TU Delft

Delft University of Technology

Document Version
Final published version

Citation (APA)
Mariot, L., Jakobovic, D., Bäck, T., & Hernandez-Castro, J. C. (2022). Artificial Intelligence for the Design of Symmetric Cryptographic Primitives. In L. Batina, T. Back, & S. Picek (Eds.), *Security and Artificial Intelligence* (pp. 3-24). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 13049). Springer. https://doi.org/10.1007/978-3-030-98795-4_1

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.

Artificial Intelligence for the Design of Symmetric Cryptographic Primitives

Luca Mariot[1(✉)], Domagoj Jakobovic[2], Thomas Bäck[3], and Julio Hernandez-Castro[4]

[1] Cyber Security Research Group, Delft University of Technology, Delft, The Netherlands
L.Mariot@tudelft.nl
[2] Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia
domagoj.jakobovic@fer.hr
[3] Leiden University, Leiden, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl
[4] School of Computing, University of Kent, Canterbury, UK
J.C.Hernandez-Castro@kent.ac.uk

Abstract. This chapter provides a general overview of AI methods used to support the design of cryptographic primitives and protocols. After giving a brief introduction to the basic concepts underlying the field of cryptography, we review the most researched use cases concerning the use of AI techniques and models to design cryptographic primitives, focusing mainly on Boolean functions, S-boxes and pseudorandom number generators. We then point out two interesting directions for further research on the design of cryptographic primitives where AI methods could be applied in the future.

1 Introduction

Cryptography can be broadly defined as a discipline that studies how to enable *secure communication* between two or more parties in the presence of *adversaries*. Historically, cryptography has been associated with *encryption*, which aims at protecting the *confidentiality* of messages transmitted over an insecure channel. On the opposite side, the goal of *cryptanalysis* is to analyze a particular encryption scheme to search for eventual vulnerabilities that can be exploited to attack the scheme and violate message confidentiality. Collectively, *cryptology* encompasses both the fields of cryptography and cryptanalysis.

Modern cryptography stands on the use of precise mathematical definitions and rigorous proofs to guarantee a certain security level under a particular model of the adversary's strategy. As such, designing a sound cryptographic primitive or protocol is usually a hard task, as well as it is cryptanalyzing it. In this respect, *Artificial Intelligence* (AI) provides a host of interesting approaches and tools to address problems in the design of cryptographic schemes. By looking at the existing literature, one can find many works that use various approaches from the field

© Springer Nature Switzerland AG 2022
L. Batina et al. (Eds.): Security and Artificial Intelligence, LNCS 13049, pp. 3–24, 2022.
https://doi.org/10.1007/978-3-030-98795-4_1

of AI to address several use cases relevant to cryptography. One may classify such works in two main areas, depending on the nature of the underlying problem:

Search and Optimization. Several questions in the design of cryptographic primitives can be cast as combinatorial optimization problems over a discrete search space, such as, among others, the search of *Boolean functions* and *S-boxes* with desirable cryptographic properties, which are fundamental building blocks in the design of symmetric encryption schemes. To this end, AI-based heuristic techniques such as *Evolutionary Algorithms* [51], *Simulated Annealing* [14] and *Swarm Intelligence* [37] have proved to be quite useful to tackle optimization problems related to cryptography.

Computational Models. The second area concerns the use of computational models belonging to the domain of AI as components in the design of cryptographic schemes. In this case, the underlying idea is to link the overall scheme's security to the complex dynamic behavior of such computational models, which in principle are difficult to cryptanalyze. Perhaps the best known examples in this research thread are *Cellular Automata*, which have been mainly studied to design symmetric encryption primitives such as Pseudorandom Number Generators (PRNG) for stream ciphers [66,67] and S-boxes for block ciphers [20,39].

This chapter aims to provide a broad overview of the state of the art concerning the use of AI methods and models for designing cryptographic primitives and protocols, focusing on the two areas mentioned above. In particular, we consider the most significant use cases of AI-based cryptography, namely the design of Boolean functions, S-boxes, and Pseudorandom Number Generators (PRNG). For each use case, we introduce the corresponding cryptographic design problem, and then we give an overview of the related literature. We conclude by considering two new directions along which this research field could evolve in the next years.

The rest of this chapter is organized as follows. Section 2 gives a concise introduction to the basic notions of cryptography, and covers the basic notions related to AI-based heuristic techniques and cellular automata. Sections from 3, 4 and 5 focus on AI techniques used to design cryptographic primitives, addressing, in particular, the use cases of Boolean functions, S-boxes and pseudorandom number generators. Finally, Sect. 6 concludes the chapter by discussing open problems and directions for future research in the field of AI-based cryptography.

2 Background

In this section, we first illustrate the basic terminology and concepts related to cryptography over which we ground the discussion of the next sections. Next, we introduce the basic AI techniques and models that have been used in the literature to design cryptographic primitives, namely heuristic optimization techniques and cellular automata. Clearly, given the broad scopes of these research fields, a complete treatment is well beyond the scope of this chapter. For more information on the topic, we refer the interested reader to standard textbooks such as [27,64] for cryptography and [17,23] respectively for heuristic optimization algorithms and cellular automata.

2.1 Cryptography

As we mentioned in the Introduction, one of the goals of secure communication is *confidentiality*, which ensures that only the intended recipient is able to read a particular message. The main tools studied in cryptography to achieve this goal are *encryption schemes*, which are applied to the following basic communication scenario. Suppose that a *sender*, Alice, wants to send a *plaintext message* P to a *receiver*, Bob, over a communication channel. In particular, the plaintext message P can be thought of as a finite string over an alphabet Σ. However, the communication channel is eavesdropped by an *adversary*, Eve, who can intercept and read everything transmitted over it. To solve this problem, Alice and Bob adopt the encryption scheme depicted in Fig. 1, which works as follows. Alice first feeds P in input to an *encryption function*, which also depends on an *encryption key* K_E. The encryption function output is a *ciphertext* C that Alice sends to Bob over the insecure channel tapped by Eve. On his end, Bob applies a *decryption function* to the received ciphertext, which similarly depends on a *decryption key* K_D, and whose output is the original plaintext message P. The encryption and decryption functions must be the inverses of one another so that Bob can decode the correct message from the ciphertext.

The confidentiality property of this scheme rests on the assumption that Eve cannot recover the plaintext message P by just observing the ciphertext C transmitted over the communication channel. In particular, *Kerchoff's principle* states that the security of an encryption scheme should not rely on the secrecy of the encryption and decryption functions used by Alice and Bob, but rather only on the secrecy of the decryption key K_D. Thus, one may assume that Eve knows the encryption and decryption functions, so they must be designed not to leak any useful information on the plaintext message or the decryption key if the latter is not known.

Depending on the nature of the keys employed by Alice and Bob, one can classify encryption schemes in *symmetric* and *asymmetric* ones. In a symmetric encryption scheme the same key is used both for encryption and decryption, i.e. one has that $K_E = K_D = K$. Since this key must be kept secret, Alice

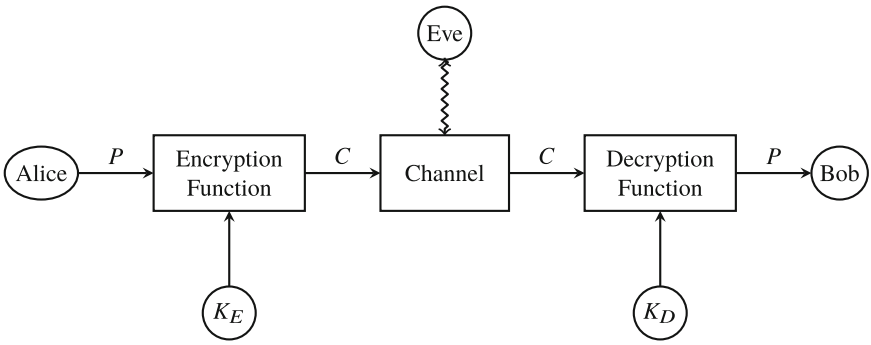


Fig. 1. Block diagram of a generic encryption scheme.

and Bob have to figure out a way to share it securely before the communication takes place. Symmetric encryption schemes can be further divided in *block ciphers* and *stream ciphers*. In a stream cipher, each symbol of the plaintext P is combined with a corresponding symbol of a *keystream* z, computed from the initial secret key k through a *keystream generator algorithm*. Perhaps the most widely studied model in this context is the *Vernam-like cipher*, where the plaintext, the keystream and the ciphertext are all bitstrings of the same length, and the encryption operation corresponds to the bitwise XOR between the plaintext and the keystream (see Fig. 2a). In a block cipher, on the contrary, the plaintext is processed in fixed-size *blocks*, that are iteratively combined with several round keys derived from the secret key through a scheduling algorithm. One of the most common paradigms for the design of block ciphers is the *Substitution-Permutation Network* (SPN, see Fig. 2b). In this case, the plaintext block undergoes first a *confusion phase*, followed by a *diffusion phase* and finally by the *key combination phase* with the current round key. This process is repeated for a certain number of *rounds*.

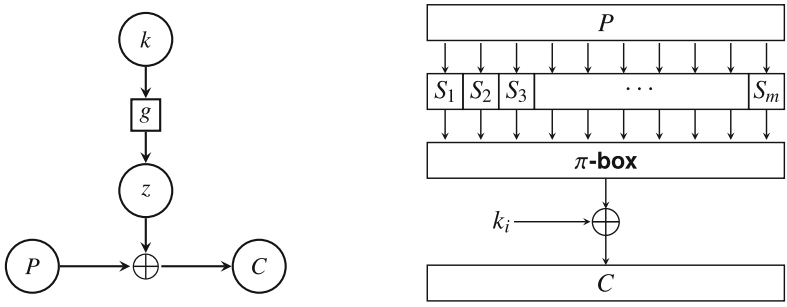
Confusion and *diffusion* are two general principles stated by Shannon [62] that every symmetric encryption scheme should satisfy, to frustrate statistical attacks. In particular, the aim of the confusion phase in an SPN cipher is to make the relationship between the plaintext and the secret key the most complicated as possible. This property is accomplished by processing the plaintext block through a set of smaller *Substitution boxes* (S-Boxes), which we will discuss more in detail in Sect. 4. On the other hand, the goal of the diffusion phase is to spread the statistical structure of the plaintext over the ciphertext, so that each symbol of the ciphertext depends on many symbols of the plaintext. In a SPN cipher this is done by using a *permutation box* (π-box, also called a *diffusion layer*) right after the confusion phase.

In an asymmetric (or *public-key*) encryption scheme the keys used for encryption and decryption are distinct. Typically, there are no key-sharing issues in this case since Bob can make its encryption key public while he keeps only the decryption key private. Alice will then use Bob's public key to encrypt the plaintext and send him the corresponding ciphertext.

Most of the applications of AI techniques to cryptography concern the symmetric setting, although a few works also address the public-key case (see e.g. [1]). Therefore, in the next sections, we will focus on the use cases related to symmetric cryptography. Also, in this chapter, we will not consider AI-based approaches to solve other aspects of secure communication such as message integrity and authentication, since the mass of the existing literature about AI methods in cryptography is focused on encryption and confidentiality.

The security of an encryption scheme is usually analyzed in terms of adversarial goals, attack models and security levels. The *adversarial goals* specify when a particular attack performed by Eve can be considered successful. In particular, in a *total break* of the encryption scheme the goal is to recover the decryption key, while in a *partial break* Eve is able to decrypt ciphertexts with a certain probability, without knowing the decryption key. Finally, Eve achieves a

distinguishing break if is she can tell apart the encryption of two distinct ciphertexts (usually corresponding to a plaintext chosen by Alice and a random one) with a probability higher than 1/2.



(a) Vernam-like Stream Cipher

(b) SPN Block Cipher round

Fig. 2. Encryption diagrams for Vernam-like stream ciphers and SPN block ciphers.

The attack models define the type of information available to Eve to perform an attack on an encryption scheme. These vary from very weak assumptions, as in the *ciphertext-only attacks* where Eve only knows some ciphertexts encrypted under the same unknown key, up to more sophisticated ones such as *chosen-ciphertext attacks*, in which Eve can choose some ciphertexts and obtain the corresponding plaintexts decrypted under the same unknown key.

The *security level* models the *computational resources* that Eve has at her disposal to carry out a particular attack. In particular, in the *unconditional security* level an encryption scheme cannot be broken under a particular attack model, even if the adversary has unlimited computational resources. While this is the most robust definition that a cryptographer can adopt, unconditional security usually yields encryption schemes that are hardly usable in practice, typically because the encryption key has the same length of the plaintext and the ciphertext. The *provable security* level is often employed in the design of public-key encryption schemes. Here, the goal is to reduce a particular computational problem assumed to be hard (such as factoring integer numbers or computing discrete logarithms) to the task of breaking the encryption scheme. Finally, in the *computational security* level the best known attack that Eve can perform on an encryption scheme requires at least N operations, where N is a very large number. In particular, a scheme can be shown to be computationally secure only for certain specific attacks, and there is no guarantee that it is not vulnerable to others. Despite this drawback, computational security is perhaps the most widespread security level considered in the design of *cryptographic primitives*, especially in the symmetric key setting. One of the reasons is that low-level cryptographic primitives are simpler to analyze, and few classes of attacks are known on them. Thus, it makes sense from a practical point of view to defend these primitives in light of such attacks. Further, this is the security level under-

lying most of the literature of AI-based cryptography and cryptanalysis, which is why we will mainly consider it in the rest of this chapter.

2.2 Heuristic Optimization Algorithms

Optimization can be generally defined as the process of searching for a best solution to a particular problem. Formally, one has a finite set S equipped with an *objective function* $f : S \to \mathbb{R}$ assigning to each *candidate solution* $x \in S$ a measure of how good x is in solving a particular problem instance. The goal of *combinatorial optimization* is to find an *optimal solution* x^* that maximizes f, that is, $x^* = argmax_{x \in S}\{f(x)\}$.

In most combinatorial optimization problems of practical importance, the solution space S is usually too huge to be explored in an exhaustive manner. One of the possible ways to address this shortcoming is to resort to *heuristic optimization algorithms*, in order to find a (sub)optimal solution in a reasonable amount of time. Several heuristic techniques traditionally belong to the field of AI. The general characteristic shared by these techniques is that they are *iterative algorithms*, i.e., they start from an initial solution and iteratively tweak it using the objective function to drive the search. After a certain number of evaluations, the best solution found so far is returned.

We now give a short overview of the main heuristic algorithms that have been used to design cryptographic primitives, which of course is far from being exhaustive. The reader is referred to [2,10,17] for more comprehensive treatments of this topic.

Single-State Optimization Methods. The idea underlying single-state optimization methods is to optimize a single solution at a time, and to search in its *neighborhood* to find a solution with a better objective function value. This approach assumes that a topology is defined on the solution space, usually induced by the Hamming distance in the case of binary strings. Two of the most widely used single-state optimization methods in cryptography are *Hill Climbing* (HC) and *Simulated Annealing* (SA). Hill Climbing replaces the current solution whenever another one having a better objective function value is sampled in the neighborhood. This strategy tends however to get stuck in *local optima*. To overcome this drawback of HC, simulated annealing accepts with a certain probability a *worse* solution in the neighborhood than the current one. The acceptance probability in SA is controlled by a *temperature* parameter, which is decreased throughout the optimization process. Hence, at the beginning SA favors *exploration* of the search space, making the escape from local optima more likely. In later iterations, the algorithm focuses on *exploitation* of the current region of the search space.

Population-Based Optimization Methods. In population-based optimization methods the main idea is to optimize a set of candidate solutions instead of a single one. This approach allows for a more global search on the solution space, decreasing the risk of getting stuck in local optima.

In this domain, the most popular methods are *Evolutionary Algorithms* (EA), which are loosely inspired by the principles of biological evolution. Each individual in the population is represented by a *genotype* which decodes to a candidate solution of the search space, also called a *phenotype* in this context. At each iteration, an EA manipulates the genotypes of the individuals in the population by following a three-step process. First, the objective function (also called a *fitness function* in this case) is evaluated on all individuals. Then, a *selection operator* is used to choose the individuals that will reproduce in the next generation, using their fitness values to drive the selection in a probabilistic way. Next, *variation operators* are used to create the new generation from the selected individuals. This step usually consists of a *crossover operator*, where the genotypes of two parent individuals is mixed to create an offspring, and then a *mutation operator* is applied on the offspring to introduce random variation in its genotype.

Several variants of EA have been considered in the literature depending on the encoding adopted for the genotypes and the variation operators used to modify them. For example, in *Genetic Algorithms* (GA), the genotypes are usually encoded by fixed-length bitstrings, but integer vectors and permutations have also been used. In *Genetic Programming* (GP), on the other hand, the aim is to evolve syntactically correct programs that are usually represented by trees.

Finally, another type of population-based heuristics used in cryptography are *Swarm Intelligence methods*, among which *Particle Swarm Optimization* (PSO) is the most common one. In PSO, the elements of the population are *particles* that move over the search space. Each particle is described by its current position and velocity. During a single PSO iteration, each particle updates its position by adding its velocity. Then, the velocity controlled by taking into account the *global best* solution found so far in the particle's neighborhood and the *local best* solution found so far by the particle itself.

2.3 Cellular Automata

A *cellular automaton* (CA) is a discrete computational model described by a regular lattice of *cells*, where each cell updates itself by considering its current state and those of the neighboring cells. In its simplest form, a CA is defined by a one-dimensional array of n binary cells and by a *local rule* $f : \{0,1\}^d \to \{0,1\}$ of *diameter* $d \leq n$. The update of the array's global state is performed by evaluating in parallel f on each cell and the neighborhood formed by the $d-1$ cells on its right (although variations on the center of the neighborhood exist, e.g. by considering also the cells on the left). There are several approaches to update the cells at the boundaries that do not have enough neighbors to apply the local rule. One common solution is to consider the array as a ring, with the first cell following the last one. This induces a *global rule* $F : \{0,1\}^n \to \{0,1\}^n$ that defines the next state of the CA array as:

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

for all $(x_1, x_2, \cdots, x_n) \in \{0,1\}^n$. Thus, a CA can be viewed as a vectorial function defined by a *shift-invariant coordinate function*, with periodic boundary con-

ditions. Traditionally, CA have been studied in the domain of *natural computing*, and thus belong to the larger class of nature-inspired AI computational models. The reason underlying the interest of CA for implementing cryptographic applications is twofold. First, the shift-invariance property that characterizes CA allows for uniform and efficient hardware implementations. Second, depending on the underlying local rule, the dynamic evolution of a CA can be quite complex and unpredictable. We point the reader to [26] for a broader introduction to the basic concepts concerning CA.

3 Boolean Functions

Boolean functions are a fundamental class of cryptographic primitives used in the design of stream and block ciphers. This section gives a brief introduction to Boolean functions and the optimization problems related to their cryptographic properties. Next, we survey the main works in the literature addressing the design of cryptographically strong Boolean functions using AI techniques.

3.1 Background

One of the most common models for the keystream generator in the Vernam-like stream cipher defined in Sect. 2.1 is the *combiner model*, where the bits produced by several *Linear Feedback Shift Registers* (LFSR) are combined by a *Boolean function*, whose output is used as the next keystream bit. The computational security of this model can be reduced to the study of the combiner function's cryptographic properties, the reason being that some attacks become more efficient if the function does not meet certain criteria.

We now give a concise overview of the properties that Boolean functions used as cryptographic primitives in stream and block ciphers should satisfy, referring the reader to [6] for a thorough treatment of the subject. In what follows, we denote by $\mathbb{F}_2 = \{0, 1\}$ the finite field with two elements, and $\mathbb{F}_2^n$ the set of n-bit vectors endowed with a vector space structure, with bitwise XOR as the vector sum and logical AND as the multiplication by a scalar. A Boolean function of n variables is then defined as a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2$. There are mainly three unique representations of Boolean functions used in cryptography, the most natural one being the truth table. Assuming that the vectors of $\mathbb{F}_2^n$ are totally ordered, the *truth table* of a function is a 2^n-bit vector Ω_f that specifies the corresponding output value $f(x)$ for each vector $x \in \mathbb{F}_2^n$. The *Algebraic Normal Form* (ANF) represents a Boolean function as a multivariate polynomial over $\mathbb{F}_2$, i.e., as a sum (XOR) of monomials, where each monomial is the product (AND) of a subset of input variables. Finally, the *Walsh transform* represents a Boolean function f in terms of *correlations* with *linear functions*, which are defined as the XOR over a subset of the input variables. Hence, each monomial in their ANF is composed of a single variable.

These three representations can be used to define several cryptographic properties for Boolean functions used in stream ciphers. We briefly mention the four most important criteria below:

- *Balancedness*: the truth table of f should be composed of an equal number of zeros and ones. Unbalanced functions present a bias that can be exploited in statistical attacks.
- *Algebraic degree*: the degree of the polynomial represented by the ANF of f should be as high as possible, to avoid attacks that exploit the low linear complexity of the LFSRs used in the combiner model.
- *Nonlinearity*: the distance of the truth table of f from the set of all linear functions should be as high as possible. This property can be measured by considering the highest absolute value in the Walsh transform of f. Functions with low nonlinearity can be vulnerable to fast-correlation attacks in stream ciphers.
- *t-th order correlation immunity*: Each subset of t or fewer variables should be statistically independent from the output value of f. This condition is met if and only if the Walsh transform vanishes for all input vectors with at most t ones. Functions used in the combiner model should be correlation immune of a high order to avoid *correlation attacks*.

As we touched upon earlier, all the above properties are always considered in a computational security setting. Therefore, using a Boolean function that satisfies a subset of these properties protects from the specific attacks for which they are tailored, but does not grant security concerning other attacks. Moreover, as there exist several theoretical bounds among these properties, most of them cannot be satisfied simultaneously. The problem thus becomes to select a Boolean function with a suitable trade-off of the most important criteria. Finding such a function can be formulated as a *combinatorial optimization problem*: given the desired number of variables n, the designer's goal is to find an n-variable Boolean function which is balanced and has the highest possible degree, nonlinearity and order of correlation immunity. However, this problem cannot be solved by enumerating all Boolean functions of n variables. Indeed, the size of the resulting search space is 2^{2^n}, which hinders exhaustive search already for $n = 6$ variables. To give an intuition of the search space size, for $n = 8$ variables, there exist $2^{256} \approx 1.16 \cdot 10^{77}$ functions, which is approximately equal to the number of atoms in the observable universe. Since in practical stream ciphers a larger number of variables is required (at least $n = 13$ for the functions in the combiner model), the designer needs to resort either to *algebraic constructions* (for which the reader can refer to [6]) or to *heuristic optimization algorithms*, which include the AI-based methods introduced in Sect. 2.2. There are two main reasons for using heuristic algorithms to find proper Boolean functions in place of algebraic constructions. The first one is *diversity*: although algebraic constructions are easy to use and there is a formal proof that they satisfy specific cryptographic properties, the resulting functions usually belong to only a few known classes under affine equivalence relations.

On the other hand, heuristic algorithms from the AI domain are "blind", that is, they base their search solely upon the optimization of an objective function. Hence, heuristic algorithms can provide a wider variety of optimal Boolean functions, and in principle, they can discover new ones belonging to previously

unknown classes. The second reason is that heuristic algorithms are more flexible than algebraic constructions. Indeed, the designer can take into account more properties of interest by merely combining the appropriate terms to optimize in the objective function. Notice also that these properties do not need to be all related to cryptographic criteria: one may also include, for instance, properties concerning the *implementation costs* of the resulting Boolean functions in hardware, such as gate count or circuit area.

3.2 Survey of Related Works

Most of the AI-based approaches that tackle the search of Boolean functions and S-boxes with good cryptographic properties revolve around the use of genetic algorithms (GA) and genetic programming (GP). Especially among earlier works, one can see that simulated annealing (SA) was also a quite popular technique employed on this problem. At the same time, a minor research thread considered the use of swarm intelligence optimization algorithms such as particle swarm optimization (PSO).

Historically, GA have been extensively investigated to evolve the truth tables of Boolean functions with good cryptographic properties. The first attempt in this area can be traced back to Millan et al. [44]. There, the authors used the classic bitstring representation of GA to encode the truth table of Boolean functions, to maximize their nonlinearity. The same authors refined their approach in [45] by forcing the GA to generate only balanced Boolean functions, through the use of a custom crossover operator. The authors further combined their GA with a hill climbing step, and the fitness function maximized the correlation immunity order, besides the nonlinearity of the candidate Boolean functions. The idea of designing ad-hoc crossover operators to reduce the space explored by GA when searching for cryptographic Boolean functions has been later adopted in other works [36,38]. However, its advantage over classic operators such as one-point crossover has been systematically confirmed only recently by Manzoni et al. [33].

The first work considering the use of SA to generate Boolean functions with high nonlinearity is that of Clark et al. [11], where the authors adopted a two-stage optimization approach. In the first stage, SA was used to drive the search into a region containing highly nonlinear Boolean functions, which were then located in the second stage using a hill climbing algorithm. Successively, SA has also been investigated by Clark et al. in [14], where correlation immunity was also considered in the optimization process, and by Clark et al. in [12], where the authors adopted a different representation for the candidate solutions. In particular, they proposed to use the *spectral inversion method*, where the candidate solutions are represented as Walsh spectra already satisfying specific properties (e.g., vanishing for all coefficients with at most t ones to meet t-th order correlation immunity). The optimization objective is to minimize the deviation of the *pseudo-Boolean function* resulting from the inverse Walsh transform applied on the spectra searched by SA; when this deviation is zero, the corresponding map is a Boolean function with the desired cryptographic criteria. The authors adopted this approach to design *plateaued* Boolean functions, which satisfy the

best possible trade-off among algebraic degree, nonlinearity, balancedness, and correlation immunity. More recently, Mariot et al. [36] investigated the spectral inversion method with GA, remarking that it is more efficient than SA in finding plateaued Boolean functions.

The approach considering swarm intelligence algorithms also gave interesting results and insights in this optimization problem. However, this research line is not as developed as those based on GA and SA. Saber et al. [60] used a Particle Swarm Optimizer (PSO) to evolve Walsh spectra of plateaued functions, leveraging on the spectral inversion method of Clark et al. mentioned above. In particular, the position of a particle is updated by permuting the coefficients of the Walsh spectrum. Interestingly, this method allowed them to discover balanced Boolean functions of 9 variables with nonlinearity 240, algebraic degree 5, and correlation immunity order 3, whose existence question was open until then. Mariot et al. [37] designed a discrete PSO algorithm to search for Boolean functions with a good trade-off of nonlinearity and correlation-immunity, using the truth table encoding. The position update operation only swaps the bits in the truth table, to preserve balancedness.

To date, Genetic Programming turned out to be the most successful optimization technique for designing Boolean functions with good cryptographic properties. Although GP shares the same evolutionary algorithm structure of GA, the candidate solutions are *computer programs* instead of bitstrings, which are usually represented as trees. In this case, a Boolean function is encoded by a tree where the leaves represent the input variables, while the internal nodes are Boolean operators, and the root node gives the output of the function. The truth table is thus obtained by evaluating the GP tree over all possible assignments of the input variables on the leaf nodes and then taking the corresponding output value on the root. Castro et al. [7] were the first to address the design of Boolean functions for cryptographic applications. In particular, the authors considered the *average avalanche effect* as an optimization criterion, which is relevant for Boolean functions used in cryptographic hash functions. Picek et al. [50] applied GP to the evolution of Boolean functions with good cryptographic properties, comparing its performance with GA. Successively, Picek et al. [54] experimented with several EA algorithms, including GP, and combined them with algebraic constructions to investigate the maximum nonlinearity value achievable by balanced Boolean functions of 8 variables, which is today still an open question. More recently, Picek et al. [51] performed a systematic comparison of four evolutionary algorithms under three different fitness functions, that took into account several combinations of cryptographic properties of Boolean functions. The results showed that *Cartesian* GP (a GP variant where the solutions are represented as graphs, instead of trees) obtained the best results.

Pushing the hybrid approach of [54] further, Picek et al. [49] used GP to directly evolve secondary algebraic constructions of *bent* functions, which are those Boolean functions reaching the highest possible nonlinearity value. Although they exist only for even numbers of variables, and they are always unbalanced, bent functions can be used to construct highly nonlinear balanced

functions (see [6] for an overview of related methods). The search for bent functions with heuristic optimization algorithms is nonetheless an interesting problem, even though bent functions cannot be directly used in the design of stream ciphers. Using this approach, the authors of [49] were able to construct bent Boolean functions of up to 24 variables, which are practically impossible to find using direct search methods based on more traditional representations such as the truth table encoding. Along this research direction, we also mention the work by Hrbacek et al. [22], who evolved bent functions up to 16 variables with Cartesian GP. Picek et al. [53], on the other hand, used GA ad GP to design *quaternary* bent functions, that can be turned into bent Boolean functions of a larger number of variables. Finally, Mariot et al. [34] applied GA and GP to search for *hyper-bent* functions. Hyper-bent functions are a subclass of bent Boolean functions at the highest possible distance from the set of linear functions represented by bijective monomials. This property is useful to thwart approximation attacks.

The merits of evolutionary algorithms concerning the property of correlation immunity have been investigated by Picek et al. in [46], where the authors compared the performances of GA, GP, and Cartesian GP in the design of correlation immune functions. Besides its utility in stream ciphers, correlation immunity is relevant in masking countermeasures for *side-channel attacks*. In this context, the goal is to find a t-th order correlation immune function with minimal *Hamming weight* (i.e., with the lowest possible number of ones in the truth table), in order to minimize the implementation cost of the countermeasure. To this end, Picek et al. [48] employed GP to design low-weight Boolean functions with various orders of correlation immunity. Correlation immune functions can also be defined in terms of different combinatorial objects, such as binary *Orthogonal Arrays* (OA). Indeed, the rows of an OA form the *support* (that is, the set of input vectors mapping to 1) of a correlation immune function. Therefore, a different approach to designing low-weight correlation immune functions of order t for side-channel countermeasures is to construct binary OA of strength t with the smallest number of rows N possible. Along this direction, Mariot et al. [38] investigated the use of GA and GP to evolve binary OA of various sizes, remarking that GP is much more effective at generating them.

4 S-Boxes

S-boxes, which we already introduced in Sect. 2.1 as cryptographic primitives for block ciphers based on the SPN paradigm, are the parallelization of several Boolean functions computed on the same input vector. In this section, we cover the background notions concerning S-boxes. Then, we overview the literature about using AI methods to construct S-boxes with good cryptographic properties.

4.1 Background

An S-box is a vectorial mapping $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, mapping n-bit to m-bit vectors, also denoted as an (n, m)-function. In SPN block ciphers, usually, one has $n = m$

since the S-boxes are required to be bijective for decryption purposes. An S-box is defined by m *coordinates functions* $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$, which determine the i-th output bit of F for $1 \leq i \leq m$. The *component functions* of an S-box, additionally, are all the linear combinations of its coordinate functions (excluding the null combination). Similarly to Boolean functions in stream ciphers, the S-boxes composing the confusion layer in an SPN block cipher must also satisfy certain cryptographic properties to thwart attacks, among which we describe the most important ones below. The reader can refer to [6] for a deeper treatment of the cryptographic properties of vectorial Boolean functions.

- *Balancedness/Bijectivity*: Analogously to the single-output setting, an S-box $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ needs to be *balanced*, meaning that each output vector in $\mathbb{F}_2^m$ must appear the same number of times 2^{n-m}. When $n = m$ as in the case of S-box for SPN ciphers, balancedness corresponds to bijectivity.
- *Nonlinearity*: The nonlinearity of an S-box S is defined as the minimum nonlinearity among all its component functions, and it should be as high as possible to withstand *linear cryptanalysis* attacks.
- *Differential uniformity*: The differential uniformity corresponds to the minimum value that can be observed in the *difference distribution table* of the S-box, where each entry is indexed by a pair of vectors $a \in \mathbb{F}_2^n \backslash \{\underline{0}\}$, $b \in \mathbb{F}_2^m$ and it reports the number of times that the value of $F(x) \oplus F(x \oplus a)$ equals b. Ideally, the differential uniformity of an S-box should be as low as possible to avoid *differential cryptanalysis attacks*.

The unfeasibility of exhaustive search is even more prominent for S-boxes than in the case of Boolean functions. The number of (n, m)-functions is 2^{m2^n}, since each S-box is defined by the m truth tables of its coordinate functions. Considering only bijective (n, n)-functions, their number is approximately $2.6 \cdot 10^{35}$ already for $n = 5$ variables, which is not amenable to exhaustive enumeration. Hence, the use of heuristic optimization algorithms is even more motivated when searching for an S-box that is optimal with respect to the three properties above. Indeed, as in the Boolean functions' case, the designer can resort to algebraic constructions of good S-boxes, which, however, cover only a tiny fraction of all possible optimal S-boxes. Heuristic algorithms can provide a wider variety of S-boxes, that can also be optimal concerning additional properties such as those related to their implementation costs.

4.2 Survey of Related Works

The body of literature regarding the use of AI methods to construct cryptographically strong S-boxes can be roughly divided into two main approaches. The first one seeks to solve the optimization problem related to the cryptographic properties of S-boxes using nature-inspired optimization algorithms, which is the straightforward extension of the same approach applied to single-output Boolean functions discussed in Sect. 3.2. On the other hand, the second direction leverages on AI-based computational models to synthesize S-boxes with good properties.

Perhaps the best known example in this regard is the *cellular automata* (CA) model, which we introduced in Sect. 2.3.

We start with an overview of the first approach. The first work adopting evolutionary algorithms to optimize the cryptographic properties of S-boxes dates back to Millan et al. [43]. There, the authors designed a genetic algorithm to evolve S-boxes with high nonlinearity and low autocorrelation, a property that is closely related to differential uniformity. Burnett et al. [5] designed a heuristic method, mostly based on hill climbing, to generate S-boxes with the same structure of those used in the Mars block cipher, one of the AES finalists. Fuller et al. [19] proposed a multi-objective optimization approach for the heuristic construction of cryptographically strong S-boxes. In particular, they showed that power mappings (S-boxes whose polynomial representation is defined by a single monomial) could be evolved through iterated mutations to obtain solutions with the best possible trade-off of nonlinearity and autocorrelation. Clark et al. [13] employed a two-stage process inspired by that adopted for Boolean functions in [11] to search for highly nonlinear S-boxes, using simulated annealing and hill climbing to minimize a new objective function motivated by Parseval relation. Picek et al. [57] applied GP and Cartesian GP to the evolution of S-boxes, devising a method to adapt these two heuristics to the permutation encoding which limited the search space only to bijective (n, n)-functions.

Further, Picek et al. proposed a new fitness function in [47] to design S-boxes with evolutionary algorithms, experimentally assessing that it allows one to find highly nonlinear solutions more quickly than other objective functions already defined in the literature. Ivanov et al. [24] set forth an interesting *reverse* approach where one starts from an initial pool of optimal S-boxes in terms of nonlinearity and differential uniformity (obtained, for example, through algebraic constructions based on finite field inversion). Then, they apply a GA to tweak these S-boxes and generate new ones with slightly suboptimal properties. The motivation for this strategy is to generate a large set of S-boxes with similar cryptographic properties. The designer can then choose among them those that satisfy additional implementation requirements for a specific block cipher.

Let us now turn to the second approach based on cellular automata. Daemen et al. [15] were among the first to pioneer the use of CA for the design of S-boxes in block ciphers. There, the authors studied a simple local rule called χ, which flips a cell's state if and only if the two cells on its right present the pattern 10, and proved that the resulting S-box is bijective only when the size of the CA is odd. Moreover, they showed that rule χ induces CA-based S-boxes with good nonlinearity and diffusion properties. Interestingly, the only nonlinear component used in the design of Keccak [4], the cryptographic primitive that has been adopted as the SHA-3 standard for hash functions, is a CA of 5 cells defined by rule χ. Seredynsky et al. [61] investigated S-boxes defined by *second-order* CA, whose invertibility is granted by the fact that the next state of each cell is computed as the XOR of the result of the local rule evaluated on its neighborhood and its previous state. In particular, the authors analyzed the *avalanche properties* of the S-boxes defined by such CA equipped with local rules

of 5 and 7 variables. Szaban et al. [65] investigated the set of all 256 local rules of 3 variables. They selected those which resulted in the best nonlinearity and autocorrelation values by evolving a CA-based S-box of size 8×8 for a certain number of steps. Ghoshal et al. [20] considered 4×4 S-boxes with optimal nonlinearity and differential uniformity defined by multiple iterations of CA rules, and presented efficient *threshold implementations* for them that can be used for the design of lightweight SPN block ciphers.

We conclude this section by mentioning some works at the intersection of the two approaches discussed above, i.e., those based on the use of heuristic optimization algorithms to construct S-boxes defined by CA. Up to now, this last direction turned out to be the most successful one, since it can produce S-boxes on par with those yielded by algebraic constructions from the cryptographic point of view and also having good implementation properties. Picek et al. [55] used GP to evolve CA-based S-boxes with sizes ranging from 5×5 to 8×8, obtaining optimal solutions for nonlinearity and differential uniformity up to size 7×7. Picek et al. [56] further explored this optimization approach based on GP by considering the S-boxes defined by CA rules also from the implementation perspective. Under this experimental setting, they managed to evolve CA-based S-boxes with optimal nonlinearity and differential uniformity, and having hardware implementation costs to those of other state of the art S-boxes. Finally, Mariot et al. [39] performed a systematic theoretical investigation of the cryptographic properties of CA-based S-boxes and developed a reverse-engineering approach based on GP to find what is the shortest CA rule resulting in a specific S-box.

5 Pseudorandom Number Generators

Pseudorandom number generators (PRNG) are a crucial component for the security of basically any cryptographic scheme. In this section, we first cover the basic notions about PRNG and the statistical tests used to assess the quality of pseudorandom sequences. Next, we survey the main applications of AI methods to design PRNG.

5.1 Background

Random numbers and sequences are fundamental for several cryptographic tasks, such as the generation of keys, nonces, and masks for side-channel countermeasures. However, truly random numbers are difficult to come by since they can be generated only through physical phenomena such as radioactive decay, atmospheric noise, or reflection of photons in semi-transparent mirrors, which require specialized (and expensive) hardware generators. Consequently, the designers of cryptographic primitives mostly rely on *Pseudorandom Number Generators* (PRNG), which stretch a short initial random *seed* into an arbitrarily long pseudorandom sequence.

The term "pseudo" comes from the fact that a PRNG is a deterministic function; hence it always generates the same pseudorandom sequence if it starts from the same seed. In particular, achieving unconditional security under a distinguishing attack is not possible with a PRNG, since given enough time any pseudorandom sequence can be trivially distinguished from a true one. Thus, cryptographic PRNG are usually designed under the computational security setting. One of the possible approaches is to model the PRNG with Boolean functions, thus leveraging the cryptographic properties discussed in Sect. 3.1. Indeed, the keystream generator used in the Vernam-like stream cipher can be considered a PRNG and can thus be designed using the combiner model, for instance.

On the other hand, the most common way to assess the quality of a cryptographic PRNG is by applying a battery of *statistical tests* on a sample of its sequences. The rationale is that if the pseudorandom sequences do not pass one or more tests in the suite, then the PRNG that produced them should not be used in cryptographic applications. Examples of statistical tests for cryptographic PRNG include the DIEHARD suite [40] and the NIST suite [3].

5.2 Survey of Related Works

Like S-boxes, one can remark two common trends in the use of AI methods for the design of cryptographic PRNG. The first one is centered on the use of cellular automata (CA) to define the structure of a PRNG. The second direction employs evolutionary algorithms to directly optimize the structure of a PRNG, or to evolve a CA whose dynamics is then used to produce pseudorandom sequences.

Wolfram [67] was the first to propose a PRNG based on a cellular automaton for stream ciphers. Specifically, he suggested using a CA equipped with local rule 30, which is known to induce a chaotic behavior in the CA dynamics. The seed of the PRNG is represented by the initial configuration of the CA, while the *trace* of the central cell (i.e., the sequence of states assumed by the central cell through multiple iterations) is taken as the pseudorandom sequence produced by the CA. However, Meier et al. [42] proved that Wolfram's PRNG is very weak from a cryptographic standpoint since it is vulnerable to a known-plaintext attack that exploits the *quasi-linearity* of rule 30, which allows to rewrite it in an equivalent way where the initial seeds are not equiprobable. Martin [41] remarked that this weakness is linked to the fact that rule 30 is not first-order correlation immune when interpreted as a Boolean function. Thus, later works in this direction, such as Formenti et al. [18] and Leporati et al. [31], focused on the search of larger CA local rules with a good trade-off of nonlinearity and correlation immunity. In particular, they assessed the quality of the pseudorandom sequences produced by the selected CA rules, respectively, with the DIEHARD and NIST statistical test suites. More recently, Manzoni et al. [32] considered CA-based PRNG from the *asynchrony* perspective, where the cells do not update all at the same time. Interestingly, through the NIST suite, the authors remarked that the quality of the pseudorandom sequences produced by certain local rules improves by allowing a small degree of asynchrony.

Concerning the approach based on evolutionary algorithms, Koza [28] used genetic programming to evolve Boolean trees used as CA local rules in Wolfram's PRNG model. In particular, the fitness function measured the *entropy* of the sequence produced by a CA equipped with the local rule defined by a GP tree, and the optimization objective was to maximize it. Remarkably, the best local rule resulting from Koza's experiments turned out to be equivalent to rule 30 used by Wolfram. Sipper and Tomassini [63] proposed a *cellular programming approach* based on a *hybrid CA*, where each cell can update its state according to a different local rule. In particular, the rule map was evolved using a genetic algorithm variation, where the fitness function to maximize was again Koza's entropy. Castro et al. [8] applied GP to design PRNG not based on the cellular automata approach; instead, the trees evolved by GP directly represented computer programs to generate pseudorandom numbers. In order to drive the GP optimization process, the authors adopted a fitness function based on the *avalanche effect*, which measures how many bits change in the output of the generator when only a single bit is complemented in the seed. Martinez et al. [30] later refined this approach, by proposing the PRNG Lamar designed through GP, and assessed through the DIEHARD test suite. Picek et al. [58] investigated the construction of PRNG with cartesian GP, showing that the evolved generators passed all NIST randomness tests and were quite efficient and small to be implemented in hardware. Along a similar research line, Picek et al. [59] presented three different methods based on Cartesian GP to design PRNG for masking countermeasures in side-channel attacks, each one allowing for a different degree of reconfigurability in hardware implementations.

6 Conclusions and New Directions

As we have seen in the previous sections, the field of AI-based cryptography is characterized by an extensive literature. We remark that our overview is far from being exhaustive. As a matter of fact, in this chapter, we did not cover several other minor research threads, such as the construction of ciphers with co-evolutionary methods [52], the design of secret sharing schemes based on cellular automata [35], construction of quantum key distribution protocols with evolutionary algorithms [29], and AI-driven cryptanalysis [9,21]. However, we believe that the use cases of Boolean functions, S-boxes, and pseudorandom generators are representative of the main research trends in this area, both for the variety of methods adopted to address their design and also for the research challenges underlying them.

We conclude by discussing two new directions of research regarding the design of cryptographic primitives, where AI methods could be applied in the future.

Concerning the area of Boolean functions and S-boxes, most of the related literature focuses on applying a particular heuristic and then experimentally assessing its performances by verifying the cryptographic properties of the solutions produced by it. However, different heuristic techniques can have a very different degree of success in generating good Boolean functions and S-boxes. The

reasons underlying these performance gaps are still not very clear. It would thus be interesting to perform a meta-analysis of the main heuristic techniques used up to now (e.g., evolutionary algorithms and swarm intelligence methods), and investigate more closely why some approaches are more successful than others. Along the same direction, it would also be useful to study more in detail the difficulty of the combinatorial optimization problems related to Boolean functions and S-boxes, for example, by employing *fitness landscape analysis* techniques on the underlying search spaces. As far as the authors know, only Jakobovic et al. [25] performed an analysis of the fitness landscape associated with S-boxes, and there are still several avenues for further research that could be explored in this domain.

Joan Daemen suggested an alternative approach to the design of block ciphers where AI techniques could help, which does not focus on the cryptographic properties of S-boxes[1]. This approach starts from the observation that recent lightweight primitives inspired by the structure of Keccak [4] employ very simple and small S-boxes, whose optimality can be easily achieved without having to resort to heuristic algorithms. Examples in this class of primitives include the cryptographic permutation Xoodoo [16], which employs a 3×3 S-box defined by the same CA rule χ of Keccak. The design of Xoodoo depends on several real-valued parameters, such as *round constants*, used to break the symmetry produced by the shift-invariant structure of χ. An interesting direction of research in this setting would be to use heuristic techniques, such as evolutionary algorithms, to optimize the parameters of Xoodoo for different dimensions, for example, by considering more constrained hardware implementations.

References

1. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography by cellular automata or how fast can complexity emerge in nature? In: Proceedings of ICS 2010, pp. 1–19 (2010)
2. Bäck, T.: Evolutionary Algorithms in Theory and Practice - Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford (1996)
3. Bassham III, L.E., et al.: Sp 800–22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. National Institute of Standards & Technology (2010)
4. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference, January 2011. http://keccak.noekeon.org/
5. Burnett, L., Carter, G., Dawson, E., Millan, W.: Efficient methods for generating MARS-like S-boxes. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 300–313. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44706-7_21
6. Carlet, C.: Boolean Functions for Cryptography and Coding Theory. Cambridge University Press, Cambridge (2021)

[1] Personal communication.

7. Castro, J.C.H., Viñuela, P.I., del Arco-Calderón, C.L.: Finding efficient nonlinear functions by means of genetic programming. In: Palade, V., Howlett, R.J., Jain, L. (eds.) KES 2003. LNCS (LNAI), vol. 2773, pp. 1192–1198. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45224-9_161
8. Castro, J.C.H., Seznec, A., Isasi, P.: On the design of state-of-the-art pseudorandom number generators by means of genetic programming. In: IEEE Congress on Evolutionary Computation, pp. 1510–1516. IEEE (2004)
9. Castro, J.C.H., Viñuela, P.I.: New results on the genetic cryptanalysis of TEA and reduced-round versions of XTEA. New Gener. Comput. **23**(3), 233–243 (2005)
10. Chopard, B., Tomassini, M.: An Introduction to Metaheuristics for Optimization. Natural Computing Series. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-93073-2
11. Clark, J.A., Jacob, J.L.: Two-stage optimisation in the design of boolean functions. In: Dawson, E.P., Clark, A., Boyd, C. (eds.) ACISP 2000. LNCS, vol. 1841, pp. 242–254. Springer, Heidelberg (2000). https://doi.org/10.1007/10718964_20
12. Clark, J.A., Jacob, J.L., Maitra, S., Stanica, P.: Almost boolean functions: the design of boolean functions by spectral inversion. Comput. Intell. **20**(3), 450–462 (2004)
13. Clark, J.A., Jacob, J.L., Stepney, S.: The design of s-boxes by simulated annealing. New Gener. Comput. **23**(3), 219–231 (2005)
14. Clark, J.A., Jacob, J.L., Stepney, S., Maitra, S., Millan, W.: Evolving boolean functions satisfying multiple criteria. In: Menezes, A., Sarkar, P. (eds.) INDOCRYPT 2002. LNCS, vol. 2551, pp. 246–259. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36231-2_20
15. Daemen, J., Govaerts, R., Vandewalle, J.: Invertible shift-invariant transformations on binary arrays. Appl. Math. Comput. **62**(2), 259–277 (1994)
16. Daemen, J., Hoffert, S., Assche, G.V., Keer, R.V.: The design of Xoodoo and Xoofff. IACR Trans. Symmetric Cryptol. **2018**(4), 1–38 (2018)
17. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series, 2nd edn. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-44874-8
18. Formenti, E., Imai, K., Martin, B., Yunès, J.-B.: Advances on random sequence generation by uniform cellular automata. In: Calude, C.S., Freivalds, R., Kazuo, I. (eds.) Computing with New Resources. LNCS, vol. 8808, pp. 56–70. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13350-8_5
19. Fuller, J., Millan, W., Dawson, E.: Multi-objective optimisation of bijective S-boxes. In: Proceedings of CEC 2004, pp. 1525–1532 (2004)
20. Ghoshal, A., Sadhukhan, R., Patranabis, S., Datta, N., Picek, S., Mukhopadhyay, D.: Lightweight and side-channel secure 4×4 S-boxes from cellular automata rules. IACR Trans. Symmetric Cryptol. **2018**(3), 311–334 (2018)
21. Gohr, A.: Improving attacks on round-reduced speck32/64 using deep learning. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 150–179. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_6
22. Hrbacek, R., Dvorak, V.: Bent function synthesis by means of cartesian genetic programming. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 414–423. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_41
23. Ilachinski, A.: Cellular Automata: A Discrete Universe. World Scientific Publishing Co. Inc. (2001)

24. Ivanov, G., Nikolov, N., Nikova, S.: Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties. Cryptogr. Commun. **8**(2), 247–276 (2016). https://doi.org/10.1007/s12095-015-0170-5
25. Jakobovic, D., Picek, S., Martins, M.S.R., Wagner, M.: A characterisation of S-box fitness landscapes in cryptography. In: Auger, A., Stützle, T. (eds.) Proceedings of GECCO 2019, pp. 285–293. ACM (2019)
26. Kari, J.: Basic concepts of cellular automata. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 3–24. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-540-92910-9_1
27. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, 2nd edn. CRC Press, Boca Raton (2014)
28. Koza, J.R.: Evolving a computer program to generate random numbers using the genetic programming paradigm. In: ICGA, pp. 37–44. Morgan Kaufmann (1991)
29. Krawec, W., Picek, S., Jakobovic, D.: Evolutionary algorithms for the design of quantum protocols. In: Kaufmann, P., Castillo, P.A. (eds.) EvoApplications 2019. LNCS, vol. 11454, pp. 220–236. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16692-2_15
30. Lamenca-Martinez, C., Hernandez-Castro, J.C., Estevez-Tapiador, J.M., Ribagorda, A.: Lamar: a new pseudorandom number generator evolved by means of genetic programming. In: Runarsson, T.P., Beyer, H.-G., Burke, E., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 850–859. Springer, Heidelberg (2006). https://doi.org/10.1007/11844297_86
31. Leporati, A., Mariot, L.: Cryptographic properties of bipermutive cellular automata rules. J. Cell. Autom. **9**(5–6), 437–475 (2014)
32. Manzoni, L., Mariot, L.: Cellular automata pseudo-random number generators and their resistance to asynchrony. In: Mauri, G., El Yacoubi, S., Dennunzio, A., Nishinari, K., Manzoni, L. (eds.) ACRI 2018. LNCS, vol. 11115, pp. 428–437. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99813-8_39
33. Manzoni, L., Mariot, L., Tuba, E.: Balanced crossover operators in genetic algorithms. Swarm Evol. Comput. **54**, 100646 (2020)
34. Mariot, L., Jakobovic, D., Leporati, A., Picek, S.: Hyper-bent boolean functions and evolutionary algorithms. In: Sekanina, L., Hu, T., Lourenço, N., Richter, H., García-Sánchez, P. (eds.) EuroGP 2019. LNCS, vol. 11451, pp. 262–277. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16670-0_17
35. Mariot, L., Leporati, A.: Sharing secrets by computing preimages of bipermutive cellular automata. In: Was, J., Sirakoulis, G.C., Bandini, S. (eds.) ACRI 2014. LNCS, vol. 8751, pp. 417–426. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11520-7_43
36. Mariot, L., Leporati, A.: A genetic algorithm for evolving plateaued cryptographic boolean functions. In: Dediu, A.-H., Magdalena, L., Martín-Vide, C. (eds.) TPNC 2015. LNCS, vol. 9477, pp. 33–45. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26841-5_3
37. Mariot, L., Leporati, A.: Heuristic search by particle swarm optimization of boolean functions for cryptographic applications. In: Silva, S., Esparcia-Alcázar, A.I. (eds.) Companion Proceedings of GECCO 2015, pp. 1425–1426. ACM (2015)
38. Mariot, L., Picek, S., Jakobovic, D., Leporati, A.: Evolutionary search of binary orthogonal arrays. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.) PPSN 2018. LNCS, vol. 11101, pp. 121–133. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99253-2_10
39. Mariot, L., Picek, S., Leporati, A., Jakobovic, D.: Cellular automata based s-boxes. Cryptogr. Commun. **11**(1), 41–62 (2019)

40. Marsaglia, G.: The marsaglia random number CDROM including the diehard battery of tests of randomness (2008). http://www.stat.fsu.edu/pub/diehard/
41. Martin, B.: A Walsh exploration of elementary CA rules. J. Cell. Autom. **3**(2), 145–156 (2008)
42. Meier, W., Staffelbach, O.: Analysis of pseudo random sequences generated by cellular automata. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 186–199. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_17
43. Millan, W., Burnett, L., Carter, G., Clark, A., Dawson, E.: Evolutionary heuristics for finding cryptographically strong S-boxes. In: Varadharajan, V., Mu, Y. (eds.) ICICS 1999. LNCS, vol. 1726, pp. 263–274. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-540-47942-0_22
44. Millan, W., Clark, A., Dawson, E.: An effective genetic algorithm for finding highly nonlinear boolean functions. In: Han, Y., Okamoto, T., Qing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 149–158. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0028471
45. Millan, W., Clark, A., Dawson, E.: Heuristic design of cryptographically strong balanced Boolean functions. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 489–499. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054148
46. Picek, S., Carlet, C., Jakobovic, D., Miller, J.F., Batina, L.: Correlation immunity of boolean functions: an evolutionary algorithms perspective. In: Proceedings of GECCO 2015, pp. 1095–1102 (2015)
47. Picek, S., Cupic, M., Rotim, L.: A new cost function for evolution of S-boxes. Evol. Comput. **24**(4), 695–718 (2016)
48. Picek, S., Guilley, S., Carlet, C., Jakobovic, D., Miller, J.F.: Evolutionary approach for finding correlation immune boolean functions of order t with minimal hamming weight. In: Dediu, A.-H., Magdalena, L., Martín-Vide, C. (eds.) TPNC 2015. LNCS, vol. 9477, pp. 71–82. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26841-5_6
49. Picek, S., Jakobovic, D.: Evolving algebraic constructions for designing bent boolean functions. In: Friedrich, T., Neumann, F., Sutton, A.M. (eds.) Proceedings of GECCO 2016, pp. 781–788. ACM (2016)
50. Picek, S., Jakobovic, D., Golub, M.: Evolving cryptographically sound boolean functions. In: Blum, C., Alba, E. (eds.) Companion Proceedings of GECCO 2013, pp. 191–192. ACM (2013)
51. Picek, S., Jakobovic, D., Miller, J.F., Batina, L., Cupic, M.: Cryptographic boolean functions: one output, many design criteria. Appl. Soft Comput. **40**, 635–653 (2016)
52. Picek, S., Knezevic, K., Jakobovic, D., Derek, A.: C^3po: cipher construction with cartesian genetic programming. In: López-Ibáñez, M., Auger, A., Stützle, T. (eds.) Companion Proceedings of GECCO 2019, pp. 1625–1633. ACM (2019)
53. Picek, S., Knezevic, K., Mariot, L., Jakobovic, D., Leporati, A.: Evolving bent quaternary functions. In: 2018 IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, 8–13 July 2018, pp. 1–8. IEEE (2018)
54. Picek, S., Marchiori, E., Batina, L., Jakobovic, D.: Combining evolutionary computation and algebraic constructions to find cryptography-relevant boolean functions. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 822–831. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_81

55. Picek, S., Mariot, L., Leporati, A., Jakobovic, D.: Evolving S-boxes based on cellular automata with genetic programming. In: Bosman, P.A.N. (ed.) Companion Proceedings of GECCO 2017, pp. 251–252. ACM (2017)
56. Picek, S., Mariot, L., Yang, B., Jakobovic, D., Mentens, N.: Design of S-boxes defined with cellular automata rules. In: Proceedings of CF 2017, pp. 409–414. ACM (2017)
57. Picek, S., Miller, J.F., Jakobovic, D., Batina, L.: Cartesian genetic programming approach for generating substitution boxes of different sizes. In: Companion Proceedings of GECCO 2015, pp. 1457–1458 (2015)
58. Picek, S., Sisejkovic, D., Rozic, V., Yang, B., Jakobovic, D., Mentens, N.: Evolving cryptographic pseudorandom number generators. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 613–622. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_57
59. Picek, S., et al.: PRNGs for masking applications and their mapping to evolvable hardware. In: Lemke-Rust, K., Tunstall, M. (eds.) CARDIS 2016. LNCS, vol. 10146, pp. 209–227. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54669-8_13
60. Saber, Z., Uddin, M.F., Youssef, A.M.: On the existence of (9, 3, 5, 240) resilient functions. IEEE Trans. Inf. Theory **52**(5), 2269–2270 (2006)
61. Seredynski, F., Bouvry, P., Zomaya, A.Y.: Cellular automata computations and secret key cryptography. Parallel Comput. **30**(5–6), 753–766 (2004)
62. Shannon, C.E.: Communication theory of secrecy systems. Bell Syst. Tech. J. **28**(4), 656–715 (1949)
63. Sipper, M., Tomassini, M.: Generating parallel random number generators by cellular programming. Int. J. Mod. Phys. C **7**(02), 181–190 (1996)
64. Stinson, D.R., Paterson, M.: Cryptography: Theory and Practice. CRC Press, Boca Raton (2018)
65. Szaban, M., Seredynski, F.: Cryptographically strong S-boxes based on cellular automata. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 478–485. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79992-4_62
66. Tomassini, M., Perrenoud, M.: Cryptography with cellular automata. Appl. Soft Comput. **1**(2), 151–160 (2001)
67. Wolfram, S.: Cryptography with cellular automata. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 429–432. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_32