How to Use Metaheuristics for Design of Symmetric-Key Primitives[⋆]

Ivica Nikolić

National University of Singapore

Abstract. The ultimate goal of designing a symmetric-key cryptographic primitive often can be formulated as an optimization problem. So far, these problems mainly have been solved with trivial algorithms such as brute force or random search. We show that a more advanced and equally versatile class of search algorithms, called metaheuristics, can help to tackle optimization problems related to design of symmetric-key primitives. We use two nature-inspired metaheuristics, simulated annealing and genetic algorithm, to optimize in terms of security the components of two recent cryptographic designs, SKINNY and AES-round based constructions. The positive outputs of the optimization suggest that metaheuristics are non-trivial tools, well suited for automatic design of primitives.

Keywords: Metaheuristic, simulated annealing, genetic algorithm, automatic tool, cryptographic primitive

1 Introduction

In the past several years we have seen a major development of computer tools for automatic analysis of symmetric-key primitives. The tools cover a wide range of analysis techniques: differential [5, 6, 7, 16, 18, 19, 21, 30, 31, 32, 33, 38, 39, 40], linear [17, 30], impossible differential [12, 15, 26, 29, 36, 43], meet-in-the-middle [8, 14, 15], etc. Among other applications, the tools can serve as a proof of security of new designs because they can provide resistance of new designs against most (sometimes all) of the known cryptographic attacks.

Advanced computer tools for design of symmetric-key primitives, however, have not been considered. Instead, most of the design problems have been solved either analytically or with trivial computer algorithms such as brute force and random search. Consider, for example, the problem of tweaking AES to make it more resistant against meet-in-the-middle attacks. With automatic tools for analysis against meet-in-the-middle attacks we can check the security margin of each tweaked version of AES. If we tweak only `ShiftRows`, then we can brute force the space of all tweaks, check each tweak with the above automatic tools,

⋆ Python implementation of the results is available at https://github.com/nusnikolic/metaheuristics and find the one that provides the highest security. On the other hand, if we decide to tweak both ShiftRows and MixColumns, then the space of tweaks may be too large for a brute force, and thus we will use a random search. That is, we will check only a subset of randomly chosen tweaks and find the best among them. These two simple algorithms have been basically the only available computer tools to designers.

Assume the goal is to create a tool for automatic design of symmetric-key primitives that is: 1) based on more advanced search methods, and is 2) versatile, can tackle a variety of design problems. Note, brute force and random search do not satisfy the first point because they are trivial, but do satisfy the second point because they can be applied to many design problems. In a nutshell, these two algorithms are the simplest optimization methods. Therefore, to build a better design tool we need to focus on the next class of known optimization algorithms, that is also universal, but more sophisticated. That is the class of metaheuristics.

A metaheuristic is a search algorithm used to find a sufficiently good solution to an optimization problem. It makes almost no assumptions about the optimized (objective) function and it performs equally well when the function is not explicitly defined, but it can be queried. The search strategy implemented in a metaheuristic is often based on some nature-inspired method or technique – metaheuristics are named according to their nature equivalent, for instance, particle swarm optimization, simulated annealing, ant colony optimisation, evolutionary computation, etc. In cryptography, metaheuristics have been used mainly to design Sboxes that satisfy special criteria, such as resistance against cryptographic attacks [1,11,34,42,44].

Our Contributions. Arguably, the design decision behind any part of a symmetric-key cryptographic primitive is driven by the goal of optimization (in terms of security, size, throughput, etc.). Therefore, we regard the problem of design purely as an optimization problem. The computer algorithms that solve the optimization problem we call *tools for automatic design*.

Our tools are based on metaheuristics. These search algorithms are sufficiently universal to solve most of the design optimization problems. We use two nature-inspired metaheuristics: simulated annealing and genetic algorithm. We introduce the metaheuristics in Section 2; for each of them we point the main idea, provide a description in pseudo-code, and give a list of the most important parameters. In Section 3, we apply the metaheuristics to solve two concrete design optimization problems. To do so, first we identify the optimization problem, then formally defined it (describe the objective function and its input space), and finally we use metaheuristics to find good solutions. Our two problems are related to finding new components in the recently proposed block cipher SKINNY [3] and the AES-round based constructions [23]. We choose these two primitives because they best demonstrate the effectiveness of metaheuristics. Both SKINNY and the AES-round constructions are designed with clear optimization goals and, considering their excellent performance, achieve these goals. Nonetheless, metaheuristics allow even further optimization. We show that simulated annealing and genetic algorithm can be used to find specific components in the two primitives that results in even higher performance according to criteria which was considered important by the designers. More precisely, we use the metaheuristics to find for SKINNY a permutation in the tweakey schedule that leads to a higher resistance against related-tweakey attacks and for the AES-round constructions to find a round transformation that results in a better security against internal collisions.

To summarize, our main objective and contribution is to provide an empirical proof that, due to their simplicity and versatility, metaheuristics are perhaps the most effective tools for automatic design of symmetric-key primitives.

2 Metaheuristics

Consider a simple optimization problem: find optimum (maximum or minimum) of an objective function $f(x) : D \to R$. If $f(x)$ is given as a blackbox, i.e. it can be queried but is not explicitly defined, then mathematical and standard computer science methods for solving optimization problems cannot be applied because they require full definition of $f(x)$. In addition, if the domain D is discrete and has a large size, then the optimization problem cannot be solved by a brute force in practical time.

To cope with these type of problems, we use metaheuristics. They are approximate algorithms – the solution they provide is not guaranteed to be optimal (although some have a theoretical proof of asymptotic convergence). However, metaheuristics output the solution by using only limited computational resources, i.e. they are practical algorithms. Hence, among other applications, metaheuristics are well suited for search of near optimal solutions to optimization problems where the (blackbox) objective function is expensive.

There are various classifications of metaheuristics. According to the search strategy, they are divided into local search (try to find only the local optimum) and global search (global optimum). For instance, one of the most popular metaheuristic is the hill climbing method which tries to find only the local optimum. Another classification is single vs population-based search. A single metaheuristic works with only one candidate solution at a time, while population-based works simultaneously with multiple candidates. Hill climbing, simulated annealing, iterated local search are examples of single search, while genetic algorithm, ant colony optimization are examples of population-based search.

The efficiency of metaheuristics is tested experimentally by comparing the time complexities (measured in calls to $f(x)$) the metaheuristics require to solve some well-known problems. Depending on the problems, the comparative efficiency of two metaheuristics can vary, i.e. for some problems the first may be better, while for others the second. Therefore, the term "best metaheuristic" is meaningless. Testing the efficiency of a metaheuristic is not trivial because each is associated with a set of parameters. A metaheuristic needs a fine tuning of its parameters for each problem – this can be a very long and tedious process but it can have a major impact on its efficiency. For each metaheuristic, there are recommended set of values for its parameters, however, they were deduced empirically from its previous applications and thus provide no guarantee of optimality.

Further we will use two metaheuristics: simulated annealing and genetic algorithm. The choice is not accidental – both of them have been reported as one of the best performing on wide variety of problems in the single-based and population-based categories, respectively. In the sequel we give a minimal description of the two metaheuristics which we believe is sufficient to understand our ideas that follow. An interested reader can find more details about the metaheuristics for instance in [37, 41].

2.1 Simulated Annealing

Simulated annealing [9, 27] is a single-based, global search metaheuristic. It is a nature-inspired algorithm that mimics a physical process occurring in chemical substances: heating, followed by cooling and crystallizing.

Given an objective function $f(x)$, simulated annealing tries to find its maximum[1] by iteratively improving the potential solution. That is, starting from some random x_0, it builds $x_1, x_2, \ldots$. At iteration i, the value of x_i is produced from the previous value x_{i-1}, with the goal of maximizng further the function $f(x)$, i.e. $f(x_i) \geq f(x_{i-1})$. The main idea of simulated annealing is to allow probabilistic degradation of solutions, i.e. sometimes it accepts x_i even if $f(x_i) < f(x_{i-1})$. However, the probability of acceptance varies: in the early stages (when i is smaller) it accepts more degrading solutions, while later less. Such a strategy allows at the beginning to explore more variety of solutions, including degrading, while later to focus only on local optimization. Note, the degrading solutions allow the algorithm to escape local optima.

A formal description of simulated annealing is given in Algorithm 1. It takes as inputs three parameters: initial temperature T, cooling schedule function $\alpha(T)$, and neighbour function $\epsilon(x)$. In the initialization, it assigns a random value as a best solution x to the maximization problem of $f(x)$. Then it keeps trying to build better solution by iterating the same procedure: from x generate a new candidate x', and if it complies to a certain criteria, accept it as a new solution x. The function $\epsilon(x)$ generates x' from x by slightly changing[2] the value of x. If x' is a better solution than x, then x is updated to x'. However, if x' is worse, then it is not immediately rejected. Rather, it can be accepted, but only with some probability. The probability of acceptance (expressed with $r < e^{\frac{f(x')-f(x)}{T}}$, where $f(x') - f(x)$ is negative) is higher when the temperature T is higher and when the value of the objective function on the new candidate x' is closer to the value of the old candidate x. The iterations are stopped once the termination criteria is met. The criteria can be set differently: through the number of iterations, the value of the temperature, etc.

[1] Finding the minimum can be achieved similarly, with minor changes.

[2] For instance, when x is a vector, then $\epsilon(x)$ returns another vector in some predefined ϵ environment of x.

Algorithm 1 Simulated Annealing

```
Input: temperature $T_0$, cooling schedule $\alpha(T)$, neighbour function $\epsilon(x)$

$x \leftarrow \$$                                  ▷ Generate random initial value
$T \leftarrow T_0$
do
    $x' \leftarrow \epsilon(x)$                    ▷ Generate random neighbour
    if $f(x') > f(x)$ then                         ▷ If new maximum then accept it
        $x \leftarrow x'$
    else
        $r \leftarrow U[0,1]$                      ▷ Generate uniformly random number
        if $r < e^{\frac{f(x')-f(x)}{T}}$ then
            $x \leftarrow x'$                      ▷ Accept degrading solution
        end if
    end if
    $T \leftarrow \alpha(T)$                       ▷ Reduce the temperature
while (termination criteria not met)

Output: $x$
```

Parameters. As mentioned earlier, the main objective when choosing the values of the parameters is to optimize the efficiency of the metaheuristic so that it can produce a solution close to the global maximum in the shortest possible time. Simulated annealing requires the following parameters:

– **Neighbour function:** $\epsilon(x)$ should return x' that is in the neighbourhood of x, i.e. $\|x - \epsilon(x)\|$ should be small. For instance, if x is a vector then we can define $\epsilon(x)$ as a vector that coincides with x on all coordinates except one. Note, if $\|x - \epsilon(x)\|$ is large (or unlimited), then simulated annealing turns into a plain random search. Refer to Appendix B for more discussion on neighbour functions.

– **Cooling schedule:** $\alpha(T)$ should be monotonic (strictly decreasing) function. There are several choices for $\alpha(T)$: linear, exponential, inverse, logarithmic and other cooling schedules. We will use inverse cooling, defined as $\alpha(T) = \frac{T}{1+\beta T}$, where β is small constant, usually of order 0.001. We choose inverse cooling because it outperformed other cooling schedules in our preliminary experiments.

– **Initial temperature:** if T_0 is high then simulated annealing will explore more possibilities, however, it will require more time to converge to a near-optimal solution. Conversely, lower initial temperature leads to faster finding some solution that may not be so optimal. The value of T_0 should be chosen depending on the values of $\epsilon(x)$ and $\alpha(T)$ as well as the allowed time complexity, in order to balance the possibility of exploring more solutions with the maximal allowed time.

2.2 Genetic Algorithm

Genetic algorithm [22] is a population-based, global search metaheuristic. It belongs to the larger family of evolutionary algorithms which simulate natural selection to solve optimization problems.

Algorithm 2 Genetic Algorithm

```
Input: population size $N$, selection function $Selection(\{F_i\})$, crossover function $Crossover(P_A, P_B)$, mutation probability $MutationProbability$ and function $Mutate(\{C_i\})$

for $i$=1 to $N$ do
    $P_i \leftarrow \$$                                    ▷ Generate random parents
end for
do
    for $i$=1 to $N$ do
        $F_i \leftarrow f(Parent_i)$                       ▷ Compute fitness of parents
    end for
    for $i$=1 to $\frac{N}{2}$ do
        $(P_A, P_B) = Select(\{F\})$                       ▷ Select 2 parents
        $(C_{2i}, C_{2i+1}) \leftarrow Crossover(P_A, P_B)$ ▷ Produce 2 children
    end for
    for $i$=1 to $N$ do
        $r \leftarrow U[0,1]$                              ▷ Generate uniformly random number
        if $r < MutationProbability$ then
            $C_i \leftarrow Mutate(C_i)$                   ▷ Mutate child
        end if
    end for
    $\{P_i\} \leftarrow \{C_i\}$                           ▷ Update the generation
while (termination criteria not met)

Output: the best parent among $\{P_i\}$
```

To find maximum of an objective function (called a fitness function), genetic algorithm works in iterations (called generations). At each iteration it tries to improve a set of solutions, rather than a single solution. This set is called a population of individuals. To produce a new population from an old population, i.e. to change the generation, genetic algorithm uses two operations: mutation and crossover. A mutation is applied to one individual and it consists of slightly changing it. On the other hand, crossover is a synonym for reproduction. It takes two individuals (called parents) and produces two new individuals (called children)[3]. The choice of parents is controlled by so-called selection function which decides how to choose the parents. The selection function is biased towards

[3] Variations of crossover operators exist where two parents can produce only a single child or more than two children.

individuals with better fitness (higher value of fitness function). This is done to mimic the natural selection of parents – those with better qualities (genes) have higher chance of reproduction. A formal description of genetic algorithm is given in Algorithm 2.

Parameters. Genetic algorithm uses a wide range of parameters:

– **Population size** N: the number of individuals. The recommended value of N is in the range $[\log |D|, 2\log |D|]$, where $|D|$ is the size of the search space.
– **Selection function**: the most popular types of selections are roulette-wheel (individuals' probabilities of being selected as parents are proportional to their fitness functions), tournament (several individuals are first randomly selected and then in tournament-like fashion the winner is chosen according to his/her fitness value), rank (the individuals are sorted according to their fitness value, and their positions – called ranks– are used to determine their probability of selection), and stochastic (several individuals are simultaneously selected as parents according to their probability distributions). More detailed descriptions of the selection functions are given in Appendix B.
– **Crossover function**: it produces children that share similarities with the parents. For instance, if the two parents are given as vectors (the coordinates of these vectors are called genes), then the corresponding coordinates of the vectors of their children will have values either of the first or of the second parent[4]. Crossover function decides how the children inherit parents' genes. We will use a uniform crossover function, i.e. each gene of the children (each coordinate of the vector) has an equal probability to come from any of the two parents, and this probability is independent of the previous genes.
– **Mutation probability and function**: within one generation, the mutation is applied only to a small number of individuals defined by mutation probability. A recommended value for this probability is in the range [0.001, 0.01], i.e. only around 0.1-1% of the individuals are mutated. The mutation function defines how an individual is changed – it alters slightly the genes of an individual.
– **Elitism:** usually the best individuals within each generation are kept, that is, at the end of a generation, a certain percentage of the best parents progress to the next generation (are copied to children). This is called elitism (from elite). A recommended elitism is in the range [0.05-0.2], i.e. 5-20% of the parents with best fitness progress to the next generation.

3 Applications

Usually the objective of a new cryptographic primitive is to provide at least one better functionality than all known designs. This functionally can vary and may include better throughput, smaller footprint, higher security, etc. Regardless

[4] In some cases this is not possible, so some of the genes will be random.

of the chosen functionality, the goal of designers essentially can be seen as an optimization problem.

The optimization of cryptographic designs may or may not be solved with the use of metaheuristics. If the optimization problem is too general or the objective function is not clearly stated, then metaheuristics cannot solve the problem. For instance, trying to tweak somehow the round function of AES to maximize its resistance against impossible differential attacks does not formulate a good objective function. On the other hand, trying to tweak the `MixColumns` matrix by changing its coefficients, provides a clear objective function: the input to the function is some `MixColumns` matrix, and the output is the security level against impossible differential attacks[5]. Some optimization problems can be solved better (faster or with higher precision) with methods other than metaheuristics, such as heuristics or even brute force. For instance, trying to tweak the `ShiftRows` constants in AES to maximize its resistance against impossible differential attack can be solved simply by brute force as the number of all possible variants is small.

From the above discussion it follows that we can *use metaheuristics to design or improve symmetric-key primitives when:*

1. The optimization goal can be quantified (the objective function is clearly stated and can be computed on arbitrary inputs),
2. The search space is relatively large and cannot be covered by a brute force,
3. The solution not necessarily has to be globally/locally optimal (recall, metaheuristics may or may not return optimal solution in feasible time).

Further we give two examples of good optimization goals, that can be tackled with metaheuristics. They are related to improving the security margin[6] of SKINNY [3] and of the AES-round based constructions from [23]. These two primitives are ideal candidates for testing the effectiveness of metaheuristics because they are recent designs, have strong emphasis on optimization of components, and have clear optimization goals. Note, we have considered as well the use of metaheuristics to a few other recent designs, however, for various reasons we omit the details of their applications. For instance, the potential optimization of the functions Simpira v2 [20] and Haraka [28] can be solved with a brute force, therefore the optimization does not satisfy the above second requirement, and hence metaheuristics are not the first choice. On the other hand, optimizing component in the authenticated encryption scheme Deoxys [25] can be done with metaheuristics, however, the problem is too similar to the further analyzed problem of SKINNY, and thus we omit it.

[5] Assuming that one can compute the security level against impossible differential attacks with tweaked `MixColumns` matrix.

[6] However, we remind the reader that this is not necessarily the only use of metaheuristics – they can be applied to optimize designs with respect to throughput, size, etc.

3.1 SKINNY

SKINNY [3] is a family of block ciphers proposed at CRYPTO'16. Its goal is to be on par with NSA cipher SIMON [2] in hardware and software, while providing higher security. The ciphers are tweakable, i.e. besides a key and a plaintext, they have a third input called a tweak. The tweaking is based on a framework [24] that treats the key and the tweak universally, as a single input called tweakey. SKINNY ciphers have state sizes $n = 64$ or $n = 128$ bits, regarded as 4x4 matrices of nibbles. On the other hand, the tweakey sizes t are multiples of the state size n, and have three versions: $t = n, t = 2n, t = 3n$.

SKINNY are iterative substitution-permutation ciphers. In Figure 1 we give one round of the ciphers when $t = 3n$. A state round consists of five familiar transformations: `SubCells` is an Sbox layer, `AddConstants` xors constants, `AddRoundTweakey` xors the two top rows of each tweakey word to the two top rows of the state, `ShiftRows` rotates the nibbles of the state rows, and `MixColumns` multiplies the state columns by some matrix. In the tweakey schedule, the three tweakey words TK_1, TK_2, and TK_3 undergo two transformations: state-wise nibble permutation P_T which is the same for all the tweakeys, and nibble-wise linear transformations l_i.

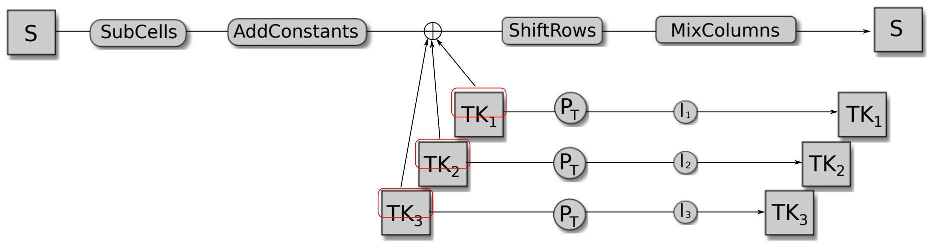

Fig. 1: One round of SKINNY with three tweakeys, $t = 3n$.

To be competitive in hardware and software, SKINNY ciphers have been highly optimized. Most of the transformations used in the ciphers have above average performance according to some design criteria and have been found as a result of some heuristic or a computer search. According to the extended version of the submission document [4], the nibble permutation P_T used in the tweakey schedule "has been chosen to maximize the bounds on the number of active Sboxes ... in the related-tweakey model". The search method used to find P_T is not specified.

With the use of metaheuristics, we will further optimize P_T. Note, the optimization problem has already been well formulated: find P_T to maximize the number of active Sboxes in the best related-tweakey characteristic. To find this number for a particular choice of P_T, as suggested by the designers we use an automatic tool based on integer linear programming (ILP). Therefore, ILP can be seen as the objective function f, which takes as input a permutation P_T and returns the number of active Sboxes. Hence our problem becomes

$$\max_{P_T} f(P_T),$$

where P_T is a permutation of 16 elements with an additional constraint: the first eight elements can be send only to the last eight positions, and vice versa. In fact, besides this constraint, the designers of SKINNY have imposed two additional: 1) P_T must consist of a single cycle, and 2) it sends the first 8 elements to the last 8 positions. In our search, we will relax these two constraints. This increases the search space from slightly under 8! possible choices of P_T to $(8!)^2$. Hence we will operate in a space that cannot be covered by a brute force and that has candidate permutations that may lead to ciphers with higher security margins. However, as we relax constraint 2), our permutations may require higher implementation cost in certain environments. Hence, our search for P_T should be seen in general as tradeoff between possibly higher security and lower speed.

Before we apply the metaheuristics, let us clarify a few points. First, SKINNY has several versions and we will focus on SKINNY-64-192 which is the 64-bit version with three tweakeys ($n = 64, t = 3n = 192$), i.e on the lightweight version which gives the most freedom to the attacker. Other versions can be processed similarly: moving from 64-bit to 128-bit will require more computational power[7], while reducing the number of tweakey words from three to two or one will require less power[8]. Second, the best characteristics not necessarily have to be found for the full cipher. Rather, once in a round-reduced characteristic the number of active Sboxes reaches some threshold, the cipher is already considered secure. In SKINNY-64-192 this number[9] is 33, and according to [4], for the original choice of P_T it is reached after 18 rounds. We will try to achieve 33 active Sboxes earlier, in 16 rounds[10]. Therefore, our objective function $f(P_T)$ is defined as the number of active Sboxes in the best characteristics on 16 rounds.

Let us clarify the above points. First note that the original permutation P_T^o of SKINNY is defined as

$$P_T^o = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 9 & 15 & 8 & 13 & 10 & 14 & 12 & 11 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix} \tag{1}$$

[7] The 128-bit version of SKINNY uses 8-bit Sboxes that have the same maximal differential propagation probability of 2^{-2} as the 4-bit Sboxes used in the 64-bit version. Therefore, to achieve 128-bit security (rather than 64-bit security) the number of active Sboxes in the best characteristic has to be much larger, which in return results in higher complexity search.

[8] For results on these versions refer to Appendix A.

[9] The number is defined by the state size and the probability of the best differential transition of the Sbox. The state of SKINNY is 64 bits, and the highest probability of a differential transition in the 4-bit Sbox is 2^{-2}, thus if the number of active Sboxes is $1 + \lfloor \frac{64}{2} \rfloor = 33$, the cipher is resistant against related-tweakey differential attacks.

[10] The number of rounds cannot be predicted a priori. We focus on 16 rounds, but if we do not succeed we can always compare either how many active Sboxes we have reached on 16 rounds, or if we have reached 33 active Sboxes on 17 rounds.

According to the designers, and confirmed with our own ILP tool, $f(P_T^o) = 27$. We are looking for another permutation P_T

$$P_T = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \end{pmatrix},$$

such that $f(P_T)$ is as large as possible. Note, there is an *additional* condition, which requires that $a_i \geq 8$ for $i = 0, 1, \ldots, 7$ and $a_i < 8$ for $i = 8, 9, \ldots, 15$, which assures that the first 8 elements are sent to the last 8 positions, and vice versa.

Let us focus on simulated annealing. To solve the optimization problem with this metaheuristic, we first need to specify the three parameters: $\epsilon(P_T), \alpha(T), T_0$. As a neighbour function $\epsilon(P_T)$ we use a random transposition. That is, we randomly choose two indices in P_T and switch the value of the elements with such indices. Note, however, the choice of indices cannot be completely random because $\epsilon(P_T)$ must fulfil the additional condition. Hence, to properly implement $\epsilon(P_T)$, we first choose the half of P_T where the transposition will occur, and only then the two random elements that belong to the same half. For cooling schedule $\alpha(T)$, as mentioned before, we use inverse cooling $\alpha(T) = \frac{T}{1+\beta T}$ and experiment with value of β in the range $[0.001, 0.003]$. Finally, as an initial temperature T_0 we take values in the range $[1, 2]$. Our termination criteria is time-based, i.e. we stop the search and output the best found solution after running the metaheuristics around a day on an 8-core processor.

Further, let us focus on genetic algorithms and the used parameters. In all of our implementations, the population size is 50. To test the effectiveness and impact of different selection functions, we used all four of them. In addition, we use mutation rate of 0.01 and a mutation function that closely resembles $\epsilon(P_T)$ from simulated annealing (i.e. mutation consists of one random transposition). Finally, we use elitism with 20% rate. The termination criteria is similar to that of simulated annealing, but we allow more time.

Table 1: Examples of permutations P_T found with simulated annealing (second row) and genetic algorithm (third row) and the resulting security level against related-key tweakey attacks when used in SKINNY-64-192. In the second column are the specifications of the permutations. In the third column is their security, i.e. the number of active Sboxes in the round-reduced characteristics according to the number of rounds. Highlighted numbers correspond to the lowest number of active Sboxes that already match the threshold for related-tweakey security.

Method	P_T	**Rounds**				
		14	15	16	17	18
Original	9 15 8 13 10 14 12 11 0 1 2 3 4 5 6 7	19	24	27	31	35
Simulated annealing	11 9 14 8 12 10 15 13 2 0 3 6 7 5 1 4	24	28	33	36	39
Genetic algorithm	14 11 8 9 15 13 10 12 1 2 0 7 5 4 3 6	24	28	33	36	39

The results of the optimization with the two metaheuristics are as follows. Both simulated annealing and genetic algorithm were able to find permutations P_T such that $f(P_T) = 33$. Simulated annealing performed similarly on different choices of parameters β and T_0, i.e. we did not detect any significant difference. On average, it required around 1000 calls to the objective function $f(x)$ to find a permutation P_T such that $f(P_T) = 33$. On the other hand, genetic algorithm performed better for some choices of selection functions. To find P_T such that $f(P_T) = 33$ on average over three trials, with stochastic selection it required 950 calls, with rank selection 1380 calls[11], with roulette-wheel 2250 calls, and with tournament selection 5900 calls. Therefore, we can conclude that simulated annealing and genetic algorithm with stochastic or rank selection performed similarly.

In Table 1 are shown examples of permutations P_T^{SA}, P_T^{GA} found with simulated annealing and genetic algorithm such that $f(P_T^{SA}) = f(P_T^{GA}) = 33$. For performance measurements, we give in the table as well the number of active Sboxes of the best characteristics reduced to not only 16 rounds, but in the range of 14 to 18 rounds. Evidently, the two new permutations result in higher numbers of active Sboxes in comparison to the original permutation of SKINNY.

We conclude this subsection with a discussion on further use of metaheuristics in SKINNY. One potential direction would be to optimize the resistance against related-tweakey attacks with respect to both P_T and `AddRoundTweakey`, i.e. by changing the permutation P_T and by identifying which 8 nibbles of the tweakey words should be xored to the state (instead of the 8 nibbles of the first two rows).

3.2 AES-round Based Constructions [23]

Software optimized designs based on the AES round function are presented in [23]. The main objective of the authors of this paper is to provide symmetric-key constructions (as building blocks of MACs and authenticated encryption schemes) that are efficient on the latest Intel processors[12]. The authors show seven constructions that run at only a few tenths of a cycle per byte on Intel processors Ivy Bridge, Haswell and Skylake.

The proposed constructions have a state composed of s words of 128 bits. The state is transformed by a round function given in Figure 2, where A stands for one AES round. Besides A, the only remaining operation is the xor (of message words M_{i_j} and state words X_{i_j}). Each construction is characterized by a parameter called a rate ρ which is defined as the number of AES rounds required to process a 128-bit message word. That is, ρ is the ratio of the number of calls to A to the number of different message words (in one round). The lower the rate, the faster

[11] This number (1380) not necessarily has to be divisible by the population size (50). The reason is two-fold: 1) we halt the search once a sufficiently good construction is found, without updating the whole population, and 2) we use elitism, which dictates that at each generation only $50 \cdot (1 - \mathit{elitism})$ individuals are updated.

[12] These processor have special instructions set called AES-NI, that can execute AES round function as a single instruction.

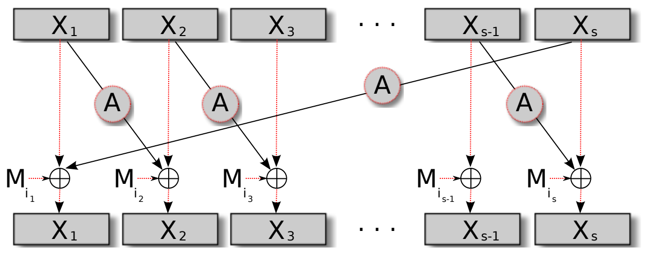

Fig. 2: The round function of the constructions from [23]. The transformations in red and dashed are optional.

the design, hence the goal of the authors has been to reduce the rate as much as possible.

A construction is considered secure if it is free of so-called internal collisions, which are special type of differential characteristics: they start and end in zero differences in the state[13]. A construction should provide 128-bit security, that is, differential characteristics that lead to internal collisions must not have a probability higher than 2^{-128}. To find the best characteristic and its probability, the authors reduce the problem to counting active Sboxes and use the aforementioned integer linear programming tool to get the lower bound on this number. The security level of 128 bits corresponds to at least 22 active Sboxes[14] in the best characteristics.

The seven proposed constructions have different number of state words (7 to 12) and different rates (in the range $[2, 3]$). For a particular choice of state size and rate, the authors use some heuristic (which has not been explained in the paper) to search the space of all constructions as defined in Figure 2 and consider only those which are resistant against internal collisions, i.e. their best characteristics have at least 22 active Sboxes. Constructions that have the lowest probability of internal collisions (i.e. the highest number of active Sboxes) are considered the best.

Further we use metaheuristics to optimize the constructions according to the design criteria of [23]. The optimization problem is clear: for a particular choice of state size s and rate ρ, *find a round function* as in Figure 2 that defines a construction whose best differential characteristic that leads to internal collisions *has maximal number of active Sboxes*. Once again, the role of objective function f is played by the integer linear program that returns a lower bound on the number of active Sboxes. To understand what the input to the objective function is, let us focus on Figure 2. Note, there are three types of red (optional) transformations in the round function. First, each of the s calls to A are optional. Therefore, we can use s-bit vector *aes_masks* to describe a particular configuration of the calls to A, where i-th bit of *aes_masks* is set iff in the round function, A is applied to

[13] The difference is introduced and later cancelled through the message words.

[14] Because the differential propagation probability of an Sbox in AES is 2^{-6}, thus 128-bit security means $\lfloor \frac{128}{6} \rfloor + 1 = 22$ active Sboxes.

the X_i. Second, all s feedforwards of words (the red vertical lines) can also be described with an s-bit vector $feed\_masks$. Finally, the xors of message words M_{i_j} can be described with vector $messages$ of s coordinates, each an integer value in the range $[0, w]$, where w is the total number of message words in a round. A value of 0 denotes that no message words is xored, while any positive integer value corresponds to the index of the message word being xored. As a result, each potential construction can be described with the three vectors: aes_masks, $feed\_masks$, and $messages$. However, note that not all combinations are possible because the values of aes_masks and $messages$ cannot be arbitrary. Rather, they must agree with the rate ρ. For instance, if $\rho = 2$ and the Hamming weight of aes_masks is 6, then the vector $messages$ can contain the values $1, 2,$ and 3, and it must have each of these values at least once. This assures that the rate of the constructions is indeed 2. Let us assume further that the tuples $(aes\_masks, feed\_masks, messages)$ agree with the predefined rate ρ. Then our optimization problem for fixed state size s and rate ρ can be defined as:

$$\max_{aes\_masks, feed\_masks, messages} f(aes\_masks, feed\_masks, messages)$$

We optimize six of the seven constructions proposed in [23]. We omit one, with rate $\rho = 2$ and size $s = 12$, because it has too expensive objective function – it took us half a day to compute f on an input.

To solve the optimization problems, we run simulated annealing and genetic algorithm with the following parameters. In simulated annealing, the neighbour function $\epsilon(x)$ consists of flipping 1-2 bits in some (or all) of the three vectors $aes\_masks, feed\_masks, messages$ (with an additional check on the rate ρ). Furthermore, we use inverse cooling $\alpha(T)$ with $\beta = 0.003$, and initial temperature $T_0 = 1.5$. In genetic algorithm, the population size is 30, combined with stochastic selection function, uniform crossover, mutation rate of 0.01, a mutation function based on random flip of bits, and 20% elitism. The termination criteria for both of the metaheuristics is based on number of calls to the objective function, and it is either 500 calls (for smaller search spaces) or 700 calls (for larger).

The outputs of the metaheuristics are given in Table 4. For all six constructions, both of the metaheuristics were able to find better candidates with an in crease of 13%-44% to the number of active Sboxes in comparison to the original proposals from [23]. Simulated annealing performed slightly better than genetic algorithm – in limited number of calls to the objective function, it managed to find constructions with higher security margin. We suspect this is due to the termination criteria as genetic algorithm requires more generations to find better solutions.

Finally, we note that we have also run metaheuristics to find competing constructions to the published ones [23] not only in terms of higher security, but in terms of efficiency too. We refer the reader to Appendix C for more details.

Table 2: AES-round based constructions. SA and GA stand for simulated annealing and genetic algorithm.

Method	**State size**	**Rate**	*aes_mask*	*feed_mask*	*messages*	**Active Sboxes**
[23]	6	3	111111	100100	011022	22
GA	6	3	111111	100100	122020	26
SA	6	3	111111	011100	102110	27
[23]	7	3	1111110	1101101	1012020	25
GA	7	3	0111111	0100110	0101201	35
SA	7	3	1110111	1111100	0100211	36
[23]	7	2.5	1101110	0100001	1111222	22
GA	7	2.5	1111001	0010110	1201102	25
SA	7	2.5	1011110	0110100	1021102	26
[23]	8	3	11101110	11011101	10102020	34
GA	8	3	11110011	00110000	10102022	42
SA	8	3	00111111	01001000	20221220	45
[23]	8	2.5	11011100	01000011	11112222	23
SA	8	2.5	10011011	10000110	02012021	30
GA	8	2.5	11111000	01100100	11022102	30
[23]	9	3	111111111	100100100	011022033	25
GA	9	3	111111111	111101111	012133031	34
SA	9	3	111111111	100100111	010321121	34

4 Conclusion

Metaheuristics are widely used algorithms for search of solutions to optimization problems. The design of symmetric-key primitives can be seen as one such problem, thus metaheuristics can be used to find better designs. Therefore, metaheuristics can serve as tools for automatic designs of symmetric-key primitives. Unlike brute force and random search, metaheuristics are non-trivial tools which should be scrutinized in absence of better heuristics or of other more advanced search methods.

We used two metaheuristics, simulated annealing and genetic algorithm, to optimize designs with respect to security. Our choice of metaheuristics was guided by their popularity and reported success – both of them are considered among the best performing on well known problems. On the other hand, as an optimization parameter we chose security because that led to well defined and computable objective functions[15]. We wrote the implementations of the two metaheuristics on C – they were straightforward to code. It took us several thousand CPU hours to test for good set of parameters and to find approximate solutions for the design optimization problems in SKINNY and the AES-round constructions. The outputs were positive – the metaheuristics were able to find better components for both of the primitives, sometimes improving the optimized component by more than 40%. Thus we can conclude that metaheuristics can serve as effective tools for automatic design of symmetric-key primitives.

Future research may focus on expanding the area of application and variety of metaheuristics. This includes formulating other design problems as optimization problems and subsequently using the proposed metaheuristics for their solution. We stress out that the optimization problems not necessarily have to be related to an increase in security, but may target better throughput, smaller size, etc. Furthermore, using metaheuristics other than simulated annealing and genetic algorithms may also improve design methods of crypto primitives. Some more advanced metaheuristics, such as the multi-objective genetic algorithm NSGA-II [13], may well excel in solving design problems related to multidimensional optimization, i.e. optimization by several criteria.

Acknowledgments

The author would like to thank the anonymous reviewers of ASIACRYPT'17 for their constructive comments and Yu Sasaki for helping to finalize the paper. This work is supported by the Ministry of Education, Singapore under Grant No. R-252-000-560-112.

References

1. M. Ahmad, D. Bhatia, and Y. Hassan. A novel ant colony optimization based scheme for substitution box design. *Procedia Computer Science*, 57:572–580, 2015.
2. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.
3. C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Robshaw and Katz [35], pages 123–153.
4. C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. Cryptology ePrint Archive, Report 2016/660, 2016. http://eprint.iacr.org/2016/660.
5. A. Biryukov and I. Nikolić. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, Camellia, Khazad and

[15] The objective function is well defined because the security criteria is characterized by a single parameter. On the other hand, it is computable, because there are various tools such as those based on ILP that can produce the output for an arbitrary input.

others. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 322–344. Springer, 2010.

6. A. Biryukov and I. Nikolić. Search for related-key differential characteristics in DES-like ciphers. In A. Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2011.
7. A. Biryukov and V. Velichkov. Automatic search for differential trails in ARX ciphers. In J. Benaloh, editor, *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, volume 8366 of *Lecture Notes in Computer Science*, pages 227–250. Springer, 2014.
8. C. Bouillaguet, P. Derbez, and P. Fouque. Automatic search of attacks on round-reduced AES and applications. *IACR Cryptology ePrint Archive*, 2012:69, 2012.
9. V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.
10. J. H. Cheon and T. Takagi, editors. *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, 2016.
11. J. A. Clark, J. L. Jacob, and S. Stepney. The design of S-boxes by simulated annealing. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1533–1537. IEEE, 2004.
12. T. Cui, K. Jia, K. Fu, S. Chen, and M. Wang. New automatic search tool for impossible differentials and zero-correlation linear approximations. *IACR Cryptology ePrint Archive*, 2016:689, 2016.
13. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation*, 6(2):182–197, 2002.
14. P. Derbez and P. Fouque. Exhausting Demirci-Selçuk meet-in-the-middle attacks against reduced-round AES. In S. Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 541–560. Springer, 2013.
15. P. Derbez and P. Fouque. Automatic search of meet-in-the-middle and impossible differential attacks. In Robshaw and Katz [35], pages 157–184.
16. D. Dinu, L. Perrin, A. Udovenko, V. Velichkov, J. Großschädl, and A. Biryukov. Design strategies for ARX with provable bounds: Sparx and LAX. In Cheon and Takagi [10], pages 484–513.
17. C. Dobraunig, M. Eichlseder, and F. Mendel. Heuristic tool for linear cryptanalysis with applications to CAESAR candidates. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 490–509. Springer, 2015.
18. S. Emami, S. Ling, I. Nikolić, J. Pieprzyk, and H. Wang. The resistance of PRESENT-80 against related-key differential attacks. *Cryptography and Communications*, 6(3):171–187, 2014.

19. P. Fouque, J. Jean, and T. Peyrin. Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 183–203. Springer, 2013.
20. S. Gueron and N. Mouha. Simpira v2: A family of efficient permutations using the AES round function. In Cheon and Takagi [10], pages 95–125.
21. D. Grault, P. Lafourcade, M. Minier, and C. Solnon. Revisiting AES related-key differential attacks with constraint programming. Cryptology ePrint Archive, Report 2017/139, 2017. http://eprint.iacr.org/2017/139.
22. J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
23. J. Jean and I. Nikolić. Efficient design strategies based on the AES round function. In T. Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2016.
24. J. Jean, I. Nikolić, and T. Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2014.
25. J. Jean, I. Nikolić, T. Peyrin, and Y. Seurin. Deoxys v1.4. Submitted to CAESAR, 2016.
26. J. Kim, S. Hong, J. Sung, C. Lee, and S. Lee. Impossible differential cryptanalysis for block cipher structures. In T. Johansson and S. Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings*, volume 2904 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 2003.
27. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
28. S. Kölbl, M. M. Lauridsen, F. Mendel, and C. Rechberger. Haraka v2 - efficient short-input hashing for post-quantum applications. *IACR Trans. Symmetric Cryptol.*, 2016(2):1–29, 2016.
29. Y. Luo, X. Lai, Z. Wu, and G. Gong. A unified method for finding impossible differentials of block cipher structures. *Information Sciences*, 263:211–220, 2014.
30. M. Matsui. On correlation between the order of S-boxes and the strength of DES. In A. D. Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.
31. S. Moriai, M. Sugita, K. Aoki, and M. Kanda. Security of E2 against truncated differential cryptanalysis. In H. M. Heys and C. M. Adams, editors, *Selected Areas in Cryptography, 6th Annual International Workshop, SAC'99, Kingston, Ontario, Canada, August 9-10, 1999, Proceedings*, volume 1758 of *Lecture Notes in Computer Science*, pages 106–117. Springer, 1999.
32. N. Mouha, Q. Wang, D. Gu, and B. Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *International Conference on Information Security and Cryptology*, pages 57–76. Springer, 2011.

33. I. Nikolić. Tweaking AES. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 198–210. Springer, 2010.

34. S. Picek, B. Yang, V. Rozic, and N. Mentens. On the construction of hardware-friendly 4x4 and 5x5 S-boxes. *Lecture Notes in Computer Science*, 2016.

35. M. Robshaw and J. Katz, editors. *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*. Springer, 2016.

36. Y. Sasaki and Y. Todo. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 185–215, 2017.

37. D. Simon. *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.

38. S. Sun, D. Gerault, P. Lafourcade, Q. Yang, Y. Todo, K. Qiao, and L. Hu. Analysis of AES, SKINNY, and others with constraint programming. *IACR Trans. Symmetric Cryptol.*, 2017(1):281–306, 2017.

39. S. Sun, L. Hu, K. Qiao, X. Ma, J. Shan, and L. Song. Improvement on the method for automatic differential analysis and its application to two lightweight block ciphers DESL and LBlock-s. In K. Tanaka and Y. Suga, editors, *Advances in Information and Computer Security - 10th International Workshop on Security, IWSEC 2015, Nara, Japan, August 26-28, 2015, Proceedings*, volume 9241 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2015.

40. S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014.

41. E.-G. Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

42. P. Tesar. A new method for generating high non-linearity S-boxes. *Radioengineering*, 2010.

43. S. Wu and M. Wang. Automatic search of truncated impossible differentials for word-oriented block ciphers. In S. D. Galbraith and M. Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 283–302. Springer, 2012.

44. M. Yang, Z. Wang, Q. Meng, and L. Han. Evolutionary design of S-box with cryptographic properties. In *Parallel and Distributed Processing with Applications Workshops (ISPAW), 2011 Ninth IEEE International Symposium on*, pages 12–15. IEEE, 2011.

A Applications to SKINNY-64-64 and SKINNY-64-128

In addition to the full search given in Section 3.1 on SKINNY-64-192 (i.e. in TK_3), we have also run search for P_T in SKINNY-64-64 and SKINNY-64-128, i.e. in TK_1 and in TK_2. The search criterion for P_T was identical as in Section 3.1. With the use of simulated annealing only, we have looked for P_T in three different related-tweakey differential models:

1. **Find P_T for SKINNY-64-64 secure in TK_1.** The search returned several permutations, each resulting in a cipher that has at least 33 active Sboxes in any 11-round related-tweakey differential characteristics.
2. **Find P_T for SKINNY-64-128 secure in TK_2.** Similarly, we found several permutations with at least 34 active Sboxes in any 14-round characteristic.
3. **Find P_T simultaneously for SKINNY-64-64 secure in TK_1 and for for SKINNY-64-128 secure in TK_2.** We found a few permutations P_T that simultaneously provide security in both TK_1 and TK_2. Interestingly, the corresponding characteristics have 33 active Sboxes on 11 rounds in TK_1 and 34 active Sboxes on 14 rounds in TK_2. In other words, any of these permutations can be used as an optimal candidate in scenarios 1) and 2).

Examples of permutations found with the search are given in Table 3.

Table 3: Examples of SKINNY permutations P_T found with simulated annealing that result in TK_1 and TK_2 secure ciphers.

Source	Permutation	Target	Sboxes	Rounds
Original [3]	9 15 8 13 10 14 12 11 0 1 2 3 4 5 6 7	TK_1	32	11
		TK_2	31	14
Model 1)	11 10 15 14 8 9 13 12 6 0 5 1 7 3 2 4	TK_1	33	11
Model 2)	9 8 11 13 14 12 10 15 7 6 4 2 0 5 3 1	TK_2	34	14
Model 3)	12 11 9 8 14 10 13 15 6 2 3 0 7 4 1 5	TK_1	33	11
		TK_2	34	14

B Specification and Implementation Details of the Metaheuristics

Selection functions. We work with four types of selection functions:

– **Roulette-wheel selection** is also called furness-proportionate selection. The selection probability of each individual is proportional to its fitness value. In Figure 3, we assume the population is composed of four individuals with

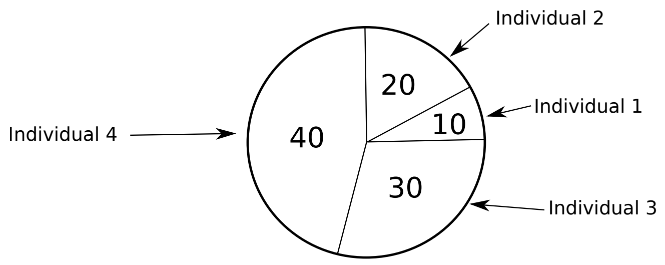

Fig. 3: Roulette-wheel selection with 4 individuals.

fitness measures of 40,30,20,10. Then, the roulette-wheel selection dictates that the first individual has $\frac{40}{100}$ probability of being selected as a parent, the second $\frac{30}{100}$, etc. To select a single parent, we "run the roulette", i.e. uniformly at random choose a number in the range $[0, 100)$, and accordingly choose that individual on which slice the "ball" has landed, e.g. if the number is anywhere in the range $[0, 10)$ then it is the individual 1, if in the range $[10, 30)$ then it is individual 2, etc.

- **Stochastic selection** is similar to roulette-wheel, but it increases the chance of high fitness individuals becoming parents. In stochastic selection the parents are selected in bulk. For example, in Figure 4 we show how to use the wheel to select four parents at once. We run the roulette once, i.e. select a random

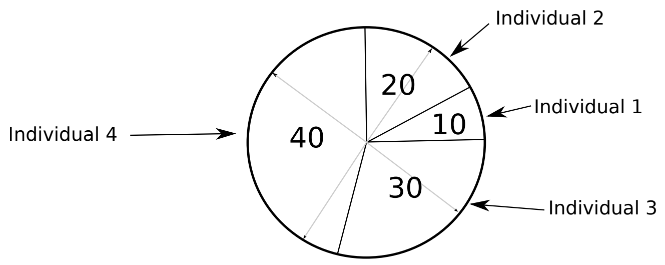

Fig. 4: Stochastic selection with 4 individuals. A spinner (in gray) with for evenly-spaced pointers is spun once to obtain the four parents.

number in range $[0, 100]$ and, as in roulette-wheel, choose the corresponding individual as a first parent. Then, the remaining three parents are the ones the correspond to the other three uniformly-spaced numbers. That is, if the ball has landed on 23, then we assume that it has also landed on $23 + \frac{100}{4} = 48$, $23 + 2 \cdot \frac{100}{4} = 73$, and $23 + 3 \cdot \frac{100}{4} = 98$.

- **Rank selection** is as well similar to roulette-wheel. However, instead of using the fitness to determine the portion of the wheel, individuals' rank is used. That is, all individuals within a population are sorted according to their fitness in ascending order, and their position is taken as a fitness measure in roulette-wheel fashion. For instance, individuals with fitnesses of 1,5,20,8, after sorting will be at positions 1,2,4,3, thus have $\frac{1}{10}, \frac{2}{10}, \frac{4}{10}, \frac{3}{10}$ probabilities to be selected as parents.

– **Tournament selection** depends on the tournament size and we use the most common size of 2. That is, to select a parent, we uniformly at random choose two individuals, compare their fitness, and choose the one that has higher fitness.

Pseudo code of a full selection procedure (based on roulette-wheel) used by the genetic algorithm for search of P_T in SKINNY is given below.

Algorithm 3 Roulette-wheel selection of parents used in the search for SKINNY

```
Input: Population of $N$ individuals given as pairs $(P_i, F_i)$, where $P_i$ is an individual
(permutation), and $F_i$ is its corresponding fitness

$total\_fitness \leftarrow \sum_1^N F_i$                       ▷ Sum up all fitnesses
$left\_range \leftarrow 0$                                    ▷ Create the roulette-wheel
for $i$=1 to $N$ do                                           ▷ Compute the slices for each ind.
    $range[i]\_left = left\_range$                            ▷ Left range
    $range[i]\_right = left\_range + \frac{F_i}{total\_fitness}$   ▷ Right range
    $left\_range = range[i]\_right$
end for

$Parents \leftarrow \emptyset$
for $i$=1 to $N/2$ do                                         ▷ Select $\frac{N}{2}$ pairs of parents
    $C_1 \leftarrow rand()$                                   ▷ Spin the ball for the first parent
    $C_2 \leftarrow rand()$                                   ▷ Spin the ball for the second parent
    for $j$=1 to $N$ do                                       ▷ Find the corresponding individuals
        if $range[j]\_left \leq C_1 < range[j]\_right$ then
            $parent_1 \leftarrow j$
        end if
        if $range[j]\_left \leq C_2 < range[j]\_right$ then
            $parent_2 \leftarrow j$
        end if
    end for
    $Parents = Parents \cup (P_{parent_1}, P_{parent_2})$
end for

Output: $Parents$
```

Neighbour functions $\epsilon(x)$. Intuitively, the task of a neighbour function is to produce a value in the neighbourhood of x. Thus they output values (or vectors) that are very similar to the input.

In SKINNY we define $\epsilon(P_T)$ as a swap of two elements. That is, we randomly choose two positions, and then exchange the elements in these two positions.

Since there is an additional requirement on the form of P_T, the swap can occur only between two elements that belong both to the same half. Further we give a simple pseudo-code that accomplishes this.

Algorithm 4 Neighbour function ϵ for the search in SKINNY

```
Input: Permutation $P_T$

$half \leftarrow randInt()\%2$                         ▷ Randomly choose a half
$i \leftarrow randInt()\%8$                            ▷ Randomly choose the first index
$j \leftarrow randInt()\%8$                            ▷ Randomly choose the second index
$P'_T \leftarrow P_T$                                  ▷ Assign the permutation
$P'_T[8 \cdot half + i] = P_T[8 \cdot half + j]$       ▷ Swap
$P'_T[8 \cdot half + j] = P_T[8 \cdot half + i]$       ▷ Swap

Output: $P'_T$
```

In the AES-round based construction search, the solution is composed of three vectors, thus $\epsilon(x)$ can be defined as a composition of three separate neighbour functions, one for each of the vectors. A pseudo-code of such function for the vector *aes_masks* is given below.

Algorithm 5 Neighbour function for *aes_masks*

```
Input: Binary array $aes\_masks$ of $s$ elements

$i \leftarrow randInt()\ \%\ s$                                ▷ Randomly choose an index
$aes\_masks' \leftarrow aes\_masks$                            ▷ Assign the vector
$aes\_masks'[i] = aes\_masks'[i] \oplus 1$                     ▷ Flip the bit

Output: $aes\_masks'$
```

C Efficient AES-Based Constructions

The only goal in the search for AES-based constructions presented in Section 3.2 was to improve the security in comparison to the already published constructions in [23], without affecting their efficiency. Further we focus on the latter goal, i.e. our imperative below is to improve the efficiency of the constructions, while still maintaining sufficient security level of at least 22 active Sboxes.

A construction has better efficiency if it has smaller state size, better rate, or both. Constructions that have smaller state size but worse rate, or vice versa, are not considered to be more efficient. To search for efficient constructions, once again we use the two metaheuristics. The formulations of the optimization problems are identical to the formulations given in Section 3.2, i.e. we still optimize with respect to the security and with fixed state size and rate. However, once a metaheuristic identifies a construction as optimal, we compare its efficiency to all of the constructions given in [23]. We report in Table 4 the constructions we found to be more efficient than some of the previously known constructions.

Table 4: More efficient AES-round based constructions found with metaheuristics. The numbers in bold denote better efficiency.

Source	**State size**	**Rate**	**Active Sboxes**	*aes_mask*	*feed_mask*	*messages*
[23]	6	3	22	111111	100100	011022
new	**5**	3	30	11001	10100	01101
[23]	7	3	25	1111110	1101101	1012020
new	**6**	3	27	111111	011100	102110
new	**6**	**2.5**	22	101111	110010	102120
[23]	8	3	34	11101110	11011101	10102020
new	8	**2.66**	26	11111111	11000010	10313032
new	**7**	**2.5**	26	1011110	0110100	1021102
[23]	9	3	25	111111111	100100100	011022033
new	9	**2.33**	25	111110011	001010000	201223033
[23]	12	2	28	110110110000	001001001000	111222333123
new	**10**	2	24	1110000100	0100100000	2122102110
new	**8**	2	22	00011011	01000100	12012122
new	**8**	2	26	11000000	01100000	11100010