CryptoBench: Benchmarking Evolutionary Algorithms with Cryptographic Problems

Stjepan Picek
MIT, CSAIL
32 Vassar St
Cambridge, MA, USA
stjepan@computer.org

Domagoj Jakobovic
University of Zagreb, Faculty of El. Engineering and Computing
Unska 3, Zagreb, Croatia
domagoj.jakobovic@fer.hr

Una-May O'Reilly
MIT, CSAIL
32 Vassar St
Cambridge, MA, USA
unamay@csail.mit.edu

ABSTRACT

We propose a new set of benchmark problems that come from the field of cryptography. These problems are often easy to define, they are relevant in practice, and some of them have a known optimal value. The solutions from these problems can be used as cryptographic primitives and consequently function as components of cryptographic algorithms. They can be compared not only with other evolutionary computation techniques but with a broad spectrum of techniques. Finally, due to the fact that many of the optimal solutions are not known (although we know they must exist), finding such solutions would also have a significant practical impact.

KEYWORDS

Evolutionary computation, Benchmark, Boolean functions, Vectorial Boolean functions, Physically Unclonable Functions

ACM Reference format:
Stjepan Picek, Domagoj Jakobovic, and Una-May O'Reilly. 2017. CryptoBench: Benchmarking Evolutionary Algorithms with Cryptographic Problems. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082535

1 INTRODUCTION

We propose a benchmark suite based on several well-known problems from the domain of cryptography. That approach offers several advantages such as:

(1) Problems that are well-understood, yet difficult.
(2) Problems that have practical significance and where new results can also attract attention from the cryptographic community.
(3) Problems that are well investigated and where one often has a significant amount of knowledge attainable bounds and trade-offs.
(4) Problems where it is possible to compare the solutions not only with EC techniques but also with solutions obtained by deterministic techniques or other domain specific heuristics.

We present three classes of cryptographic problems that offer tunability with respect to problem size, number of relevant cryptographic properties, and possible representations of solutions including discrete and continuous fixed-length vectors as well as genetic programming expressions. The problems considered here are based on:

(1) finding Boolean functions with a certain set of properties – Section 4,
(2) finding vectorial Boolean functions with a certain set of properties – Section 5,
(3) finding the delay vectors describing Physically Unclonable Functions (PUFs) – Section 6.

Our presentation takes a solution representation perspective. All problems herein can be fundamentally represented by a bitstring, however there are more practical mappings to conventional evolutionary computation (EC) representations. The problems presented here are already well explored and have plethora of results for comparison. One can easily find a number of sources and publicly available tools that offer the needed functionalities for testing cryptographic primitives [17, 31, 39]. Finally, we refer interested readers for further details on applications of evolutionary computation to cryptographic problems to [29, 30].

The rest of this paper is organized as follows. In Section 2 we give details about our framework. Section 3 introduces the notation and gives a general introduction to cryptographic problems we consider here. Sections 4, 5, and 6 present the three classes of problems: Boolean functions, vectorial Boolean functions, and PUFs, respectively. Finally, Section 7 gives a brief conclusion.

2 CRYPTOBENCH FRAMEWORK

The code for all considered problems is offered as a part of the Evolutionary Computation Framework (ECF) [15] written in C++ and also as a standalone set of functions [16]. There, we give not only the implementations for properties presented here, but also predefined fitness functions that cover all the specific problem instances given here (as well as other problem instances). For each class of the problems presented here, there is a separate project to avoid confusion around which properties align with a class. In [16] we also give, for each problem, state-of-the-art solutions for a number of different sizes and techniques (both heuristic and deterministic). Finally, it is possible to submit new solutions to enter a "Hall of Fame" as well as to provide the reference to work.

To facilitate a better understanding of the problems, but also to potentially improve the system performance, our framework (see Figure 1) offers several properties and representations of cryptographic primitives written in different ways. For example, the balancedness property of a Boolean function can be determined from truth table and the Walsh-Hadamard transformation. We provide the code for all such options for flexibility. We also provide the source code for both naive and optimized implementations for various properties and representations of cryptographic primitives.

In Figure 1 we depict our framework for Boolean functions. We show several possible representations of a Boolean function and how it is translated into truth table form. From there, the property checker can be regarded as a black box where for each input it outputs the property value. If desired, this black box can be easily examined in order to learn additional information about a specific problem.

3 BACKGROUND AND NOTATION

One standard division of cryptography is into symmetric key cryptography and public key cryptography [11, 28]. Going one step further, symmetric key cryptography can be divided into block ciphers and stream ciphers. Two out of three problems we propose belong to symmetric key cryptography where one has more importance in stream ciphers and the other in block ciphers.

A common trait of all such ciphers is that they are designed to fulfill a specific number of cryptographic criteria. These varying criteria enable ciphers to resist different cryptanalysis attacks like differential [3], linear [22], and algebraic [8].

For instance, to make a block cipher resilient against linear cryptanalysis, one option is to use vectorial Boolean functions with as high as possible nonlinearity property value (we will explain the meaning of this property and how its value is derived in Section 5). In stream ciphers, one usual source of nonlinearity are Boolean functions [5]. Both of these scenarios, while not unique, show the importance of Boolean functions in cryptography. Finding Boolean functions with good properties and analyzing the best possible trade-offs among these properties are still crucial questions today. Such functions could then be used in new, better designs of ciphers. For a detailed discussion on Boolean functions and vectorial Boolean functions, we refer readers to [5, 6].

Physically Unclonable Functions (PUFs) are embedded or standalone devices used as a means to generate either a source of randomness or to obtain an instance-specific uniqueness for secure identification. This is achieved by relying on inherent uncontrollable manufacturing process variations, which results in each chip having a unique response. Optimization techniques can be used to find a model ("clone") of a PUF by modeling the delay vector of an actual PUF in as few measurements as possible. For more details on PUFs, we refer interested readers to [1, 20].

3.1 Notation

Let n, m be positive integers, i.e., $n, m \in \mathbb{N}^+$. We denote by $\mathbb{F}_2^n$ the n-dimensional vector over $\mathbb{F}_2$ and by $\mathbb{F}_{2^n}$ the finite field with 2^n elements. The set of all n-tuples of elements in the field $\mathbb{F}_2$ is denoted by $\mathbb{F}_2^n$, where $\mathbb{F}_2$ is the Galois field with two elements. For any set S, we denote $S\backslash\{0\}$ by S^*. The usual inner product of a and b equals $a \cdot b = \bigoplus_{i=1}^n a_i b_i$ in F_2^n. The Hamming weight (HW) of a vector a, where $a \in \mathbb{F}_2^n$, is the number of non-zero positions in the vector.

Table 1: The search space size for various input size n.

n	4	6	8	10	12	14	16
#	2^{16}	2^{64}	2^{256}	$2^{1\,024}$	$2^{4\,096}$	$2^{16\,384}$	$2^{65\,536}$

4 PROBLEMS BASED ON BOOLEAN FUNCTIONS

The role of Boolean functions is prominent in several areas besides the cryptography, like sequences and coding theory. Therefore, various methods for the construction of Boolean functions with desired properties are of the direct interest. To be clear, we propose Boolean function problems that will use EC to find Boolean functions with specific properties while it is also necessary for explanatory purposes to describe in some detail these properties. For further details on Boolean functions, we refer interested readers to [5].

A Boolean function is any mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. For a Boolean function with n inputs, there are in total 2^{2^n} possible Boolean functions. We list several search space sizes for a Boolean function with n inputs in Table 1 where it can be seen that the number of functions grows exponentially and there is a variety of search space sizes one can consider.

In order to be able to calculate different cryptographic properties of a Boolean function, we need to be able to switch among various representations of Boolean functions. We discuss three such representations in the next section.

4.1 Boolean Function Representations

A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a truth table (TT), which is a vector $(f(0), ..., f(1))$ that contains the function values of f, ordered lexicographically, i.e., $a \leq b$.

The Walsh-Hadamard transform W_f is a second unique representation of a Boolean function that measures the correlation between $f(x)$ and the linear functions $a \cdot x$ [5]:

$$W_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus a \cdot x}. \tag{1}$$

Finally, one can uniquely represent a Boolean function f on $\mathbb{F}_2^n$ by means of a polynomial in $\mathbb{F}_2\left[x_0, ..., x_{n-1}\right] / (x_0^2 - x_0, ..., x_{n-1}^2 - x_{n-1})$, i.e., by the Algebraic Normal Form (ANF). ANF is a multivariate polynomial defined as [24]:

$$f(x) = \bigoplus_{a \in \mathbb{F}_2^n} h(a) \cdot x^a, \tag{2}$$

where $h(a)$ is defined by the Möbius inversion principle

$$h(a) = \bigoplus_{x \leq a} f(x), \text{ for any } a \in \mathbb{F}_2^n. \tag{3}$$

4.2 Solution Representation

The problem of finding a Boolean function with the desired cryptographic properties is amenable to different solution representations.

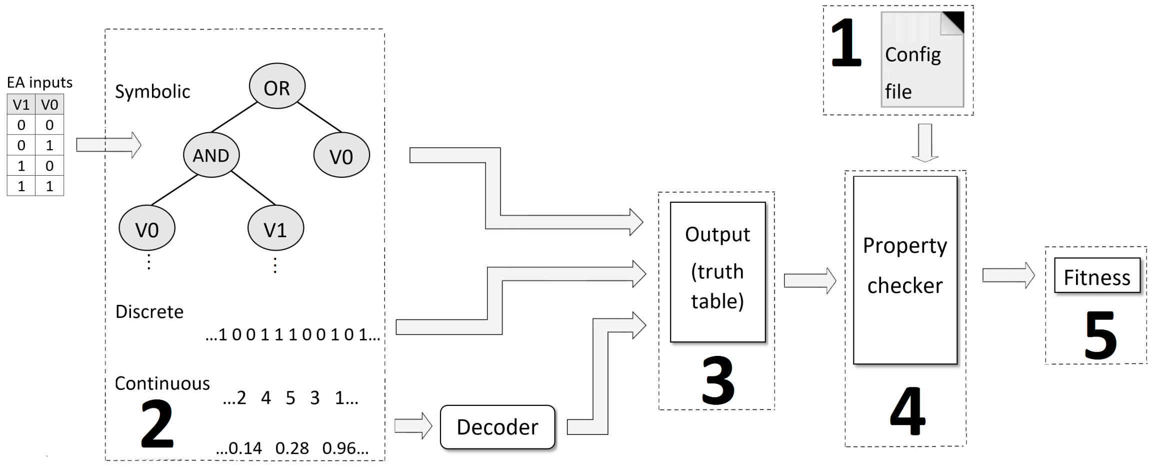


Figure 1: Workflow of CryptoBench framework. 1) Configure file with function size and cryptographic properties 2) Run evolutionary algorithm 3) Obtain truth table representation of a solution 4) Run Property checker 5) Use the checker's output as a metric of merit with values of desired properties. Note: this workflow is valid for Boolean functions, vectorial Boolean functions, and PUFs.

This is important since many properties can be directly calculated only from certain representations.

The selection of the most appropriate representation of a solution depends on different objectives. For instance, one option would be to use a solution representation that requires the least number of changes of Boolean function representation in order to calculate the necessary properties. A second option could be to use a solution representation that is easier to understand or shorter – for instance, truth table encoded with a bitstring representation is easy to understand since each bit represents an output value for a certain input values. Executable expression encoding could be much more difficult to understand but could potentially compactly encode a Boolean function. What is common for all representations is that they are unconstrained and it is of no importance for the solution from the cryptographic perspective what encoding has been used. Any unique encoding of a solution can be used to transform the solution into any other encoding. Since specific cryptographic properties pose challenges that differ from one representation to another, we expect the best results for problems of different properties to also differ in which representation has been used.

Bitstring Representation. An obvious way of representing a Boolean function is with its truth table, encoded as a sequence of bits. Since the length of the truth table is 2^n, where n is the number of Boolean variables, this representation quickly becomes very large and inefficient. Experiments with bitstring representation have been conducted for sizes of up to 18 variables [14], but this encoding is commonly applicable only for small scale problems, see [23, 32, 34].

Integer and Floating-point Representation. In the integer case, the solution is encoded with an array of integer values, where each value replaces a certain number of bits from the truth table. For instance, a truth table of size 256 can also be represented with 32 integer variables in the range $[0, 255]$; this way, each integer value is decoded into 8 bits that are concatenated in a complete truth table.

The same approach can be simulated with floating-point values in the range of $[0, 1]$, where each floating-point number is first decoded into a corresponding integer value using a linear transformation, and then copied in the truth table. This allows the use of optimization algorithms that search the continuous domain. These representations have been used for instance in [21, 36].

Executable expression representation. Another representation is based on Genetic programming (GP), where a Boolean function is represented through a combination of Boolean primitives such as AND, XOR, NOT, etc. and input variables, and the expression is evaluated to generate the Boolean truth table for each input combination. This representation is probably the most versatile one, and has usually been shown to exhibit much better performance than integer or floating-point [32, 34]. There are many GP variants, such as Gene expression programming (GEP) or Cartesian genetic programming (CGP), that can also be used to evolve vectorial Boolean functions (see Section 5). Examples of evolution of Boolean functions with the executable expressions representation are [14, 35].

4.3 Properties and Bounds

In this section, we list a number of properties of Boolean functions. We also note conflicting properties, which means a trade-off needs to be chosen.

- A Boolean function f is **balanced** if it takes the value 1 exactly the same number 2^{n-1} of times as value 0 when the input ranges over $\mathbb{F}_2^n$.
- The minimum Hamming distance between a Boolean function f and all affine functions (in the same number of variables as f) is called the **nonlinearity** of f. The nonlinearity N_f of a Boolean function f can be expressed in

terms of the Walsh-Hadamard coefficients as [5]:

$$N_f = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} |W_f(a)|. \quad (4)$$

The Parseval's relation equals:

$$\sum_{a \in \mathbb{F}_2^n} W_f(a)^2 = 2^{2n}, \quad (5)$$

and it implies that the mean of $W_f(a)^2$ equals 2^n, and $\max_{a \in \mathbb{F}_2^n} |W_f(a)|$ is then at least equal to the square root of this mean. From Eq. (5), it follows that the maximal value of the Walsh-Hadamard spectrum equals at least $2^{\frac{n}{2}}$, which occurs with equality in the case of **bent** Boolean functions. From this equation we see that the nonlinearity of any Boolean function is less or equal to:

$$N_f \leq 2^{n-1} - 2^{\frac{n}{2}-1}. \quad (6)$$

- One related notion is of **plateaued** functions where the Walsh-Hadamard spectrum takes one nonzero absolute values and a value 0. If there is only one nonzero value, we call such functions near-bent functions and if there are two such values (i.e., both positive and a negative of some value) then we call such functions semi-bent functions.
- A Boolean function f is **correlation immune** of order t (in brief, $CI(t)$) if the output of the function is statistically independent of the combination of any t of its inputs. For the Walsh-Hadamard spectrum it holds equivalently [13]:

$$W_f(a) = 0, \text{ for } 1 \leq HW(a) \leq t. \quad (7)$$

- A Boolean function f is t-**resilient** if it is balanced and with correlation immunity of order t [38].
- The **algebraic degree** deg of a Boolean function f is defined as the number of variables in the largest product term of the function's ANF having a non-zero coefficient [19]:

$$deg = \max(HW(a) : h(a) = 1). \quad (8)$$

Here, $h(a)$ is defined by the Möbius inversion principle.
- The **algebraic immunity** (AI) of a Boolean function f is the lowest degree of a nonzero function g from $\mathbb{F}_2^n$ into $\mathbb{F}_2$ for which $fg = 0$ or $(f \oplus 1)g = 0$ where f and g are Boolean functions. Here, fg is the Hadamard product of f and g, whose support is the intersection of the supports of f and g. A function g such that $fg = 0$ is called an annihilator of f [24].

For a resilient Boolean function where $t > 1$ and $t \neq n - 1$, the following Siegenthaler bound holds [38]. Therefore, the correlation immunity and the algebraic degree properties are conflicting and it is not possible to obtain a Boolean function with both properties optimal.

$$t \leq n - deg - 1. \quad (9)$$

4.4 Problems

The variety of solution representations ans properties allows us to define a number of problems ranging from single-objective to many-objective. Examples of problems valid for any Boolean function size are:

(1) Finding bent functions (single-objective optimization). This could be stated as a problem of **maximizing** the nonlinearity property since bent functions have maximal possible nonlinearity:

$$objective = N_f. \quad (10)$$

(2) Finding balanced, highly nonlinear functions (single objective optimization with a constraint).
(3) Finding Boolean functions with a minimum HW and different values for correlation immunity (multi-objective optimization). This problem can be defined as:

$$objective_A = |CI - TARGET\_CI|; \quad (11)$$

$$objective_B = MAX\_HW - HW, \quad (12)$$

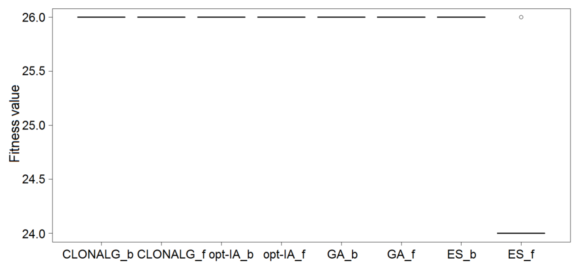
where the first criterion, $objective_A$, is **minimized**, while the second criterion, $objective_B$, is **maximized**. CI represents the current value of correlation immunity and $TARGET\_CI$ represents the desired value for correlation immunity. MAX_HW is the maximal Hamming weight for a Boolean function of n inputs and HW is the current value of the Hamming weight.
(4) Finding balanced Boolean functions with a high nonlinearity and a high algebraic degree (multi-objective optimization).
(5) Finding balanced Boolean functions with a high nonlinearity, large algebraic degree, large algebraic immunity, large fast algebraic immunity, and large correlation immunity (many-objective optimization).
(6) From Eq. (5) we see that the bent Boolean functions have the Walsh-Hadamard spectrum equal to $\left|2^{\frac{n}{2}}\right|$ and therefore we can pose the problem as a combinatorial one of finding the correct ordering of $\pm 2^{\frac{n}{2}}$ values. This combinatorial problem can be also extended for the near-bent and semi-bent functions.

As an example, Figure 2 contrasts solutions for a number of algorithms and encodings when looking for balanced, highly nonlinear functions. It compares solutions based on genetic algorithm (GA), evolution strategy (ES), opt-IA, and CLONALG algorithms. Each of the algorithms uses binary and floating-point representation as denoted with _b and _f, respectively. When investigating Boolean functions with 6 variables, seven out of eight algorithms reach maximal value in 100% of cases (see Eq. (6)) while for the case with 16 variables the behavior of the investigated algorithms differs significantly.

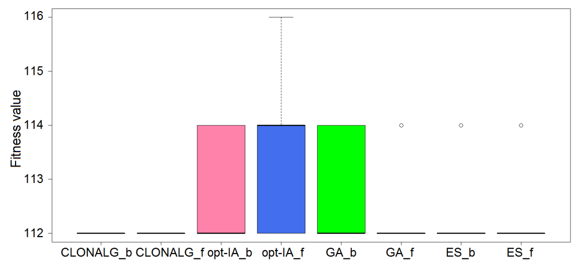
5 PROBLEMS BASED ON VECTORIAL BOOLEAN FUNCTIONS

Similarly as the Boolean functions represent a source of nonlinearity for stream ciphers, vectorial Boolean functions (functions with n inputs and m outputs, i.e., (n, m)-functions) or S-boxes are a source of nonlinearity for many block ciphers. We list some occurring S-box sizes and cryptographic algorithms using them: 3×3 (3Way [9]), 4 × 4 (PRESENT [4]), 5 × 5 (Keccak [2]), 8 × 8 (AES [10]), and 6 × 4 (DES [12]).

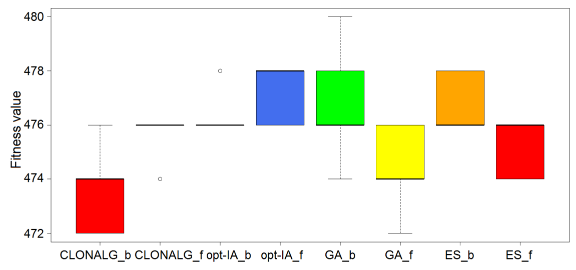
An (n, m)-function is any mapping F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Such a F can be defined as a vector $F = (f_1, \cdots, f_m)$, where the Boolean



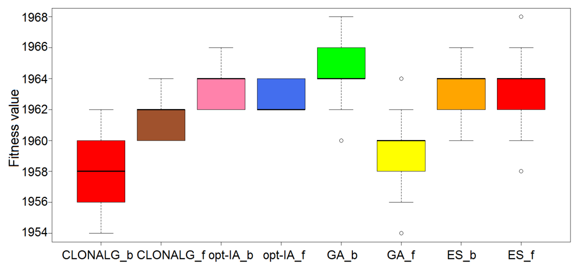
(a) Boolean functions with 6 inputs.



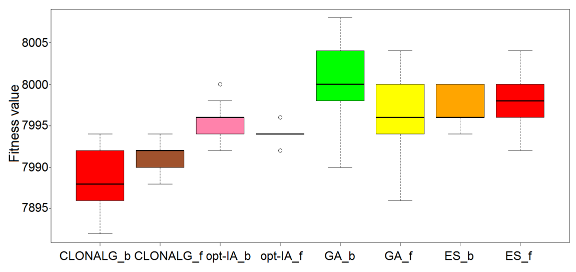
(b) Boolean functions with 8 inputs.



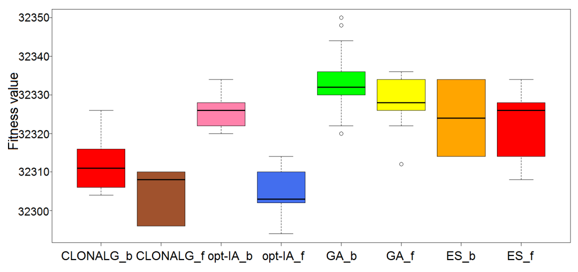
(c) Boolean functions with 10 inputs.



(d) Boolean functions with 12 inputs.



(e) Boolean functions with 14 inputs.



(f) Boolean functions with 16 inputs.

Figure 2: Results for balanced, highly nonlinear Boolean functions of various sizes, second problem in Section 4.4

Table 2: Search space size for (n, n)-functions.

n	4	5	6	7	8
#	2^{64}	2^{160}	2^{384}	2^{896}	$2^{2\,048}$

functions $f_i : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ for $i \in \{1, \cdots, m\}$ are called the coordinate functions of F. The component functions of an (n, m)-function F are all the linear combinations of the coordinate functions with non all-zero coefficients.

For S-boxes the search space size is significantly larger than that of Boolean functions since we need to consider all linear combinations of the coordinate functions when calculating a certain property. In general, the number of possible solutions for an (n, m)-function equals $2^{m \cdot 2^n}$. In Table 2 we give several search space sizes for an (n, n)-function.

5.1 S-box Representations

An S-box can be represented in the truth table form as a matrix of dimension $2^n \times m$ where each column m represents one Boolean function (i.e., one coordinate function).

The Walsh-Hadamard transform of an S-box equals [6]:

$$W_F(a, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) \oplus a \cdot x}, \tag{13}$$

where $a \in \mathbb{F}_2^n$ and $v \in \mathbb{F}_2^{m*}$.

An (n, m)-function F can be represented as a list of values (lookup table - LUT), with each value ranging from 0 to $2^m - 1$. Alternatively said, an (n, m)-function can be implemented as a lookup table with 2^n words of m bits each. When $n = m$ it is usual that the S-box is bijective, i.e., that each value in the output appears exactly once.

5.2 Solution Representation

All representations listed in Section 4.2 for Boolean functions are also valid for S-boxes and have particular importance when an

S-box is not bijective, i.e., when $n \neq m$. Such examples can be found in S-boxes where the output dimension is strictly smaller than the input dimension [6]. If an S-box is bijective, then the most natural representation for it is the permutation encoding as explained in the next section.

Permutation representation. In the permutation representation, $n \times n$ S-box is defined with an array of 2^n integer numbers with values between 0 and $2^n - 1$ (2^n distinct values). Each of those values occurs exactly once in an array and represents one entry for the S-box lookup table, where inputs are in lexicographical order. This type of encoding enforces balancedness (see Section 5.3) and significantly reduces the search space with respect to a general case.

Bitstring representation. Besides using bitstring representation directly, i.e., to represent the values of an S-box, in certain cases it is also possible to improve the properties of an S-box by utilizing various linear/affine transformations. By using an affine transformation, it is possible to change the values for certain properties (that are affine variant), while other properties stay unchanged (affine invariant properties). Two S-boxes S_1 and S_2 of dimension $n \times n$ are affine equivalent if the following equation holds [18]:

$$S_1(x) = B(S_2(A(x) \oplus a)) \oplus b, \tag{14}$$

where A and B are invertible $n \times n$ matrices in $GF(2)$ and $a, b \in \mathbb{F}_2^n$.

To evolve an improved S-box with regards to the affine variant properties, we need to find appropriate affine transformation [37]. Accordingly, we evolve affine transformations where each individual consists of four units (in a general case). The first two units represent the matrices A and B while the last two units represent the constants a and b. Since the values in matrices are bits, the usage of a bitstring representation is a natural choice.

5.3 Properties and Bounds

Some (but not all) important and common properties of S-boxes are:

- An (n, m)-function F is called **balanced** if it takes every value of $\mathbb{F}_2^m$ the same number 2^{n-m} of times. Balanced (n, n)-functions are permutations on $\mathbb{F}_2^n$ [6].
- The **nonlinearity** N_F of an (n, m)-function F equals the minimum nonlinearity of all non-zero linear combinations $b \cdot F$ of its coordinate functions f_i, where $b \in \mathbb{F}_2^{m*}$ [27]:

  $$N_F = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n, v \in \mathbb{F}_2^{m*}} |W_F(a, v)|. \tag{15}$$

  The maximal nonlinearity of any (n, n) S-box is upper bounded by the following inequality:

  $$N_F \leq 2^{n-1} - 2^{\frac{n-1}{2}}. \tag{16}$$

- Let F be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$ and $a, b \in \mathbb{F}_2^n$. We denote:

  $$D(a, b) = |\{x \in \mathbb{F}_2^n : F(x + a) + F(x) = b\}|. \tag{17}$$

  The entry at the position (a, b) corresponds to the cardinality of $D(a, b)$ and is denoted as $\delta(a, b)$. The δ-**uniformity** δ_F is then defined as [3, 26]:

  $$\delta_F = \max_{a \neq 0, b} \delta(a, b). \tag{18}$$

Table 3: Nonlinearity value for an 8×8 S-box

Technique	best obtained nonlinearity
Random search [25]	98
Hill climbing [25]	100
Simulated annealing [7]	102
Genetic algorithm [33]	104
Finite field inversion [26]	112

- To define the **algebraic degree** of an S-box, we use the algebraic normal form (ANF) representation of a Boolean function f represented by a polynomial in $\mathbb{F}_2 [x_0, ..., x_{n-1}] / (x_0^2 - x_0, ..., x_{n-1}^2 - x_{n-1})$ [5]. The algebraic degree deg_F of an S-box F is the maximum algebraic degree of all non-zero linear combinations of the coordinate functions (i.e., component functions) or coordinate functions of F [6]:

  $$deg_F = \max_{b \in \mathbb{F}_2^{m*}} deg(b \cdot F). \tag{19}$$

5.4 Problems

We emphasize that the list of properties in Section 5.3 is far from complete. Examples of several problems are:

- Finding bijective S-boxes with as high as possible nonlinearity (single-objective optimization).
- Finding bijective S-boxes with as high as possible nonlinearity and as low as possible differential uniformity (multi-objective optimization).
- Finding bijective S-boxes with as high as possible nonlinearity, as low as possible differential uniformity, and as high as possible algebraic degree (multi-objective optimization).
- When m is strictly smaller than n, finding S-boxes where each coordinate function is bent (single-objective optimization).

As an example of diversity of techniques used here, we list several techniques used to find 8×8 S-box with as high as possible nonlinearity in Table 3. The finite field inversion method is an algebraic construction and we still need to find evolutionary algorithm/representation enabling us to reach that value.

6 PROBLEMS BASED ON PHYSICALLY UNCLONABLE FUNCTIONS

The basic idea of a Physically Unclonable Function (PUF) is to provide a function with a unique response (or a measurement) that can be used as a means of secure identification. This property is usually obtained by using a unique physical structure of each individual PUF. For this approach to be feasible, one commonly relies on inherent manufacturing differences that give each PUF instance a unique identity [20].

In this case we address the so called arbiter PUFs, which consist of one or more chains of two 2-bit multiplexers that have identical layouts (Figure 3). Each multiplexer pair is denoted a *stage*, with n stages in a single chain. There is a single *input signal* that is introduced to the first stage to both bottom and top multiplexer in the pair (red and blue). The chain is fed a control signal of n bits

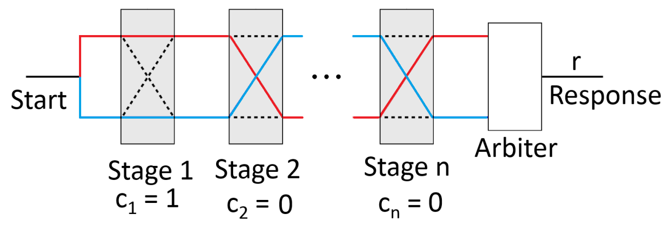


Figure 3: Depiction of an n-stage arbiter PUF.

called a *challenge* (bits c_1 to c_n), where each bit determines whether the two input signals in that stage would be switched (crossed over) or not.

In ideal conditions, the input signal would propagate at the same speed through each stage and both the lower and upper signal would arrive at the arbiter at the same time. However, due to the manufacturing inconsistencies, the delay of each multiplexer is slightly different, and the top and bottom input signals are not synchronized. The arbiter at the end of the chain determines which signal arrived earlier and thus forms the response (0 or 1). Additionally, to increase the resistance of PUFs to attacks, commonly several arbiter chains are placed on the chip and their outputs are XORed to build the final response bit.

The response of a PUF is determined by the delay difference between the top and bottom input signal, which is in turn the sum of delay differences of the individual stages. To efficiently model a PUF, one usually tries to determine the delay vector $w = (w_1, \dots, w_{n+1})$ which models the delay differences in each stage. The delay difference ΔD at the end of a chain can be calculated as

$$\Delta D = w^T \phi, \tag{20}$$

where the feature vector ϕ is derived from the challenge vector as

$$\phi_i = \prod_{l=i}^{n} (-1)^{c_l} \text{ for } 1 \leqslant i \leqslant n, \tag{21}$$

and where $\phi_{n+1} = 1$. The final response is equal to 1 if $\Delta D < 0$ and 0 otherwise.

The standard optimization approach with this problem would be to generate the pairs of challenges and corresponding responses and try to find the delay vector that exhibits the same behavior, e.g. by minimizing the number of wrong responses. While the real data from the actual PUFs can be used in this process, one can easily generate challenge and response pairs with a predefined or randomly created target delay vector.

This problem represents an excellent example of continuous optimization where the input data can be easily generated and accommodated for different *problem sizes.* For instance, a small scale instance is an example of a single-chain arbiter PUF with 64 stages, which is modeled with a floating-point vector of size 65. Larger instances can include multiple chains (e.g. 4) with as many as 256 stages in each chain, giving a search space of 4×257 variables. At the same time, by varying the number of challenges, the *problem hardness* can be modified; e.g. the desired responses for several tens of challenges are relatively easy to model, but finding a model for several thousand challenges is much more demanding, since it requires a greater precision in delay variables.

Table 4: Problem hardness for PUF modeling.

chains×stages/challenges	4x64/128	6x64/256	4x64/1 000
GA	0	1	221
GA/hybrid	0	1	102
CMA-ES	0	0	112
ClonAlg	0	19	218
OptIA	0	11	179

Table 5: Problem parameters influencing the search space size and complexity.

Problem type	Search space size	Complexity tunability
Boolean functions	# of inputs (2^{2^n})	# of properties
S-boxes	# of inputs and outputs (2^{m2^n})	# of properties
PUFs	# of chains and stages	# of challenges

Table 6: Problem properties and objective type.

Problem type	Properties	Range	Goal
Boolean functions	Balanced	$[yes, no]$	constraint
	Nonlinearity	$[0, 2^{n-1} - 2^{n/2-1}]$	maximization
	Algebraic degree	$[0, n-1]$	maximization
	Correlation immunity	$[0, n - deg - 1]$	maximization
	Algebraic immunity	$[0, \lceil n/2 \rceil]$	maximization
S-boxes	Balanced	$[yes, no]$	constraint
	Nonlinearity	$[0, 2^{n-1} - 2^{\frac{n-1}{2}}]$	maximization
	Algebraic degree	$[0, n-1]$	maximization
	Differential uniformity	$[2, 2^n]$	minimization
PUFs	error of the model	$[0, 1]$	minimization

Finally, since real-world PUFs incorporate noisy output, one can easily generate data with appropriate noise levels and try to minimize the error. This problem can also be addressed in the context of machine learning, where we use training and testing challenge/response pairs, and try to optimize the model to predict the unseen challenge data.

Table 4 illustrates the problem hardness for several algorithms and problem sizes, where the results are given as the number of incorrect responses; we report only the best result obtained by each algorithm. Since we do not conduct an algorithm comparison at this stage, we omit the algorithms' parameters and note we used the same number of evaluations as the stopping criteria. The algorithm denoted "GA/hybrid" is a combination of a classic genetic algorithm in which a local search strategy of Hooke-Jeeves is used to find a local minimum solution before each genetic operator.

7 CONCLUSION

In this paper, we present several realistic classes of problems from the cryptographic domain that could be used as a source of benchmark problems in the evolutionary computation field. Table 5 describes how the size of the search space and problem complexity can be tuned using the parameters of each presented benchmark example. With these problem classes, one can investigate various problem sizes, the number of cryptographic properties to be optimized, as well as the solution representations.

The results obtained from such tests can be used to fill the gaps in the current benchmarking but also to compare with other types of heuristics or even to propose new solutions that are better from the currently known ones. Such a scenario would offer additional benefit of increasing the level of confidence in evolutionary computation from different communities. Finally, although we present here only three problem classes to form the benchmark set, one can easily find several more interesting problems in the cryptographic domain.

ACKNOWLEDGMENTS

This work has been supported in part by Cybersecurity@CSAIL initiative. Additionally, this work has been supported in part by Croatian Science Foundation under the project IP-2014-09-4882.

REFERENCES

[1] Georg T. Becker. 2015. The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs. In *Cryptographic Hardware and Embedded Systems – CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, Tim Güneysu and Helena Handschuh (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 535–555.

[2] Guido Bertoni, Joan Daemen, Michäel Peeters, and Gilles Van Assche. 2011. The Keccak reference. (January 2011). http://keccak.noekeon.org/.

[3] Eli Biham and Adi Shamir. 1991. Differential Cryptanalysis of DES-like Cryptosystems. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '90)*. Springer-Verlag, London, UK, UK, 2–21.

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

[5] Claude Carlet. 2010. Boolean Functions for Cryptography and Error Correcting Codes. In *Boolean Models and Methods in Mathematics, Computer Science, and Engineering* (1st ed.), Yves Crama and Peter L. Hammer (Eds.). Cambridge University Press, New York, NY, USA, 257–397.

[6] Claude Carlet. 2010. Vectorial Boolean Functions for Cryptography. In *Boolean Models and Methods in Mathematics, Computer Science, and Engineering* (1st ed.), Yves Crama and Peter L. Hammer (Eds.). Cambridge University Press, New York, NY, USA, 398–469.

[7] John A. Clark, Jeremy L. Jacob, and Susan Stepney. 2005. The design of S-boxes by simulated annealing. *New Generation Computing* 23, 3 (Sept. 2005), 219–231. DOI: http://dx.doi.org/10.1007/BF03037656

[8] NicolasT. Courtois and Willi Meier. 2003. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - EUROCRYPT 2003*, Eli Biham (Ed.). Lecture Notes in Computer Science, Vol. 2656. Springer Berlin Heidelberg, 345–359. DOI: http://dx.doi.org/10.1007/3-540-39200-9_21

[9] Joan Daemen, René Govaerts, and Joos Vandewalle. 1994. A new approach to block cipher design. In *Fast Software Encryption: Cambridge Security Workshop Cambridge, U. K.,1993 Proceedings*, Ross Anderson (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 18–32.

[10] Joan Daemen and Vincent Rijmen. 2002. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[11] W. Diffie and M. Hellman. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* 22, 6 (1976), 644–654.

[12] FIPS 1999. FIPS 46-3, Data Encryption Standard (DES). National Institute for Standards and Technology (NIST), Gaithersburg, MD, USA. (1999).

[13] Xiao Guo-Zhen and J.L. Massey. 1988. A spectral characterization of correlation-immune combining functions. *IEEE Transactions on Information Theory* 34, 3 (May 1988), 569–571.

[14] Radek Hrbacek and Vaclav Dvorak. 2014. Bent Function Synthesis by Means of Cartesian Genetic Programming. In *Parallel Problem Solving from Nature - PPSN XIII*, Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith (Eds.). Lecture Notes in Computer Science, Vol. 8672. Springer International Publishing, 414–423. DOI: http://dx.doi.org/10.1007/978-3-319-10762-2_41

[15] Domagoj Jakobovic. 2014. Evolutionary Computation Framework. http://ecf.zemris.fer.hr/. (Jan. 2014). http://ecf.zemris.fer.hr/ Framework available for download.

[16] Domagoj Jakobovic. 2017. Heuristic optimization in cryptology EvoCrypt. http://evocrypt.zemris.fer.hr/. (May 2017). http://evocrypt.zemris.fer.hr/ Problem definitions available for download.

[17] Frédéric Lafitte, Dirk Van Heule, and Julien Van Hamme. 2011. Cryptographic Boolean Functions with R . *The R Journal* 3, 1 (jun 2011), 44–47. http://journal.r-project.org/archive/2011-1/RJournal_2011-1_Lafitte~et~al.pdf

[18] G. Leander and A. Poschmann. 2007. On the Classification of 4 Bit S-Boxes. In *Arithmetic of Finite Fields*, Claude Carlet and Berk Sunar (Eds.). Lecture Notes in Computer Science, Vol. 4547. Springer Berlin Heidelberg, 159–176.

[19] F. Jessie MacWilliams and Neil J. A. Sloane. 1977. *The Theory of Error-Correcting Codes*. Elsevier, Amsterdam, North Holland. 782 pages. ISBN: 978-0-444-85193-2.

[20] Roel Maes and Ingrid Verbauwhede. 2010. Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In *Towards Hardware-Intrinsic Security: Foundations and Practice*, Ahmad-Reza Sadeghi and David Naccache (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–37.

[21] Luca Mariot and Alberto Leporati. 2015. Heuristic Search by Particle Swarm Optimization of Boolean Functions for Cryptographic Applications. In *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*. 1425–1426.

[22] Mitsuru Matsui and Atsuhiro Yamagishi. 1993. A new method for known plaintext attack of FEAL cipher. In *Proceedings of the 11th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT'92)*. Springer-Verlag, Berlin, Heidelberg, 81–91.

[23] James McLaughlin and John A. Clark. 2013. Evolving balanced Boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree, and very high nonlinearity. Cryptology ePrint Archive, Report 2013/011. (2013).

[24] Willi Meier, Enes Pasalic, and Claude Carlet. 2004. Algebraic Attacks and Decomposition of Boolean Functions. In *Advances in Cryptology - EUROCRYPT 2004*, Christian Cachin and JanL. Camenisch (Eds.). Lecture Notes in Computer Science, Vol. 3027. Springer Berlin Heidelberg, 474–491. DOI: http://dx.doi.org/10.1007/978-3-540-24676-3_28

[25] W. Millan, L. Burnett, G. Carter, A. Clark, and E. Dawson. 1999. Evolutionary Heuristics for Finding Cryptographically Strong S-Boxes. In *Information and Communication Security*, Vijay Varadharajan and Yi Mu (Eds.). Lecture Notes in Computer Science, Vol. 1726. Springer Berlin Heidelberg, 263–274. DOI: http://dx.doi.org/10.1007/978-3-540-47942-0_22

[26] Kaisa Nyberg. 1991. Perfect Nonlinear S-Boxes. In *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings (Lecture Notes in Computer Science)*, Vol. 547. Springer, 378–386. DOI: http://dx.doi.org/10.1007/3-540-46416-6_32

[27] Kaisa Nyberg. 1993. On the construction of highly nonlinear permutations. In *Advances in Cryptology - EUROCRYPT' 92*, RainerA. Rueppel (Ed.). Lecture Notes in Computer Science, Vol. 658. Springer Berlin Heidelberg, 92–98.

[28] Christof Paar and Jan Pelzl. 2010. *Understanding Cryptography - A Textbook for Students and Practitioners*. Springer. I–XVIII, 1–372 pages.

[29] Stjepan Picek. 2015. *Applications of evolutionary computation to cryptology*. Ph.D. Dissertation. Radboud University Nijmegen, The Netherlands. http://repository.ubn.ru.nl/bitstream/handle/2066/141872/141872.pdf

[30] Stjepan Picek. 2016. Evolutionary Computation and Cryptology. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM, 883–909.

[31] Stjepan Picek, Lejla Batina, Domagoj Jakobovic, Bars Ege, and Marin Golub. 2014. S-box, SET, Match: A Toolbox for S-box Analysis. In *Information Security Theory and Practice. Securing the Internet of Things*, David Naccache and Damien Sauveron (Eds.). Lecture Notes in Computer Science, Vol. 8501. Springer Berlin Heidelberg, 140–149. DOI: http://dx.doi.org/10.1007/978-3-662-43826-8_10

[32] Stjepan Picek, Claude Carlet, Sylvain Guilley, Julian F Miller, and Domagoj Jakobovic. 2016. Evolutionary Algorithms for Boolean Functions in Diverse Domains of Cryptography. *Evolutionary computation* 24, 4 (2016), 667–694.

[33] Stjepan Picek, Marko Cupic, and Leon Rotim. 2016. A New Cost Function for Evolution of S-Boxes. *Evolutionary Computation* 24, 4 (2016), 695–718.

[34] Stjepan Picek, Domagoj Jakobovic, Julian F. Miller, Lejla Batina, and Marko Cupic. 2016. Cryptographic Boolean functions: One output, many design criteria. *Appl. Soft Comput.* 40 (2016), 635–653.

[35] Stjepan Picek, Domagoj Jakobovic, Julian F. Miller, Elena Marchiori, and Lejla Batina. 2015. Evolutionary Methods for the Construction of Cryptographic Boolean Functions. In *Genetic Programming - 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings*. 192–204.

[36] Stjepan Picek, Dominik Sisejkovic, and Domagoj Jakobovic. 2016. Immunological algorithms paradigm for construction of Boolean functions with good cryptographic properties. *Engineering Applications of Artificial Intelligence* (2016).

[37] Stjepan Picek, Bohan Yang, and Nele Mentens. 2016. A Search Strategy to Optimize the Affine Variant Properties of S-Boxes. In *Arithmetic of Finite Fields: 6th International Workshop, WAIFI 2016, Ghent, Belgium, July 13-15, 2016, Revised Selected Papers*, Sylvain Duquesne and Svetla Petkova-Nikova (Eds.). Springer International Publishing, Cham, 208–223.

[38] T. Siegenthaler. 2006. Correlation-immunity of Nonlinear Combining Functions for Cryptographic Applications (Corresp.). *IEEE Transactions on Information Theory* 30, 5 (Sept. 2006), 776–780.

[39] William A. Stein and others. 2013. *Sage Mathematics Software (Version 5.10)*. The Sage Development Team. `http://www.sagemath.org`.