0x142**	0x2d5	**0x242**	0x20b	**0x342**	0x003
0x043	0x35f	**0x143**	0x281	**0x243**	0x112	**0x343**	0x25a
0x044	0x161	**0x144**	0x2f3	**0x244**	0x098	**0x344**	0x1b4
0x045	0x1f3	**0x145**	0x1fb	**0x245**	0x007	**0x345**	0x14f
0x046	0x3cf	**0x146**	0x3d5	**0x246**	0x18a	**0x346**	0x04b
0x047	0x02d	**0x147**	0x0ac	**0x247**	0x23a	**0x347**	0x2a9
0x048	0x3f8	**0x148**	0x1d6	**0x248**	0x0df	**0x348**	0x296
0x049	0x117	**0x149**	0x3ee	**0x249**	0x29d	**0x349**	0x1aa
0x04a	0x1a4	**0x14a**	0x0ea	**0x24a**	0x029	**0x34a**	0x14b
0x04b	0x385	**0x14b**	0x2fe	**0x24b**	0x065	**0x34b**	0x2e3
0x04c	0x1d3	**0x14c**	0x066	**0x24c**	0x133	**0x34c**	0x1c8
0x04d	0x3b2	**0x14d**	0x13e	**0x24d**	0x338	**0x34d**	0x1e9
0x04e	0x20d	**0x14e**	0x21c	**0x24e**	0x1b6	**0x34e**	0x3f0
0x04f	0x0b2	**0x14f**	0x1c2	**0x24f**	0x019	**0x34f**	0x164
0x050	0x0c5	**0x150**	0x1df	**0x250**	0x17f	**0x350**	0x08e
0x051	0x3cb	**0x151**	0x081	**0x251**	0x1c7	**0x351**	0x26b
0x052	0x3e3	**0x152**	0x174	**0x252**	0x12e	**0x352**	0x2c4
0x053	0x29f	**0x153**	0x350	**0x253**	0x171	**0x353**	0x2b3
0x054	0x2e7	**0x154**	0x25c	**0x254**	0x151	**0x354**	0x2f8
0x055	0x0a6	**0x155**	0x235	**0x255**	0x106	**0x355**	0x3bb

Table A.1. Generated S-box input/output table (*continues*).

0x056	0x273	**0x156**	0x07a	**0x256**	0x341	**0x356**	0x1a7
0x057	0x29c	**0x157**	0x2ea	**0x257**	0x21a	**0x357**	0x3de
0x058	0x163	**0x158**	0x13d	**0x258**	0x048	**0x358**	0x3a4
0x059	0x210	**0x159**	0x09b	**0x259**	0x381	**0x359**	0x080
0x05a	0x01d	**0x15a**	0x30c	**0x25a**	0x286	**0x35a**	0x3db
0x05b	0x083	**0x15b**	0x2cf	**0x25b**	0x28f	**0x35b**	0x250
0x05c	0x19e	**0x15c**	0x2c6	**0x25c**	0x0a7	**0x35c**	0x141
0x05d	0x197	**0x15d**	0x00a	**0x25d**	0x10b	**0x35d**	0x37b
0x05e	0x279	**0x15e**	0x0fb	**0x25e**	0x39f	**0x35e**	0x1c6
0x05f	0x332	**0x15f**	0x047	**0x25f**	0x317	**0x35f**	0x1e2
0x060	0x2d0	**0x160**	0x297	**0x260**	0x246	**0x360**	0x22c
0x061	0x389	**0x161**	0x1f8	**0x261**	0x308	**0x361**	0x134
0x062	0x3b7	**0x162**	0x0da	**0x262**	0x34b	**0x362**	0x1f1
0x063	0x388	**0x163**	0x3c4	**0x263**	0x135	**0x363**	0x343
0x064	0x2dd	**0x164**	0x3f3	**0x264**	0x02b	**0x364**	0x054
0x065	0x18d	**0x165**	0x2a7	**0x265**	0x288	**0x365**	0x1e1
0x066	0x053	**0x166**	0x229	**0x266**	0x01c	**0x366**	0x093
0x067	0x37d	**0x167**	0x3ab	**0x267**	0x039	**0x367**	0x344
0x068	0x352	**0x168**	0x1a5	**0x268**	0x0b0	**0x368**	0x3c8
0x069	0x146	**0x169**	0x2b4	**0x269**	0x12c	**0x369**	0x3d1
0x06a	0x1b2	**0x16a**	0x295	**0x26a**	0x061	**0x36a**	0x2b1
0x06b	0x3d9	**0x16b**	0x31e	**0x26b**	0x156	**0x36b**	0x2b9
0x06c	0x249	**0x16c**	0x26e	**0x26c**	0x2b8	**0x36c**	0x0e2
0x06d	0x0db	**0x16d**	0x02a	**0x26d**	0x30f	**0x36d**	0x1dd
0x06e	0x2f0	**0x16e**	0x130	**0x26e**	0x014	**0x36e**	0x387
0x06f	0x3fa	**0x16f**	0x3ea	**0x26f**	0x2d9	**0x36f**	0x31a
0x070	0x125	**0x170**	0x21f	**0x270**	0x2c1	**0x370**	0x0c8
0x071	0x1e0	**0x171**	0x2ef	**0x271**	0x1f0	**0x371**	0x1a1
0x072	0x06c	**0x172**	0x280	**0x272**	0x3c0	**0x372**	0x2a8
0x073	0x3f9	**0x173**	0x0e6	**0x273**	0x0d0	**0x373**	0x198
0x074	0x073	**0x174**	0x050	**0x274**	0x267	**0x374**	0x0ee
0x075	0x1d1	**0x175**	0x0c0	**0x275**	0x0e7	**0x375**	0x234
0x076	0x122	**0x176**	0x386	**0x276**	0x30d	**0x376**	0x1d7
0x077	0x263	**0x177**	0x169	**0x277**	0x3a1	**0x377**	0x2eb
0x078	0x116	**0x178**	0x2cd	**0x278**	0x2a1	**0x378**	0x2af
0x079	0x082	**0x179**	0x0a3	**0x279**	0x2f1	**0x379**	0x398
0x07a	0x2b5	**0x17a**	0x149	**0x27a**	0x316	**0x37a**	0x378
0x07b	0x028	**0x17b**	0x324	**0x27b**	0x199	**0x37b**	0x375
0x07c	0x129	**0x17c**	0x19d	**0x27c**	0x397	**0x37c**	0x35b
0x07d	0x0b5	**0x17d**	0x21e	**0x27d**	0x3b5	**0x37d**	0x09a
0x07e	0x348	**0x17e**	0x09f	**0x27e**	0x0ce	**0x37e**	0x1ca
0x07f	0x30e	**0x17f**	0x383	**0x27f**	0x2ba	**0x37f**	0x28d
0x080	0x05a	**0x180**	0x13a	**0x280**	0x2b7	**0x380**	0x2d6

Table A.1. Generated S-box input/output table (*continues*).

0x081	0x123	**0x181**	0x27c	**0x281**	0x36a	**0x381**	0x244
0x082	0x38d	**0x182**	0x0bf	**0x282**	0x176	**0x382**	0x159
0x083	0x14d	**0x183**	0x1da	**0x283**	0x1eb	**0x383**	0x043
0x084	0x2c2	**0x184**	0x1bd	**0x284**	0x036	**0x384**	0x162
0x085	0x089	**0x185**	0x02c	**0x285**	0x010	**0x385**	0x36c
0x086	0x39d	**0x186**	0x306	**0x286**	0x34d	**0x386**	0x1e6
0x087	0x1bc	**0x187**	0x0a2	**0x287**	0x1b1	**0x387**	0x231
0x088	0x157	**0x188**	0x32f	**0x288**	0x340	**0x388**	0x193
0x089	0x096	**0x189**	0x166	**0x289**	0x205	**0x389**	0x330
0x08a	0x1ae	**0x18a**	0x132	**0x28a**	0x24f	**0x38a**	0x329
0x08b	0x268	**0x18b**	0x160	**0x28b**	0x0ad	**0x38b**	0x242
0x08c	0x064	**0x18c**	0x203	**0x28c**	0x01f	**0x38c**	0x1cf
0x08d	0x059	**0x18d**	0x307	**0x28d**	0x2e2	**0x38d**	0x241
0x08e	0x0d9	**0x18e**	0x2bd	**0x28e**	0x3e8	**0x38e**	0x336
0x08f	0x3dc	**0x18f**	0x05f	**0x28f**	0x108	**0x38f**	0x2e0
0x090	0x28b	**0x190**	0x0f0	**0x290**	0x16f	**0x390**	0x11b
0x091	0x2ac	**0x191**	0x158	**0x291**	0x2d7	**0x391**	0x0fa
0x092	0x15f	**0x192**	0x0d4	**0x292**	0x298	**0x392**	0x110
0x093	0x088	**0x193**	0x22d	**0x293**	0x3dd	**0x393**	0x178
0x094	0x33f	**0x194**	0x168	**0x294**	0x1a9	**0x394**	0x16c
0x095	0x17a	**0x195**	0x052	**0x295**	0x1ba	**0x395**	0x1ec
0x096	0x38b	**0x196**	0x264	**0x296**	0x1b5	**0x396**	0x3b6
0x097	0x08d	**0x197**	0x266	**0x297**	0x376	**0x397**	0x245
0x098	0x259	**0x198**	0x142	**0x298**	0x114	**0x398**	0x260
0x099	0x363	**0x199**	0x236	**0x299**	0x345	**0x399**	0x2e1
0x09a	0x153	**0x19a**	0x3a2	**0x29a**	0x049	**0x39a**	0x1ab
0x09b	0x2db	**0x19b**	0x35a	**0x29b**	0x0e8	**0x39b**	0x2aa
0x09c	0x367	**0x19c**	0x03c	**0x29c**	0x311	**0x39c**	0x0a5
0x09d	0x077	**0x19d**	0x28a	**0x29d**	0x3bf	**0x39d**	0x1d0
0x09e	0x078	**0x19e**	0x328	**0x29e**	0x00c	**0x39e**	0x006
0x09f	0x315	**0x19f**	0x1bf	**0x29f**	0x0c1	**0x39f**	0x06b
0x0a0	0x060	**0x1a0**	0x3a9	**0x2a0**	0x0a1	**0x3a0**	0x390
0x0a1	0x3b8	**0x1a1**	0x299	**0x2a1**	0x278	**0x3a1**	0x152
0x0a2	0x3f5	**0x1a2**	0x200	**0x2a2**	0x02e	**0x3a2**	0x0cf
0x0a3	0x3af	**0x1a3**	0x3eb	**0x2a3**	0x368	**0x3a3**	0x001
0x0a4	0x194	**0x1a4**	0x254	**0x2a4**	0x040	**0x3a4**	0x300
0x0a5	0x15b	**0x1a5**	0x294	**0x2a5**	0x1c1	**0x3a5**	0x1c9
0x0a6	0x044	**0x1a6**	0x3b3	**0x2a6**	0x3e0	**0x3a6**	0x120
0x0a7	0x3aa	**0x1a7**	0x16d	**0x2a7**	0x37e	**0x3a7**	0x232
0x0a8	0x301	**0x1a8**	0x2c3	**0x2a8**	0x1a2	**0x3a8**	0x131
0x0a9	0x11e	**0x1a9**	0x127	**0x2a9**	0x2c5	**0x3a9**	0x183
0x0aa	0x056	**0x1aa**	0x1e5	**0x2aa**	0x17e	**0x3aa**	0x0ff
0x0ab	0x2bc	**0x1ab**	0x1d4	**0x2ab**	0x148	**0x3ab**	0x21b

Table A.1. Generated S-box input/output table (*continues*).

0x0ac	0x11f	**0x1ac**	0x31d	**0x2ac**	0x20a	**0x3ac**	0x0f2
0x0ad	0x271	**0x1ad**	0x10f	**0x2ad**	0x33b	**0x3ad**	0x1f2
0x0ae	0x333	**0x1ae**	0x2f7	**0x2ae**	0x0f7	**0x3ae**	0x03a
0x0af	0x230	**0x1af**	0x1db	**0x2af**	0x0fd	**0x3af**	0x2f5
0x0b0	0x291	**0x1b0**	0x30b	**0x2b0**	0x196	**0x3b0**	0x187
0x0b1	0x12d	**0x1b1**	0x13f	**0x2b1**	0x05d	**0x3b1**	0x27e
0x0b2	0x045	**0x1b2**	0x314	**0x2b2**	0x3c6	**0x3b2**	0x3da
0x0b3	0x355	**0x1b3**	0x04c	**0x2b3**	0x01e	**0x3b3**	0x079
0x0b4	0x374	**0x1b4**	0x23f	**0x2b4**	0x219	**0x3b4**	0x360
0x0b5	0x038	**0x1b5**	0x243	**0x2b5**	0x34c	**0x3b5**	0x0be
0x0b6	0x092	**0x1b6**	0x204	**0x2b6**	0x20e	**0x3b6**	0x144
0x0b7	0x03f	**0x1b7**	0x37a	**0x2b7**	0x34a	**0x3b7**	0x3d4
0x0b8	0x3a7	**0x1b8**	0x004	**0x2b8**	0x057	**0x3b8**	0x33a
0x0b9	0x0b6	**0x1b9**	0x23d	**0x2b9**	0x0ca	**0x3b9**	0x2a4
0x0ba	0x262	**0x1ba**	0x396	**0x2ba**	0x19f	**0x3ba**	0x3c5
0x0bb	0x0d1	**0x1bb**	0x102	**0x2bb**	0x0b3	**0x3bb**	0x2ce
0x0bc	0x058	**0x1bc**	0x3df	**0x2bc**	0x222	**0x3bc**	0x05b
0x0bd	0x382	**0x1bd**	0x1cd	**0x2bd**	0x0cb	**0x3bd**	0x0ba
0x0be	0x1fa	**0x1be**	0x276	**0x2be**	0x13c	**0x3be**	0x399
0x0bf	0x0dc	**0x1bf**	0x256	**0x2bf**	0x3a0	**0x3bf**	0x39b
0x0c0	0x323	**0x1c0**	0x08c	**0x2c0**	0x095	**0x3c0**	0x051
0x0c1	0x0a8	**0x1c1**	0x06a	**0x2c1**	0x339	**0x3c1**	0x0d8
0x0c2	0x154	**0x1c2**	0x21d	**0x2c2**	0x17b	**0x3c2**	0x313
0x0c3	0x0c2	**0x1c3**	0x150	**0x2c3**	0x272	**0x3c3**	0x09d
0x0c4	0x195	**0x1c4**	0x346	**0x2c4**	0x1be	**0x3c4**	0x310
0x0c5	0x1ed	**0x1c5**	0x255	**0x2c5**	0x10c	**0x3c5**	0x32c
0x0c6	0x00f	**0x1c6**	0x207	**0x2c6**	0x3ba	**0x3c6**	0x121
0x0c7	0x31c	**0x1c7**	0x3e6	**0x2c7**	0x2d1	**0x3c7**	0x283
0x0c8	0x3ae	**0x1c8**	0x26d	**0x2c8**	0x08f	**0x3c8**	0x275
0x0c9	0x252	**0x1c9**	0x041	**0x2c9**	0x182	**0x3c9**	0x0eb
0x0ca	0x3e2	**0x1ca**	0x3d0	**0x2ca**	0x217	**0x3ca**	0x24c
0x0cb	0x15d	**0x1cb**	0x3d3	**0x2cb**	0x1af	**0x3cb**	0x128
0x0cc	0x2f4	**0x1cc**	0x1ff	**0x2cc**	0x107	**0x3cc**	0x312
0x0cd	0x290	**0x1cd**	0x072	**0x2cd**	0x2ae	**0x3cd**	0x347
0x0ce	0x138	**0x1ce**	0x1c5	**0x2ce**	0x0f3	**0x3ce**	0x087
0x0cf	0x27b	**0x1cf**	0x395	**0x2cf**	0x31b	**0x3cf**	0x1f7
0x0d0	0x32a	**0x1d0**	0x33d	**0x2d0**	0x1ef	**0x3d0**	0x227
0x0d1	0x2e4	**0x1d1**	0x3e5	**0x2d1**	0x1a6	**0x3d1**	0x2f6
0x0d2	0x2c8	**0x1d2**	0x3fc	**0x2d2**	0x086	**0x3d2**	0x173
0x0d3	0x3bd	**0x1d3**	0x304	**0x2d3**	0x0f5	**0x3d3**	0x2d2
0x0d4	0x0b4	**0x1d4**	0x097	**0x2d4**	0x1cc	**0x3d4**	0x2de
0x0d5	0x06f	**0x1d5**	0x109	**0x2d5**	0x2bf	**0x3d5**	0x39a
0x0d6	0x3a3	**0x1d6**	0x155	**0x2d6**	0x022	**0x3d6**	0x3b0

Table A.1. Generated S-box input/output table.

0x0d7	0x253	**0x1d7**	0x046	**0x2d7**	0x18b	**0x3d7**	0x327
0x0d8	0x0bd	**0x1d8**	0x0dd	**0x2d8**	0x3f1	**0x3d8**	0x119
0x0d9	0x1c3	**0x1d9**	0x289	**0x2d9**	0x0fc	**0x3d9**	0x23c
0x0da	0x12b	**0x1da**	0x14c	**0x2da**	0x3a5	**0x3da**	0x0e9
0x0db	0x16a	**0x1db**	0x0cd	**0x2db**	0x2cc	**0x3db**	0x373
0x0dc	0x349	**0x1dc**	0x1ac	**0x2dc**	0x13b	**0x3dc**	0x063
0x0dd	0x14a	**0x1dd**	0x1ce	**0x2dd**	0x104	**0x3dd**	0x26a
0x0de	0x3f6	**0x1de**	0x261	**0x2de**	0x0f1	**0x3de**	0x1f6
0x0df	0x17c	**0x1df**	0x224	**0x2df**	0x3f7	**0x3df**	0x325
0x0e0	0x0a9	**0x1e0**	0x184	**0x2e0**	0x139	**0x3e0**	0x008
0x0e1	0x031	**0x1e1**	0x19b	**0x2e1**	0x124	**0x3e1**	0x23e
0x0e2	0x18e	**0x1e2**	0x12f	**0x2e2**	0x190	**0x3e2**	0x25e
0x0e3	0x2fb	**0x1e3**	0x305	**0x2e3**	0x2f2	**0x3e3**	0x287
0x0e4	0x3c3	**0x1e4**	0x2ff	**0x2e4**	0x07e	**0x3e4**	0x1de
0x0e5	0x021	**0x1e5**	0x05c	**0x2e5**	0x103	**0x3e5**	0x2e9
0x0e6	0x037	**0x1e6**	0x017	**0x2e6**	0x0f4	**0x3e6**	0x337
0x0e7	0x07b	**0x1e7**	0x24d	**0x2e7**	0x394	**0x3e7**	0x016
0x0e8	0x302	**0x1e8**	0x3ad	**0x2e8**	0x2d3	**0x3e8**	0x35e
0x0e9	0x28c	**0x1e9**	0x334	**0x2e9**	0x1ea	**0x3e9**	0x03d
0x0ea	0x377	**0x1ea**	0x22b	**0x2ea**	0x0cc	**0x3ea**	0x369
0x0eb	0x0e5	**0x1eb**	0x2ee	**0x2eb**	0x1d8	**0x3eb**	0x1ad
0x0ec	0x076	**0x1ec**	0x0f9	**0x2ec**	0x136	**0x3ec**	0x0fe
0x0ed	0x3fe	**0x1ed**	0x1f4	**0x2ed**	0x284	**0x3ed**	0x3ce
0x0ee	0x2ca	**0x1ee**	0x2df	**0x2ee**	0x27d	**0x3ee**	0x024
0x0ef	0x2dc	**0x1ef**	0x2ab	**0x2ef**	0x3ac	**0x3ef**	0x0c3
0x0f0	0x201	**0x1f0**	0x0c7	**0x2f0**	0x0d5	**0x3f0**	0x33c
0x0f1	0x257	**0x1f1**	0x239	**0x2f1**	0x062	**0x3f1**	0x165
0x0f2	0x143	**0x1f2**	0x209	**0x2f2**	0x0bb	**0x3f2**	0x206
0x0f3	0x370	**0x1f3**	0x248	**0x2f3**	0x01b	**0x3f3**	0x0a0
0x0f4	0x10d	**0x1f4**	0x19c	**0x2f4**	0x018	**0x3f4**	0x04e
0x0f5	0x0b8	**0x1f5**	0x105	**0x2f5**	0x335	**0x3f5**	0x074
0x0f6	0x221	**0x1f6**	0x371	**0x2f6**	0x27f	**0x3f6**	0x39c
0x0f7	0x3ff	**0x1f7**	0x0e3	**0x2f7**	0x25b	**0x3f7**	0x32b
0x0f8	0x032	**0x1f8**	0x20c	**0x2f8**	0x034	**0x3f8**	0x277
0x0f9	0x233	**0x1f9**	0x1fd	**0x2f9**	0x1e4	**0x3f9**	0x1a3
0x0fa	0x3d7	**0x1fa**	0x1fe	**0x2fa**	0x18c	**0x3fa**	0x11a
0x0fb	0x0ab	**0x1fb**	0x36d	**0x2fb**	0x3f2	**0x3fb**	0x31f
0x0fc	0x213	**0x1fc**	0x012	**0x2fc**	0x2f9	**0x3fc**	0x068
0x0fd	0x10e	**0x1fd**	0x013	**0x2fd**	0x07d	**0x3fd**	0x3ed
0x0fe	0x22a	**0x1fe**	0x38f	**0x2fe**	0x359	**0x3fe**	0x39e
0x0ff	0x17d	**0x1ff**	0x100	**0x2ff**	0x189	**0x3ff**	0x321

References

[1] A. Biryukov and C. de Canniere, "Linear Cryptanalysis for Block Ciphers," *Encyclopedia of Cryptography and Security,* 2nd ed. (H. C. A. van Tilborg and S. Jajodia, eds.), Boston: Springer, 2011 pp. 722–725. doi:10.1007/978-1-4419-5906-5_589.

[2] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, New York: Springer-Verlag, 1993. doi:10.1007/978-1-4613-9314-6.

[3] D. Wagner, "The Boomerang Attack," in *Fast Software Encryption: 6th International Workshop (FSE'99)*, Rome (L. Knudsen, ed.), Berlin, Heidelberg: Springer, 2001 pp. 156–170. doi:10.1007/3-540-48519-8_12.

[4] Y. Li and M. Wang, "Constructing S-Boxes for Lightweight Cryptography with Feistel Structure," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2014)*, Busan, South Korea (L. Batina and M. Robshaw, eds.), Berlin, Heidelberg: Springer, 2014 pp. 127–146. doi:10.1007/978-3-662-44709-3_8.

[5] A. Canteaut, S. Duval and G. Leurent, "Construction of Lightweight S-Boxes Using Feistel and MISTY Structures," in *Selected Areas in Cryptography (SAC 2015), 22nd International Conference*, Sackville, NB, Canada (O. Dunkelman and L. Keliher, eds.), Cham: Springer, 2016, pp. 373–393. doi:10.1007/978-3-319-31301-6_ 22.

[6] L. Burnett, G. Carter, E. Dawson and W. Millan, "Efficient Methods for Generating MARS-Like S-Boxes," in *Fast Software Encryption, 7th International Workshop (FSE 2000)*, New York (G. Goos, J. Hartmanis, J. van Leeuwen and B. Schneier, eds.), Berlin, Heidelberg: Springer, 2001, pp. 300–313. doi:10.1007/3-540-44706-7_21.

[7] 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 2017)*, Siena, Italy, New York: Association for Computing Machinery, 2017 pp. 409–414. doi:10.1145/3075564.3079069.

[8] 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 2017)*, Berlin, Germany, New York: Association for Computing Machinery, 2017 pp. 251–252. doi:10.1145/3067695.3076084.

[9] S. Marochok and P. Zajac, "Algorithm for Generating S-Boxes with Prescribed Differential Properties," *Algorithms*, **16**(3), 2023 157. doi:10.3390/a16030157.

[10] M. M. Dimitrov, "On the Design of Chaos-Based S-Boxes," *IEEE Access*, **8**, 2020 pp. 117173–117181. doi:10.1109/ACCESS.2020.3004526.

[11] K. Nyberg, "Differentially Uniform Mappings for Cryptography," in *Advances in Cryptology—EUROCRYPT '93*, Lofthus, Norway (T. Helleseth, ed.), Berlin, Heidelberg: Springer, 1994 pp. 55–64. doi:10.1007/3-540-48285-7_6.

[12] J. Daemen and V. Rijmen, "The Block Cipher Rijndael," in *Smart Card Research and Advanced Applications (CARDIS 1998)*, Louvain-la-Neuve, Belgium (J. J. Quisquater and B. Schneier, eds.), Berlin, Heidelberg: Springer, 2000 pp. 277–284. doi:10.1007/10721064_26.

[13] A. H. Zahid and M. J. Arshad, "An Innovative Design of Substitution-Boxes Using Cubic Polynomial Mapping," *Symmetry*, **11**(3), 2019 437. doi:10.3390/sym11030437.

[14] A. H. Zahid, H. Rashid, M. M. U. Shaban, S. Ahmad, E. Ahmed, M. T. Amjad, M. A. T. Baig et al., "Dynamic S-Box Design Using a Novel Square Polynomial Transformation and Permutation," *IEEE Access*, **9**, 2021 pp. 82390–82401. doi:10.1109/ACCESS.2021.3086717.

[15] J. Daemen and V. Rijmen, *The Design of Rijndael: AES–the Advanced Encryption Standard*, New York: Springer-Verlag, 2002. doi:10.1007/978-3-662-60769-5.

[16] T. Haider, N. A. Azam and U. Hayat, "Substitution Box Generator with Enhanced Cryptographic Properties and Minimal Computation Time," *Expert Systems with Applications*, **241**, 2024 122779. doi:10.1016/j.eswa.2023.122779.

[17] A. Malal and C. Tezcan, "FPGA-Friendly Compact and Efficient AES-Like 8x8 S-Box," *Microprocessors and Microsystems*, **105**, 2024 105007. doi:10.1016/j.micpro.2024.105007.

[18] 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 *Cryptographic Hardware and Embedded Systems (CHES 2007)*, Vienna, Austria (P. Paillier and I. Verbauwhede, eds.), Berlin, Heidelberg: Springer, 2007 pp. 450–466. doi:10.1007/978-3-540-74735-2_31.

[19] C. Dobraunig, M. Eichlseder, F. Mendel and M. Schläffer, "Ascon v1.2: Lightweight Authenticated Encryption and Hashing," *Journal of Cryptology*, **34**(3), 2021 33. doi:10.1007/s00145-021-09398-9.

[20] M. Naseer, S. Tariq, N. Riaz, N. Ahmed, S. Fahd, M. Hussain and S. A. Khan, "A Quantitative Security Analysis of S-Boxes in the NIST Lightweight Cryptography Finalists." arxiv.org/abs/2404.06094.

[21] L. Zhang, C. Ma, Y. Zhao and W. Zhao, "A Novel Dynamic S-Box Generation Scheme Based on Quantum Random Walks Controlled by a Hyper-chaotic Map," *Mathematics*, **12**(1), 2023 84. doi:10.3390/math12010084.

[22] S. Wolfram, "Statistical Mechanics of Cellular Automata," *Reviews of Modern Physics*, **55**(3), 1983 pp. 601–644. doi:10.1103/RevModPhys.55.601.

[23] B. Martin, "A Walsh Exploration of Elementary CA Rules," *Journal of Cellular Automata*, **3**(2), 2008 pp. 145–156. www.oldcitypublishing.com/journals/jca-home/jca-issue-contents/jca-volume-3-number-2-2008/jca-3-2-p-145–156.

[24] E. K. Alekseev and E. K. Karelina, "Classification of Correlation-Immune and Minimal Correlation-Immune Boolean Functions of 4 and 5 Variables," *Discrete Mathematics and Applications*, **25**(4), 2015 pp. 193–202. doi:10.1515/dma-2015-0019.

[25] P. Langevin and G. Leander, "Counting All Bent Functions in Dimension Eight 99270589265934370305785861242880," *Designs, Codes and Cryptography*, **59**(1), 2011 pp. 193–205. doi:10.1007/s10623-010-9455-z.

[26] J. Daemen, R. Govaerts and J. Vandewalle, "A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function Based on a Cellular Automaton," in *Advances in Cryptology (ASIACRYPT '91)*, Fujiyosida, Japan (H. Imai, R. L. Rivest and T. Matsumoto, eds.), Berlin, Heidelberg: Springer, 1993 pp. 82–96. doi:10.1007/3-540-57332-1_7.

[27] H. Gutowitz, "Cryptography with Dynamical Systems," *Cellular Automata and Cooperative Systems* (N. Boccara, E. Goles, S. Martinez and P. Picco, eds.), Dordrecht: Springer, 1993 pp. 237–274. doi:10.1007/978-94-011-1691-6_21.

[28] L. Mariot, S. Picek, A. Leporati and D. Jakobovic, "Cellular Automata Based S-Boxes," *Cryptography and Communications*, **11**(1), 2019 pp. 41–62. doi:10.1007/s12095-018-0311-8.

[29] A. John and J. Jose, "Hash Function Design Based on Hybrid Five-Neighborhood Cellular Automata and Sponge Functions," *Complex Systems*, **32**(2), 2023 pp. 171–188. doi:10.25088/ComplexSystems.32.2.171.

[30] E. Formenti, K. Imai, B. Martin and J. B. Yunès, "Advances on Random Sequence Generation by Uniform Cellular Automata," *Computing with New Resources: Essays Dedicated to Jozef Gruska on the Occasion of his 80th Birthday* (C. S. Calude, R. Freivalds and I. Kazuo, eds.), Cham: Springer, 2014 pp. 56–70. doi:10.1007/978-3-319-13350-8_5.

[31] T. Tuncer and E. Avaroglu, "Random Number Generation with LFSR Based Stream Cipher Algorithms," in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia (P. Biljanovic, M. Koricic, K. Skala, T. G. Grbac, M. Cicin-Sain, V. Sruk, R. Slobodan Ribaric et al., eds.), Rijeka, Croatia: Croatian Society for Information and Communication Technology, Electronics and Microelectronics, 2017 pp. 171–175. doi:10.23919/MIPRO.2017.7973412.

[32] G. Bertoni, J. Daemen, M. Peeters and G. van Assche, "Keccak," in *Advances in Cryptology (EUROCRYPT 2013)*, Athens, Greece (J. Johansson and P. Q. Nguyen, eds.), Berlin, Heidelberg: Springer, 2013 pp. 313–314. doi:10.1007/978-3-642-38348-9_19.

[33] C. Carlet and S. Mesnager. "On the Supports of the Walsh Transforms of Boolean Functions." Cryptology ePrint Archive. (Mar 12, 2026) eprint.iacr.org/2004/256.

[34] G.-Z. Xiao and J. L. Massey, "A Spectral Characterization of Correlation-Immune Combining Functions," *IEEE Transactions on Information Theory*, **34**(3), 1988 pp. 569–571. doi:10.1109/18.6037.

[35] H. Feistel, "Cryptography and Computer Privacy," *Scientific American*, **228**(5), 1973 p. 15. doi:10.1038/scientificamerican0573-15.

[36] M. Luby and C. Rackoff, "How to Construct Pseudo-random Permutations from Pseudo-random Functions," in *Advances in Cryptology (CRYPTO 1985)*, Santa Barbara, CA (H. C. WIlliams, ed.), Berlin, Heidelberg: Springer, 1985 p. 447. doi:10.1007/3-540-39799-X_34.

[37] J. Patarin, "Luby–Rackoff: 7 Rounds Are Enough for $2^{n(1-\epsilon)}$ Security," in *Advances in Cryptology (CRYPTO 2003)*, Santa Barbara, CA (D. Boneh, ed.), Berlin, Heidelberg: Springer, 2003 pp. 513–529. doi:10.1007/978-3-540-45146-4_30.

[38] M. Naor and O. Reingold, "On the Construction of Pseudorandom Permutations: Luby–Rackoff Revisited," *Journal of Cryptology*, **12**(1), 1999 pp. 29–66. doi:10.1007/PL00003817.

[39] L. Poinsot, "Boolean Bent Functions in Impossible Cases: Odd and Plane Dimensions," *International Journal of Computer Science and Network Security*, **6**(8A), 2006 pp. 18–26. hal-00460344.

[40] B. Preneel, "Analysis and Design of Cryptographic Hash Functions," Ph.D. thesis, Computer Science Department, KU Leuven, Leuven, Belgium, 2003. homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf.

[41] "Security Requirements for Cryptographic Modules," Gaithersburg, MD: National Institute of Standards and Technology, 1994. doi:10.6028/NIST.FIPS.140-2.

[42] C. Adams and S. Tavares, "The Structured Design of Cryptographically Good S-Boxes," *Journal of Cryptology*, **3**(1), 1990, pp. 27–41. doi:10.1007/BF00203967.

[43] A. Waheed, F. Subhan, M. M. Suud, M. Alam and S. Ahmad, "An Analytical Review of Current S-Box Design Methodologies, Performance Evaluation Criteria, and Major Challenges," *Multimedia Tools and Applications*, **82**(19), 2023 pp. 29689–29712. doi:10.1007/s11042-023-14910-3.

[44] W. Millan, "Low Order Approximation of Cipher Functions," in *Cryptography: Policy and Algorithms (CPA 1995)*, Brisbane, Queensland, Australia (E. Dawson and J. Golić, eds.), Berlin, Heidelberg: Springer, 1996 pp. 144–155. doi:10.1007/BFb0032354.

[45] T. Jakobsen and L. R. Knudsen, "Attacks on Block Ciphers of Low Algebraic Degree," *Journal of Cryptology*, **14**(3), 2001 pp. 197–210. doi:10.1007/s00145-001-0003-x.

[46] C. Carlet and C. Ding, "Nonlinearities of S-Boxes," *Finite Fields and Their Applications*, **13**(1), 2007 pp. 121–135. doi:10.1016/j.ffa.2005.07.003.

[47] A. F. Webster and S. E. Tavares, "On the Design of S-Boxes," in *Advances in Cryptology (CRYPTO 1985)*, (H. C. Williams, ed.), Berlin, Heidelberg: Springer, 1986 pp. 523–534. doi:10.1007/3-540-39799-X_41.

[48] J. Daemen and V. Rijmen, "Probability Distributions of Correlation and Differentials in Block Ciphers," *Journal of Mathematical Cryptology*, **1**(3), 2007 pp. 221–242. doi:10.1515/JMC.2007.011.

[49] B. Arshad, N. Siddiqui, Z. Hussain and M. Ehatisham-Ul-Haq, "A Novel Scheme for Designing Secure Substitution Boxes (S-Boxes) Based on Moebius Group and Finite Field," *Wireless Personal Communications*, **124**(4), 2022 pp. 3527–3548. doi:10.1007/s11277-022-09524-1.

[50] S. Hussain, S. S. Jamal, T. Shah and I. Hussain, "A Power Associative Loop Structure for the Construction of Non-linear Components of Block Cipher," *IEEE Access*, **8**, 2020 pp. 123492–123506. doi:10.1109/ACCESS.2020.3005087.

[51] K. Nyberg and L. R. Knudsen, "Provable Security against Differential Cryptanalysis," in *Advances in Cryptology (CRYPTO 1992)*, Santa Barbara, CA (E. F. Brickell, ed.), Berlin, Heidelberg: Springer, 1992 pp. 566–574. doi:10.1007/3-540-48071-4_41.

[52] C. Carlet, "On the APN-ness and Differential Uniformity of Some Classes of (n, n)-functions over $\mathbb{F}_2^n$," *Advances in Mathematics of Communications*, **18**(2), 2024 pp. 283–303. doi:10.3934/amc.2023027.

[53] Y. Jeon, S. Baek, H. Kim, G. Kim and J. Kim, "Differential Uniformity and Linearity of S-Boxes by Multiplicative Complexity," *Cryptography and Communications*, **14**(4), 2022 pp. 849–874. doi:10.1007/s12095-021-00547-2.

[54] C. Cid, T. Huang, T. Peyrin, Y. Sasaki and L. Song, "Boomerang Connectivity Table: A New Cryptanalysis Tool," in *Advances in Cryptology (EUROCRYPT 2018)*, Tel Aviv, Israel (J. B. Nielsen and V. Rijmen, eds.), Berlin, Heidelberg: Springer International Publishing, 2018 pp. 683–714. doi:10.1007/978-3-319-78375-8_22.

[55] S. el Hirch, S. Mella, A. Mehrdad and J. Daemen, "Improved Differential and Linear Trail Bounds for ASCON," *IACR Transactions on Symmetric Cryptology*, **2022**(4), 2022 pp. 145–178. doi:10.46586/tosc.v2022.i4.145-178.