A CP-based Automatic Tool for Instantiating Truncated Differential Characteristics ⋆

François Delobel, Patrick Derbez, Arthur Gontier, Loïc Rouquette, Christine Solnon

A CP-based Automatic Tool for Instantiating Truncated Differential Characteristics *

François Delobel[1], Patrick Derbez[2], Arthur Gontier[2], Loïc Rouquette[3,4], and Christine Solnon[5]

[1] Université Clermont-Auvergne, CNRS
Mines de Saint-Étienne, LIMOS
Clermont-Ferrand, France
francois.delobel@uca.fr
[2] Univ Rennes, CNRS, IRISA, Rennes, France
{patrick.derbez,arthur.gontier}@irisa.fr
[3] EPITA Research Laboratory (LRE), 14/16 rue Voltaire - 94270 Le Kremlin-Bicêtre, France
loic.rouquette@epita.fr
publications@loicrouquette.fr
[4] LORIA, Université de Lorraine, F-54000, Nancy, France
[5] INSA Lyon, CITI, INRIA CHROMA, F-69621 Villeurbanne, France
christine.solnon@insa-lyon.fr

Abstract. An important criteria to assert the security of a cryptographic primitive is its resistance against differential cryptanalysis. For word-oriented primitives, a common technique to determine the number of rounds required to ensure the immunity against differential distinguishers is to consider truncated differential characteristics and to count the number of active S-boxes. Doing so allows one to provide an upper bound on the probability of the best differential characteristic with a reduced computational cost. However, in order to design very efficient primitives, it might be needed to evaluate the probability more accurately. This is usually done in a second step, during which one tries to instantiate truncated differential characteristics with actual values and computes its corresponding probability. This step is usually done either with ad-hoc algorithms or with CP, SAT or MILP models that are solved by generic solvers. In this paper, we present a generic tool for automatically generating these models to handle all word-oriented ciphers. Furthermore the running times to solve these models are very competitive with all the previous dedicated approaches.

Keywords: Differential cryptanalysis, Constraint Programming, Automatic tool

* The work presented in this article was funded by the French National Research Agency as part of the DeCrypt project (ANR-18-CE39-0007).

1 Introduction

The security of a symmetric primitive mostly relies on applying all known cryptanalysis techniques to show that none of them is close to endanger it and to ensure that there is enough security margin against all known attacks. Among others, this evaluation process typically involves identifying the differential and linear characteristic which allows one to distinguish the primitive from a random permutation on as many rounds as possible. Indeed, several decades after their introduction, both differential [3] and linear [22] cryptanalysis are still two very powerful techniques receiving a lot of attention from the community.

Differential cryptanalysis aims at searching for differential distinguishers, which are formed by both an input and output difference such that the transition occurs with a probability higher than expected for a random permutation. The main issue with this cryptanalysis technique lies in the inherent difficulty of finding such differentials due to the extremely large search space. To overcome this difficulty it was proposed to look for differential characteristics, meaning that, instead of defining only the input and output differences, all differences in internal states are specified as well. However, searching for the best differential characteristics is not easier than searching for differential and thus, especially for word-oriented primitives, it was proposed to search for truncated differential characteristics. In a truncated differential characteristic, the exact difference on each word is omitted and replaced by a Boolean variable indicating whether there is a non-zero difference on the word or not. Associated with the best probability of a transition through the non-linear function involved in the primitive, truncated differential analysis provides an upper bound on the probability of any differential characteristic.

However, the bounds computed from truncated differential cryptanalysis are most often not tight, since it might be impossible to instantiate a truncated differential characteristic such that all its internal transitions occur with the highest possible probability. To evaluate more accurately the resistance of a primitive against differential cryptanalysis, an interesting problem is to find the best possible instantiation of a given truncated differential characteristic, *i.e.* , find the actual difference values maximizing the overall probability. Due to the huge search space, and because difference distribution tables (DDT) of non-linear cryptographic components are hard to model by means of linear or Boolean constraints, searching for the best instanciation of a given truncated differential characteristic most often relies on Constraint Programming (CP) solvers. Constraint programming is a flexible declarative language that comes with a large collection of constraints. A CP model is composed of variables, their corresponding domains and a set of constraint on these variables. A constraint is a relation between the variables and comes with an algorithm to check if the constraint is satisfied and an algorithm to remove the values that do not respect the relation from the domains of the variables. This algorithm is called a *filtering algorithm*. The CP solving method uses both a Depth-First Search algorithm and all the filtering algorithms of each variable to solve a problem. There are many different

types of constraints and new ones can be added to CP solvers if we can provide a corresponding filtering algorithm.

Regarding the specific problem of differential cryptanalysis, we can cite the collection of works from Gerault, Minier and their co-authors [23,11,9,30,10], in which the ultimate goal is to provide the probability of the best differential characteristic for each round-reduced version of all versions of the `Rijndael` cipher. It is also worth mentioning the recent work of Delaune et al. [5], who proposed a dedicated CP model to instantiate truncated differential characteristics on the cipher `SKINNY`.

Our Contribution. The common point of the previous CP models is that they are all dedicated to a specific cipher. Designing these dedicated CP models is time-consuming and error prone. In this paper, we propose a generic approach for automatically generating these models to handle all word-oriented ciphers, providing running times close enough to the dedicated models on both `Rijndael` and `SKINNY` while being fully generic. This CP model generator is a part of the Tagada project, a tool which aims at providing a simple and easy-to-use API to describe ciphers as well as the automatic generation of CP models to search for differential characteristics. However, our CP model generator can also be used as a standalone object, taking as input the description of the cipher and a truncated differential characteristic and outputting its best possible instantiation.

Organisation of the paper. Section 2 describes the Tagada tool, it defines the notion of Differential Cryptanalysis as well as the way in which Tagada works. Section 3 describes the first contribution of this article, namely a CP model generator for computing the best differential characteristic given a truncated characteristic. Section 4 explains the integration of this model generator within the Tagada library. Section 5 presents some optimizations and Section 6 presents the results obtained. Lastly, Section 7 recalls the tools introduced by this article and proposes new areas of research allowing the extension of automated differential cryptanalysis.

2 Tagada

Tagada (Tool for Automatic Generation of Abstraction-based Differential Attack) is a tool proposed in [20] that can generate models for computing truncated differential characteristics for any word-oriented cipher. It relies on a graph representation of the cipher and uses several techniques to optimize the models. In this background section, we first recall how differential is modelled and then explain the graph representation of Tagada. Moreover, we describe some of the optimizations Tagada proposed for the first step of this problem.

2.1 Differential cryptanalysis

Differential analysis is a method to analyze the effect of differences in plaintext pairs on the differences of the resultant ciphertexts. The difference is usually

obtained with the bit-wise XOR; we will note it $+$. For a cipher function F and two plaintexts x and y where y is created by injecting an input difference δ_{in}, *i.e.* $y = x+\delta_{in}$, the output difference δ_{out} is computed with $\delta_{out} = F(x)+F(y)$. There is a differential distinguisher in F if the probability that $\delta_{out} = F(x)+F(x+\delta_{in})$ is high *i.e.* the input difference δ_{in} has a good probability of ending up in the output difference δ_{out}. Symmetric ciphers are iterated functions like $F(x) = f(f(\dots f(f(x))\dots))$. To see if the difference δ_{in} can end up in a difference δ_{out}, we study the propagation of the difference through all the rounds and all the ciphers operators. We usually note δx_i the difference of an intermediate variable of the cipher x_i. The tracking of differences from δ_{in} to δ_{out} through the complete cipher is called a *differential trail* or *differential characteristic*.

Symmetric ciphers are generally composed of two types of operators: linear operators like XORs or permutations and non-linear operators like S-Boxes and multiplications :

- *Linear operators* will always propagate differences with probability 1. Indeed a permutation will simply reorganize the differences. Another common linear operator is the XOR of three variables $y = x_2 + x_3$ were the differences will be propagated with another XOR: $\delta y = \delta x_2 + \delta x_3$. Note that if $\delta x_2 = \delta x_3$ then the output difference is cancelled $\delta y = 0$.
- However, *non-linear operators*, called S-Boxes, will propagate a difference with a probability that may be lower than one. For each S-Box, the propagation probabilities of the pair of input-output differences can be computed with their *Difference Distribution Table* (DDT). Since it is necessary to enumerate all the possible pairs of inputs to generate the DDTs they are cached and it is not possible to compute them if the input size of the associated operator is too big. Usually it is possible to compute DDTs for word oriented ciphers as they are working with 4-bit (nibbles) and 8-bit (bytes) words. Tagada is designed for word oriented ciphers, for which the propagation of differences in non-linear operators can be computed with DDTs. Tagada will have very poor performances on bit-oriented ciphers (*e.g.* [1]), or on word-oriented ciphers that operate on words larger than bytes.

If a differential trail contains two or more S-Boxes, the probability of the trail is the multiplication of the probabilities given by the DDTs because we assume that the probabilities are independent [15]. Therefore, the probability of a trail will be lower if we add more S-Boxes to it, and low-probability trails may not be useful for standard differential attacks (differential trail with zero probability can be used in impossible attacks for example). To make ciphers more resistant, we could want to search for S-Boxes with perfectly balanced probabilities. However, this seems really hard to achieve among all the other required properties of S-Boxes [12].

As the search of differential trails is hard, it is usually split into two steps in the most recent works [4,6,10].

First step: find truncated trails The first step is the search for *truncated differential trails* [13]. A truncated differential trail is an abstraction of a differential trail in which we only retain whether a difference exists or not, *i.e.* each difference variable δx_i, associated to cipher's intermediate word x_i of size n, is abstracted by a Boolean variable Δx_i where

$$\Delta x_i = \begin{cases} 0 & \text{if} \quad \delta x_i = 0 \\ 1 & \text{if} \quad \delta x_i \in [1; 2^n - 1] \end{cases}$$

Because each Boolean variable Δx_i encodes the existence of the difference without tracking its value, several aspects of the problem change.

- For the probability of propagation through the S-Boxes, we cannot know which probability to pick in the DDT except when the Boolean variable associated with the input difference is equal to 0, in which case we know for sure that the Boolean variable associated with the output difference is also equal to 0. When the Boolean variable associated with the input variable is equal to 1, we only know that the S-Box is *active i.e.* involved in the trail. Therefore, we will use the highest probability to have an upper bound on the probability of the differential trail.
- The second change is that we cannot capture the difference cancellations of the XOR operations. Therefore, there are usually a lot of false positive trails *i.e.* trails that cannot be instantiated.

Truncated differential analysis can be enough to say that a cipher is secure in the case where the best truncated trail (the one with the fewest active S-Boxes) has a low enough probability. On the other hand, when a truncated trail has a high probability, we must successfully instantiate it with real difference values to have a differential trail.

Second step: instantiate the trails In the second step, we enumerate all possible differential trails (starting from active S-boxes) to find the best one. The two steps can be done separately (find all truncated trails, then try to instantiate them all) or together like in Algorithm 3 that will be explained more in Section 4.

In [5], the authors compare a dedicated implementation (based on dynamic programming) with SAT, MILP, and CP models that are solved by generic solvers on the two steps of the differential cryptanalysis of the cipher SKINNY. In conclusion, they found that their hand-made algorithm is the fastest on the first step, followed by SAT and MILP. For the second step, they conclude that the CP solver is the most efficient solver by far. This work points out a key issue of cryptanalysis problems. It has to be done on every cipher, and the development time is not the same for a hand-made algorithm or a model for tools like SAT, MILP and CP.

2.2 How Tagada works

In [20], a generic graph representation of ciphers was proposed. This graph is encoded in a text format to be able to simplify the communication of the cipher definition. TAGADA uses this graph to generate CP,SAT of MILP models to solve the first step of the differential analysis, and we will use the same graph to generate the second step too.

DAG: unifying description of ciphers The input graph is a Directed Acyclic graph where the nodes correspond to all the cipher parameters (inputs, outputs, constants...) and operators (XOR, S-Box, permutations...). The edges of the DAG link the operators to the parameters. Hence the DAG is a bipartite graph. Figure 1 shows the DAG of a 2-round toy example Feistel cipher.

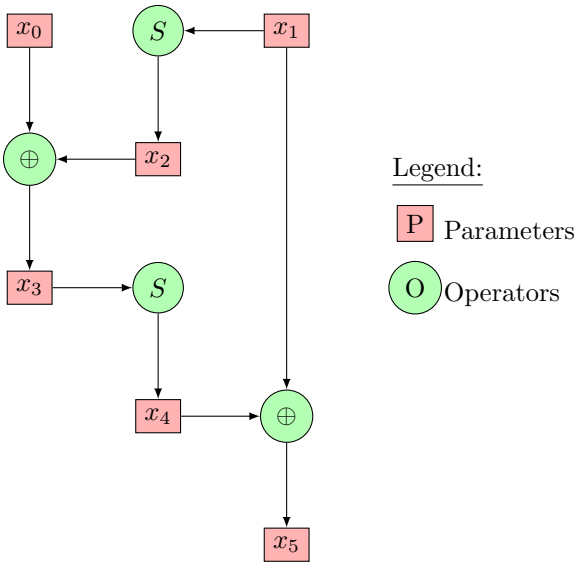

Fig. 1. DAG of a simple 2-round toy example Feistel cipher.

The text format used to define the DAG is a JSON composed of a list of three types of objects.

- The variables with their domain ranges.
- The functions (the operators) with input domains, output domains and specificities. For example, an S-Box will declare its lookup table, a LFSR will declare its length, shift direction and feedback polynomial...
- The transitions are triplets composed of a list of variables, a function and another list of variables. They describe the link between the operators and their input and output variables.

The advantage of a text format like JSON is that it is easy to generate and parse for any language. Moreover, the DAG has to be made only once for each cipher, thus saving a lot of time compared to the development of a hand-made algorithm or a solver-specific model. The difficulty of the DAG representation is that the DAG must be able to represent all the operators in order to be able to model any cipher. Tagada currently handles the following operators: equality, bit-wise XOR, Galois field multiplication, LFSR, left shift register, right shift register, permutation, concatenation, split and S-Box. Tagada also proposes to describe new operators by means of tables describing all the possible in/out tuples. However, this option is possible only when the table is not too large.

Truncated differential graph and optimizations To solve the first step of the differential analysis, Tagada first builds a truncated version of the graph of the input cipher. Then, this graph is optimized with a simplification of the useless parts of the graph. Indeed, differential analysis does not use constants nodes, and equality operators can be removed from the graph by merging the equals nodes.

A second optimization is done to detect some inconsistent differential trails by adding constraints, as illustrated in Example 1.

Example 1. Let $\delta_1, \delta_2, \delta_3$ be three differential variables, and $\Delta_1, \Delta_2, \Delta_3$ be their corresponding truncated differential variables. Let $\Delta_1 + \Delta_2 = 0$ and $\Delta_1 + \Delta_2 + \Delta_3 = 0$ be two equations generated from the DAG.

If we look at the first equation, it is satisfied if $\Delta_1 = 0$ and $\Delta_2 = 0$. However, the difference can also be cancelled if they have the same value ($\delta_1 = \delta_2$). Therefore, the equation is also satisfied if $\Delta_1 = 1$ and $\Delta_2 = 1$.

For the same reason, the second equation accepts the solution $\Delta_1 = 1, \Delta_2 = 1, \Delta_3 = 1$. In the truncated model, there is no problem. However, in the second step model, the first equation implies that $\delta_1 = \delta_2$ and that they are equals to a non-zero difference, so $\delta_3 \neq 0$ would never be a valid assignment.

To detect this kind of inconsistencies, Tagada combines XOR equations to generate new equations. Generating new equations is a key point for an efficient model. These equations were hand-made in [11,10], whereas they are automatically derived from the DAG in Tagada. In [28], an abstract-XOR constraint has been designed to better propagate XOR constraint in CP solvers.

Once optimized, a mathematical model is automatically generated from the truncated graph. This model is expressed using the MiniZinc language, which is a high-level language for defining constraint satisfaction problems [24]. Many different solvers are able to solve problems defined in MiniZinc such as, for example, Choco, Chuffed or Picat. We may also use Picat to automatically generate SAT or MILP models from a MiniZinc model, thus allowing one to use SAT or MILP solvers. In many cases, the best solver is Picat-SAT.

2.3 First step results

In [20] the TAGADA models were able to recover the state-of-the-art truncated trails of the ciphers `AES`, `Midori`, `SKINNY`, and `Craft` in either single-key or related-key scenarios.

However, truncated trails may not lead to valid differential trails. To be able to make a strong statement on the differential characteristics of a cipher, we must try to instantiate these trails with the second step. Therefore, another program or model has to be made to solve the second step, and we propose to generate it from the DAG representation of the cipher like TAGADA generated the first step models.

3 Model generation for the second step

In this section, we present our contribution to the TAGADA project, focusing on the modelling of the second step of the differential analysis, the instantiation of truncated characteristics. Contrary to the first step, we rely only on a CP solver, namely Choco [25]. There are two reasons for that. Firstly, in both [11] and [5], the CP solver was said to perform very well for this particular problem. Secondly, we wanted to develop dedicated filtering algorithms for operators like the bit-wise XOR in a CP solver to improve the overall efficiency of the solving process.

In the second step of the differential analysis, we use the solutions of the first step and we try to instantiate them. More precisely, we make a complete model of the cipher, and we constrain the S-Boxes variables according to the truncated trail *i.e.* the active S-Boxes in the truncated trail have a full domain, and the inactive S-Boxes be set to zero. To generate models for the second step, we need constraints for each operator and the most important one is the S-Box.

3.1 Modelling DDT with table constraints

The probability of the propagation of a differences through an S-Box is described in a *Difference Distribution Table* (DDT). For example, the DDT of an S-Box of the F function of `DES` is given in Table 3.1. This S-Box has six input bits and four output bits. Therefore, the DDT is a table with $2^6 \times 2^4$ entries. In this table, we can see that the first difference (0) always propagates to 0 (64 times over the 64 possible input pairs of difference 0). In the second line, we can see that the input difference 1 propagates to the output difference 3 six times over 64 possible input pairs of difference 1. Therefore, the probability of this propagation is 6×2^{-6}. In the truncated model, we would use only the best probability of this table which is 16×2^{-6}, but this probability holds only for one transition ($34 \rightarrow 2$). We use the best transition probability in order to use the probability of the truncated differential trail as an upper bound approximation of the probability of the differential trail - the best possible differential trail is the differential trail that uses only optimal transitions.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**64**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	**6**	0	2	4	4	0	10	12	4	10	**6**	2	4
2	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
3	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
4	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
5	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
6	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
7	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
8	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
9	10	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
A	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
B	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
C	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
D	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
E	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
F	2	0	2	4	4	6	4	2	4	8	2	2	2	6	8	8

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0	0	0	0	0	0	2	14	0	6	6	12	4	6	8	6
11	6	8	2	4	6	4	8	6	4	0	6	6	0	4	0	0
12	0	8	4	2	6	6	4	6	6	4	2	6	6	0	4	0
13	2	4	4	6	2	0	4	6	2	0	6	8	4	6	4	6
14	0	8	8	0	10	0	4	2	8	2	2	4	4	8	4	0
15	0	4	6	4	2	2	4	10	6	2	0	10	0	4	6	4
16	0	8	10	8	0	2	2	6	10	2	0	2	0	6	2	6
17	4	4	6	0	10	6	0	2	4	4	4	6	6	6	2	0
18	0	6	6	0	8	4	2	2	2	4	6	8	6	6	2	2
19	2	6	2	4	0	8	4	6	10	4	0	4	2	8	4	0
1A	0	6	4	0	4	6	6	6	6	2	2	0	4	4	6	8
1B	4	4	2	4	10	6	6	4	6	2	2	4	2	2	4	2
1C	0	10	10	6	6	0	0	12	6	4	0	0	2	4	4	0
1D	4	2	4	0	8	0	0	2	10	0	2	6	6	6	14	0
1E	0	2	6	0	14	2	0	0	6	4	10	8	2	2	6	2
1F	2	4	10	6	2	2	2	8	6	8	0	0	0	4	6	4

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20	0	0	0	10	0	12	8	2	0	6	4	4	4	2	0	12
21	0	4	2	4	4	8	10	0	4	4	10	0	4	0	2	8
22	10	4	6	2	2	8	2	2	2	2	6	0	4	0	4	10
23	0	4	4	8	0	2	6	0	6	6	2	10	2	4	0	10
24	12	0	0	2	2	2	2	0	14	14	2	0	2	6	2	4
25	6	4	4	12	4	4	4	10	2	2	2	0	4	2	2	2
26	0	0	4	10	10	10	2	4	0	4	6	4	4	4	2	0
27	10	4	2	0	2	4	2	0	4	8	0	4	8	8	4	4
28	12	2	2	8	2	6	12	0	0	2	6	0	4	0	6	2
29	4	2	2	10	0	2	4	0	0	14	10	2	4	6	0	4
2A	4	2	4	6	0	2	8	2	2	14	2	6	2	6	2	2
2B	12	2	2	2	4	6	6	2	0	2	6	2	6	0	8	4
2C	4	2	2	4	0	2	10	4	2	2	4	8	8	4	2	6
2D	6	2	6	2	8	4	4	4	2	4	6	0	8	2	0	6
2E	6	6	2	2	0	2	4	6	4	0	6	2	12	2	6	4
2F	2	2	2	2	2	6	8	8	2	4	4	6	8	2	4	2

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
30	0	4	6	0	12	6	2	2	8	2	4	4	6	2	2	4
31	4	8	2	10	2	2	2	2	6	0	0	2	2	4	10	8
32	4	2	6	4	4	2	2	4	6	6	4	8	2	2	8	0
33	4	4	6	2	10	8	4	2	4	0	2	2	4	6	2	4
34	0	8	**16**	6	2	0	0	12	6	0	0	0	0	8	0	6
35	2	2	4	0	8	0	0	0	14	4	6	8	0	2	14	0
36	2	6	2	2	8	0	2	2	4	2	6	8	6	4	10	0
37	2	2	12	4	2	4	4	10	4	4	2	6	0	2	2	4
38	0	6	2	2	2	0	2	2	4	6	4	4	4	6	10	10
39	6	2	2	4	12	6	4	8	4	0	2	4	2	4	4	0
3A	6	4	6	4	6	8	0	6	2	2	6	2	2	6	4	0
3B	2	6	4	0	0	2	4	6	4	6	8	6	4	4	6	2
3C	0	10	4	0	12	0	4	2	6	0	4	12	4	4	2	0
3D	0	8	6	2	2	6	0	8	4	4	0	4	0	12	4	4
3E	4	8	2	2	2	4	4	14	4	2	0	2	0	8	4	4
3F	4	8	4	2	4	0	2	4	4	2	4	8	8	6	2	2

Table 1. DDT of one S-Box of DES

DDT to table constraint. The main advantage of the CP solver is that we can directly model the DDT with a table constraint. A table constraint constrains a list of n variables to be instantiated to an n-tuple chosen within a collection of all valid n-tuples. Any constraint can be declared in *extension i.e.* declared as a table constraint. However, this might not be the best way to model a constraint, especially if there are a lot of tuples. For example, to constrain three Boolean variables a, b, and c to have a sum equal to 2, we may define the table constraint $(a, b, c) \in \{(1, 1, 0), (1, 0, 1), (0, 1, 1)\}$ but the same sum with integer variables (with domains like $[\![-1000, 1000]\!]$) would require too many tuples. This is why constraints are often best declared in *intention i.e.* with a dedicated filtering algorithm. However, an efficient filtering algorithm may not always be available.

Table filtering. The table constraint not only requires a filtering algorithm but also needs to be cautious of the data structure used to store and manipulate the table. This is because the memory used by the table depends on the number of tuples. In the literature [18,7,33,16,17,21], a lot of work has been done to optimize this algorithm for various situations (positive tables, binary tables, big

or small tables...). The general idea is to pre-compute and try to maintain a set of indexes of valid tuples. When a tuple is no longer valid, an efficient search method will search for another valid tuple in the table. If there are no more valid tuples, the constraint is violated. There are a lot of variations depending on the situation.

The DDT is an extensive definition of all the possible transitions of a difference through an S-Box with its probability and it is usually not possible to define these transitions in intention, by means of a small number of arithmetic constraints. Therefore, the table constraint is the most suited way to model it. In CP, this takes the form of a table T composed of a list of tuples $(\delta x_{in}, \delta x_{out}, p) \in T$ where δx_{in} and δx_{out} are the input and output variables of the S-Box, respectively, p is a variable which corresponds to the probability of observing the output difference δx_{out} given the input difference δx_{in}. To avoid rounding errors, the probability is replaced by the negation of its base 2 logarithm (and probability multiplications are replaced with additions).

For MILP and SAT, this table would require a lot more intermediate variables and constraints [32].

3.2 Modelling other operators

Unfortunately, the other operators are not available in CP solvers except for some exceptions, like the modular addition. To model the other operators, we could also use table constraints. However, tables are often too large to be efficient. Therefore, we will develop new filtering algorithms. In particular for the bit-wise XOR operator because it is used everywhere.

Bit-wise XOR. We first consider the bit-wise XOR in the case with three variables: $a + b = c$. Note that if there is only one variable, the XOR is a constant. If there are two variables, the XOR can be replaced with an equality operator. For the three variable case, we used some previous work from [5]. The filtering algorithm is not very smart. To filter the values of D_c, the domain of c, the algorithm computes a set that contains all the possible XORs between the values of the domains of a and b. This set is then used to remove the inconsistent values from the domain of c. The algorithm is given in Algorithm 1.

This algorithm has two weaknesses. First, the computing of the set is time-consuming but most of all, the set can reach the maximum size very fast. Indeed, if a, b and c have the same domain sizes, for example, 8 bits, then this algorithm will compute $2^8 \times 2^8$ XORs but will fill a set of maximum 2^8 values. In practice, this filtering algorithm takes a lot of time to build the set that does not filter anything in most cases. Therefore, this filtering is only performed in some cases decided by a chosen condition. In [5], this condition is that the sum of the domains of a and b is lower than the maximal domain size of c. We will see later that this condition is not the best one.

Bit-wise XOR of arbitrary arity. To model a XOR of higher arity with only three variable XOR constraints, we need to introduce intermediate variables and

Algorithm 1: 3-variable XOR filtering algorithm

Input: IntVar a, IntVar b, IntVar c: the target domain to filter

```
1 set ← ∅;
  // Loop through possible values
2 for all v1 ∈ D_a do
3     for all v2 ∈ D_b do
4         set ← set ∪ {v1 ⊕ v2};
5         if set contains all possible values in variable domains then
6             return;
7 D_c ← D_c ∩ set;
```

declare additional XORs. For Tagada, we wanted to avoid the introduction of new variables, so we extended the idea of this algorithm to a XOR of arbitrary arity. The algorithm uses a recursive loop to compute the set of all the possible XORs between $n-1$ variable domain values to filter the target domain. The algorithm is depicted in Algorithm 2. This algorithm is less efficient to constrain a 3-variable XOR than the previous propagator but the more variables we have, the more efficient this algorithm becomes compared to a decomposition with 3-variable XORs constraints and new intermediate variables (more than five variables XORs decompositions gives slower models). However, as we will see at the end of this section, the chosen condition to activate the filtering was also a problem.

Operations in the Galois Field. In cryptography, we sometimes use addition and multiplication in some fields. For CP modelling, operations like the MixColumns of `AES` have often been modelled with table constraints [23]. For the modular addition, the modulo constraint exists in CP solvers. To model the modular addition $a \boxplus b = c \mod m$, we need one intermediate variable x and the two constraints $a + b = x$ and $x \mod m = c$. Unfortunately, modelling the multiplication is not possible with existing constraints. Like the XOR, we must make a new filtering algorithm for multiplication and division in a finite field. The filtering algorithm we made follows the same idea of Algorithm 1. We create a set of all the possible values of c from the domains of a and b except that in the line "`set` $\leftarrow$ `set` $\cup \{v1 \oplus v2\}$" the $\oplus$ is replaced by ProdGF($v1$,$v2$) or DivGF($v1$,$v2$) where ProdGF and DivGF are algorithms to perform the modular product and division depending on the context. In real ciphers, the product is usually between a variable and a constant, so the algorithm is more likely to filter properly.

LFSR. Another operator we added is the LFSR. Similarly to the bit-wise XOR, we replace the line 4 of Algorithm 1 with a function that can compute the next step of the LFSR.

Algorithm 2: n-variable XOR filtering algorithm

```
   Input: int target: index in vars table of the domain to filter
 1 Function combiXor(target, current, xor):
       // skip target
 2     if current == target then
 3         combiXor(target, current + 1, xor);
 4     else
           // add the value xor to the set of values
 5         if current == vars.length - 1
 6               or (current + 1 == target
 7               and current + 1 == vars.length - 1) then
 8             for all v ∈ D_current do
 9                 set ← set ∪ {xor ⊕ v};
10         else
               // Loop through domain with recursion
11             for all v ∈ D_current do
12                 combiXor(target, current + 1, xor ⊕ v);
13 set ← ∅;
14 combiXor(target, 0, 0);
15 D_target ← D_target ∩ set;
```

Concat and Split. Bit concatenation and bit splitting are modelled by constraint tables.

Filtering efficiency. As stated before, the filtering of bit-wise XOR constraints needs a parameter to avoid useless computations. During our tests, we observed that the trade-off between the time gained from filtering and the time taken by the filtering algorithm is in favour of less filtering. To be efficient, we must find at which domain size we would have a good chance to filter. For the XOR with two variables, we can write it as follows. Let a, b be two variables, D_a and D_b, their domain of max sizes n. Let c be a constant in the constraint $a + b = c$. In this case, we can filter the values in D_b if this set of values is not contained in D_a. For two variables, this condition is very probable. However, if now c is also a variable with a domain D_c of max size n, then the filtering is possible if the XORs of all the possible values of D_a and D_b is a set that does not include D_c. The number of values from the XORs is $\#D_a \times \#D_b$, and each value is in the domain range of D_c. In the end, it is like if we pick at random $\#D_a \times \#D_b$ values in the range of $[\![0, n]\!]$ and hope that we did not pick all the values of D_c. For the XOR with more variables ($\bigoplus_i x_i$), the number of values is $\prod_{i \neq j} \#D_{x_i}$. As a consequence, the probability of being able to filter some values of D_j is very low. To test the filtering efficiency of our XOR constraints, we implemented a *forward-checking* version of the filtering algorithms. The forward-checking (FC) method only filters when all the variables are fixed except one. In some early tests, we saw that the FC version was performing nearly as well as the full filtering algorithms.

This means that the filtering algorithms for the 3-variable and n-variable XOR constraints are not helping the solving process that much. Therefore, we deduce that the CP model's strength is the DDT's table constraint. Moreover, the new dedicated filtering algorithms are still helpful because we would have to use table constraints instead, so they at least reduce the memory used by the model. A list of the operators we added to the model generator is depicted in Table 2.

Linear Operators

Operator	Name	First step support	Second step Implementation
$=$	Equal	✓	Native support
`LFSR`	Linear Feedback Shift Register	✓	Custom filtering algorithm
$AB \to (A, B)$	Split	✓	Constraint table (native)
$(A, B) \to AB$	Concat	✓	Constraint table (native)
$\ll$ or $\gg$	Left (Right) Shift	✓	Custom filtering algorithm
$\lll$ or $\ggg$	Left (Right) Circular Shift	✓	Custom filtering algorithm
$\&_K$	Bitwise AND with Constant	✓	Constraint table (native)
$\|_K$	Bitwise OR with Constant	✓	Constraint table (native)
$\oplus$	N-ary Bitwise XOR	✓	Custom filtering algorithm or decomposition (for n-ary equations)
$\otimes_K$	Galois Field Multiplication with Constant	✓	Custom filtering algorithm
$\odot_K$	Galois Field Matrix Multiplication with Constant Matrix	✓	Decomposition and delegation to the $\otimes_K$ and $\oplus$ operators
`T`	Linear Lookup Table	✓	Constraint table (native)

Non-linear Operators

Operator	Name	First step support	Second step Implementation
DDT	Differential Distribution Table	✓	Constraint table (native)

Table 2. List of supported operators in Tagada (both first and second steps). For exemple, Rijndael uses the operators: $\oplus$, $=$, $\odot_K$ and DDT, while Skinny uses the operators $\oplus$, $=$, $\odot_K$, `LFSR` and DDT.

4 Connect the two steps

At the start the first step of Tagada was not designed to work with the second step but only to find the best truncated differential trail. While it can be possible to only use truncated differential trails optimizing, the whole process can be more efficient than only optimizing the two steps separately. To do so we improve the linking algorithm of [27] by splitting the first step search in three parts.

Step1-opt. The aim of Step1-opt is to find the optimal truncated differential trail. We define its signature with Signature 1.

Step1-next. Instead of looking for an optimal solution Step1-next (Signature 2) is designed to find one truncated differential trail with a given upper bound (UB).

Signature 1 STEP1-OPT

Input:

G_Δ: the Differential Graph of the cipher
seen: the set of all the already found solutions
UB: the current upper bound

Output:

sol: the Truncated Differential Trail with the highest probability such as $P(sol) \leq UB$ and *sol* not in *seen*. If no such solution exists, returns `null`.

Signature 2 STEP1-NEXT

Input:

G_Δ: the Differential Graph of the cipher
seen: the set of all the already found solutions
UB: the current upper bound

Output:

sol: a Truncated Differential Trail such as $P(sol) \leq UB$ and *sol* not in *seen*. If no such solution exists, returns `null`.

While Step1-opt solves an optimization problem, Step1-next solves a satisfaction problem.

Usually solving optimization problems is more complicated than solving a satisfaction problem. Indeed for both optimization and satisfaction problems we have to find a solution but for optimization problems we also need to find the best one which is generally done by finding a sequence of solutions of increasing quality and proving that there is no better solution than the last found one.

To improve the overall time of the two steps algorithm (Algorithm 3) we try to use the Step1-next method as much as possible instead of the Step1-opt method. Step1-opt is only called at the beginning of the function when we have to compute the upper bound. The second call of Step1-opt is done when we have iterated over all the solutions that reach the current upper bound.

Step1-next-possible-UB (Signature 3). As said previously, step1-opt is the function that consumes the most calculation time. When we have iterated over all the solutions of a current upper bound we need to find the next upper bound. The search of the next upper bound is performed by another call to Step1-opt. However it is possible in some cases to bypass the function by using a new function Step1-next-possible-UB. The purpose of Step1-next-possible-UB is to find very quickly a lower approximation of the next upper bound. If this approximation is equal or lower than the current lower bound then we can stop the search without doing any more computation.

Example 2. Let us take the example of the 4-round `Rijndael-128-128` instance. The step1-opt finds a truncated differential trail with an upper bound probability of 2^{-72}. For this trail the second step will find a valid differential trail of probability 2^{-75} and set it as the current lower bound. As $2^{-75} < 2^{-72}$ we need to find another truncated differential trail that matches 2^{-72}. For this cipher

Algorithm 3: TWOSTEP(G_Δ, G_δ)

```
   Input:
     G_Δ: step1 model
     G_δ: step2 model
 1 LB ← 0
 2 UB ← 1
 3 best ← null
 4 sol_1 ← STEP1-OPT(G_Δ, seen, UB)
 5 seen ← {}
 6 UB ← P(sol_1)
 7 while LB < UB do
 8 |   seen ← seen ∪ {sol_1}
 9 |   sol_2 ← STEP2(G_δ, sol_1, LB)
10 |   LB ← P(sol_2)
11 |   if LB < UB then
12 |   |   sol_1 ← STEP1-NEXT(G_Δ, seen, UB)
13 |   |   if sol_1 is null then
14 |   |   |   UB ← STEP1-NEXT-POSSIBLE-UB(G_Δ, seen, UB)
15 |   |   |   if LB ≥ UB then
16 |   |   |   |   break
17 |   |   |   sol_1 ← STEP1-OPT(G_Δ, seen, UB)
18 |   |   UB ← P(sol_1)
19 return best
```

we only have one truncated differential trail of this probability. As we have a gap between the two probabilities we need to tighten the bounds which is done by decreasing the upper bound. For `Rijndael` all the S-Boxes are the same and their maximum probability is 2^{-6} (a trail of 2^{-72} is composed of 12×2^{-6} S-Boxes). In our case, the only way to find a new trail is to activate another S-Box, in that case the probability will be at least of $13 \times 2^{-6} = 2^{-78}$ which is lower than 2^{-75}. This simple computation saves one call of Step1-opt.

Signature 3 STEP1-NEXT-POSSIBLE-UB

Input:
G_Δ: the Differential Graph of the cipher
UB: the current upper bound
Output:
UB': an approximation of the next reachable UB.

5 Second Step Optimizations

To gain some generic solving efficiency, we propose two optimizations.

5.1 Heuristics

Another way to improve the search speed is to use heuristics. Constraint Programming solvers usually use two kind of heuristics, a value heuristic and a variable heuristic. As their names suggest, the value heuristic is responsible of selecting the next value to test for a variable and the variable heuristic is used to select the next variable on which to branch. By default, generic solvers propose known general purpose search heuristics for both value and variable heuristics, but when we have extra information on a specific problem we can help the solver by designing custom heuristics.

We propose a custom value heuristic adapted to our second step. In this step, we want to maximize the probability of the differential trail. As the probability only depends on non-linear operators we can focus our heuristics on those operators. For each non-linear operator we have three available variables:

- δx which is the input differential variable
- δsx which is the output differential variable
- p which is the probability of the transition $\delta x \rightarrow \delta sx$

If the solver branches on a p variable, we only have to select the highest probability available. When the solver branches on a δx variable, if its δsx variable is instantiated, *i.e.* it has only one possible value, then we can select the value that maximize the transition to δsx, more formally:

$$\texttt{next-value}(\delta x) = \underset{v \in \texttt{dom}(\delta x)}{\text{argmax}}\ P(v \rightarrow \texttt{value}(\delta sx))$$

When the solver branches on a δsx we can performs the same computation with its corresponding δx variable.

5.2 Competitive parallel solving

In the previous CP model for the second step on SKINNY [5], a parallel method was used. Each model from the first step was launched on a separate thread, and the best solution found was shared with all the other threads. This new bound is added to the models that remain to be solved, and this information can be used to cut a large part of their search space.

In Tagada, we added a similar parallel competition between the models to solve the second step models from a list of truncated trails. In this setup, Tagada was able to recover the same results with similar solving times than the dedicated model of [5]. The generated models were only two to three times slower than the ad-hoc models.

An interesting future work would be to study the feasibility of parallel computing in the two-step method (Algorithm 3).

6 Results

We have implemented the model generator in Java and Kotlin to communicate more easily with the Choco solver. This generator can parse the JSON file of the Tagada DAGs and an associated file for the truncated trails. The generator then builds a CP model and calls the Choco solver to find the differential characteristics. The second step model generator can be used with the first step of Tagada in the two-step solving algorithm depicted in Algorithm 3, or it can also be used alone if we can give a truncated trails list as input.

To compare the generated models, we reproduced the results of ad-hoc models with Tagada on the two-step differential analysis. We set a time limit of one day. The computation was done on a Debian GNU/Linux 11 (bullseye) x86_64 sever with two Intel Xeon Gold 6254 (3.10 to 4.00 GHz) processors and 64MB of RAM. Each instance was launch on a single core. We were able to reproduce the results shown in Table 6.

The code will be available at:

`https://gitlab.com/tagada-framework/tagada`

7 Conclusion

We have presented a model generator for the second step of the differential analysis that relies on the DAG representation of the model generator of the first step Tagada. We have shown that these generated models can recover the results of state-of-the-art ad-hoc models in reasonable times.

Cipher	Max Round	Probability	Reference
Midori-64	16	2^{-16}	[8]
Midori-128	20	2^{-40}	[8]
Warp	41	2^{-40}	[31]
Twine-80	18	2^{-64}	[29]
Twine-128	16	2^{-52}	[29]
Skinny-64-TK1	11	2^{-64}	[5]
Skinny-128-TK1	11	2^{-74}	[5]
Rijndael-128-128	5	2^{-105}	[10]
Rijndael-128-160	7	2^{-120}	[27]
Rijndael-128-192	9	2^{-146}	[10]
Rijndael-128-224	12	2^{-212}	[27]
Rijndael-128-256	14	2^{-146}	[10]
Rijndael-160-128	4	2^{-112}	[27]
Rijndael-160-160	6	2^{-138}	[27]
Rijndael-160-192	8	2^{-141}	[27]
Rijndael-160-224	9	2^{-190}	[27]
Rijndael-160-256	11	2^{-204}	[27]
Rijndael-192-128	3	2^{-54}	[27]
Rijndael-192-160	5	2^{-118}	[27]
Rijndael-192-192	7	2^{-153}	[27]
Rijndael-192-224	8	2^{-205}	[27]
Rijndael-192-256	9	2^{-179}	[27]
Rijndael-224-128	3	2^{-54}	[27]
Rijndael-224-160	4	2^{-122}	[27]
Rijndael-224-192	5	2^{-124}	[27]
Rijndael-224-224	7	2^{-196}	[27]
Rijndael-224-256	8	2^{-182}	[27]
Rijndael-256-128	3	2^{-54}	[27]
Rijndael-256-160	4	2^{-130}	[27]
Rijndael-256-192	5	2^{-148}	[27]
Rijndael-256-224	4	2^{-115}	[27]
Rijndael-256-256	6	2^{-128}	[27]

Table 3. Best differential trails recovered with Tagada (time limit of one day). Detailed results will be available in an extended version of the current paper on eprint.

7.1 Next optimization: DAG simplification

To make the model more efficient, we would like to add a graph simplification algorithm. From the first step solution, we know which S-Boxes are active and which are not. Therefore, we can simplify the model by removing the variables in the inactive part of the graph and all the related constraints. For example, let G be a graph composed of two S-Boxes $S1$ and $S2$, their output variables out_1 and out_2 and one XOR operation $out_1 \oplus out_2 = out_3$ (see Figure 7.1). If the truncated trail says that only the first S-Box is active, then instead of fixing the domain of out_2 to 0, we can remove the $S2$ and out_2 nodes from the graph. After this, we can further simplify the graph by removing the XOR node and the out_1 variable. In the end, we only need to keep $S1$ and out_3.

The simplification is split into two parts. The first part propagates from the active S-Boxes, all the nodes that can or will have a difference. In some cases, we can be more precise. In particular, we know that the difference always propagates through unary operators, so we can propagate this information in the

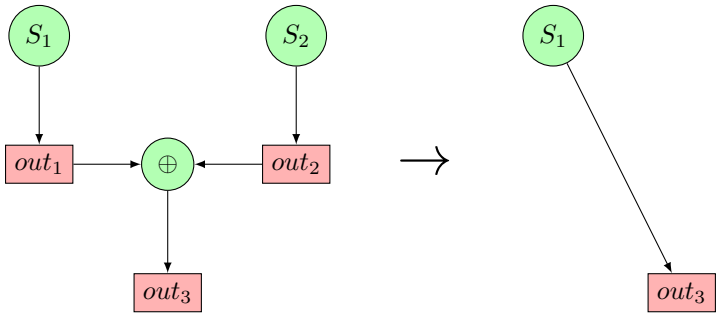

Fig. 2. Step2 graph shaving example

graph. This information can then be used in the CP model. When we know that a variable necessarily has a difference, we can remove the value 0 from the variable domain when we declare it. We start with a graph containing all nodes with an "uncertain" marker. For each input and output variable of the S-Boxes, we use the information of the truncated trail to fix them to "active" or "inactive". For the linear operators, we can deduce the following *status propagation rules* :

- If there is only one "uncertain" variable in its input or output variables and all the other variables are inactive, it can be set to "inactive".
- If there is only one variable "active" and all the others are inactive, this is an error. This is not possible if the truncated trail is correct.
- If there is only one "active" variable and one "uncertain" variable, they both must be "active".

"active" and "inactive" information are iteratively propagated in the graph until reaching a fixpoint where no more propagations are possible.

After these statuses are propagated, we can reduce the graph by removing all the "inactive" variables and all the operators only linked to them. We can also replace the XOR operators that have one inactive variable with equality operators, and finally, we can remove the equality operators and merge their input and output nodes. The simplified graph is then transformed into the CP model.

If we give the complete graph to the CP solver, it would eventually reach the same conclusion and set all the inactive variables to 0. However, this simplification is easy to do in advance and removing useless variables and constraints in a model is always a good idea.

7.2 Future work.

The idea of a simple tool to perform differential analysis is interesting, and other recent works are also working in this direction [26,2]. Moreover, we think that a unified cipher format would help the comparison and development of these tools. Tagada could be improved by integrating previous solving methods dedicated

to ARX ciphers [19] and bit-based ciphers [14]. In these cases, more work is needed to determine if the CP solver is still the best solving tool. In the end, Tagada could be extended to search for the variants of differential analysis (boomerangs, impossible differentials...).

Acknowledgements The authors would like to express their very great appreciation to Charles Prud'homme, Ph.D. from IMT for his valuable and constructive expertise of Choco during the development of this research work.

References

1. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013), https://eprint.iacr.org/2013/404
2. Bellini, E., Gérault, D., Grados, J., Huang, Y.J., Rachidi, M., Tiwari, S.K., Makarim, R.H.: CLAASP: a cryptographic library for the automated analysis of symmetric primitives. IACR Cryptol. ePrint Arch. p. 622 (2023), https://eprint.iacr.org/2023/622
3. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer (1990). https://doi.org/10.1007/3-540-38424-3_1
4. Biryukov, A., Nikolic, I.: Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to aes, camellia, khazad and others. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6110, pp. 322–344. Springer (2010). https://doi.org/10.1007/978-3-642-13190-5_17
5. Delaune, S., Derbez, P., Huynh, P., Minier, M., Mollimard, V., Prud'homme, C.: Efficient methods to search for best differential characteristics on SKINNY. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 21: 19th International Conference on Applied Cryptography and Network Security, Part II. Lecture Notes in Computer Science, vol. 12727, pp. 184–207. Springer, Heidelberg, Germany, Kamakura, Japan (Jun 21–24, 2021). https://doi.org/10.1007/978-3-030-78375-4_8
6. Fouque, P., Jean, J., Peyrin, T.: Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 183–203. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_11
7. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Data structures for generalised arc consistency for extensional constraints. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada. pp. 191–197. AAAI Press (2007), http://www.aaai.org/Library/AAAI/2007/aaai07-029.php
8. Gérault, D.: Security analysis of contactless communication protocols. (Analyse de sécurité des protocoles de communication sans contact). Ph.D. thesis, University of Clermont Auvergne, Clermont-Ferrand, France (2018), https://tel.archives-ouvertes.fr/tel-02536478
9. Gérault, D., Lafourcade, P.: Related-key cryptanalysis of midori. In: Dunkelman, O., Sanadhya, S.K. (eds.) Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10095, pp. 287–304 (2016). https://doi.org/10.1007/978-3-319-49890-4_16
10. Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Computing AES related-key differential characteristics with constraint programming. Artif. Intell. **278** (2020)

11. Gérault, D., Minier, M., Solnon, C.: Constraint programming models for chosen key differential cryptanalysis. In: Rueher, M. (ed.) Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9892, pp. 584–601. Springer (2016). https://doi.org/10.1007/978-3-319-44953-1_37
12. Heys, H.M.: A tutorial on linear and differential cryptanalysis. Cryptologia **26**(3), 189–221 (2002). https://doi.org/10.1080/0161-110291890885
13. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings. Lecture Notes in Computer Science, vol. 1008, pp. 196–211. Springer (1994). https://doi.org/10.1007/3-540-60590-8_16
14. Kölbl, S.: Cryptosmt: An easy to use tool for cryptanalysis of symmetric primitives (2015). URL: https://github. com/kste/cryptosmt
15. Lai, X., Massey, J.L., Murphy, S.: Markov ciphers and differential cryptanalysis. In: Davies, D.W. (ed.) Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings. Lecture Notes in Computer Science, vol. 547, pp. 17–38. Springer (1991). https://doi.org/10.1007/3-540-46416-6_2
16. Lecoutre, C.: STR2: optimized simple tabular reduction for table constraints. Constraints An Int. J. **16**(4), 341–371 (2011). https://doi.org/10.1007/s10601-011-9107-6
17. Lecoutre, C., Likitvivatanavong, C., Yap, R.H.C.: A path-optimal GAC algorithm for table constraints. In: Raedt, L.D., Bessiere, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., Lucas, P.J.F. (eds.) ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012. Frontiers in Artificial Intelligence and Applications, vol. 242, pp. 510–515. IOS Press (2012). https://doi.org/10.3233/978-1-61499-098-7-510
18. Lecoutre, C., Szymanek, R.: Generalized arc consistency for positive table constraints. In: Benhamou, F. (ed.) Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4204, pp. 284–298. Springer (2006). https://doi.org/10.1007/11889205_22
19. Leurent, G.: Analysis of differential attacks in ARX constructions. In: Wang, X., Sako, K. (eds.) Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7658, pp. 226–243. Springer (2012). https://doi.org/10.1007/978-3-642-34961-4_15
20. Libralesso, L., Delobel, F., Lafourcade, P., Solnon, C.: Automatic generation of declarative models for differential cryptanalysis. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021. LIPIcs, vol. 210, pp. 40:1–40:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.CP.2021.40
21. Mairy, J., Hentenryck, P.V., Deville, Y.: Optimal and efficient filtering algorithms for table constraints. Constraints An Int. J. **19**(1), 77–120 (2014). https://doi.org/10.1007/s10601-013-9156-0
22. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory

and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings. Lecture Notes in Computer Science, vol. 765, pp. 386–397. Springer (1993). https://doi.org/10.1007/3-540-48285-7_33, https://doi.org/10.1007/3-540-48285-7_33
23. Minier, M., Solnon, C., Reboul, J.: Solving a symmetric key cryptographic problem with constraint programming. In: ModRef 2014, Workshop of the CP 2014 Conference. p. 13 (2014)
24. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Bessiere, C. (ed.) Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4741, pp. 529–543. Springer (2007). https://doi.org/10.1007/978-3-540-74970-7_38, https://doi.org/10.1007/978-3-540-74970-7_38
25. Prud'homme, C., Fages, J.G.: Choco-solver: A java library for constraint programming. Journal of Open Source Software **7**(78), 4708 (2022). https://doi.org/10.21105/joss.04708, https://doi.org/10.21105/joss.04708
26. Ranea, A., Rijmen, V.: Characteristic automated search of cryptographic algorithms for distinguishing attacks (CASCADA). IET Inf. Secur. **16**(6), 470–481 (2022). https://doi.org/10.1049/ise2.12077
27. Rouquette, L., Gérault, D., Minier, M., Solnon, C.: And rijndael?: Automatic related-key differential analysis of rijndael. In: Batina, L., Daemen, J. (eds.) Progress in Cryptology - AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18-20, 2022, Proceedings. pp. 150–175. Lecture Notes in Computer Science, Springer Nature Switzerland (2022). https://doi.org/10.1007/978-3-031-17433-9_7
28. Rouquette, L., Solnon, C.: abstractxor: A global constraint dedicated to differential cryptanalysis. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12333, pp. 566–584. Springer (2020). https://doi.org/10.1007/978-3-030-58475-7_33
29. Sakamoto, K., Minematsu, K., Shibata, N., Shigeri, M., Kubo, H., Funabiki, Y., Isobe, T.: Security of related-key differential attacks on twine, revisited. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **103-A**(1), 212–214 (2020). https://doi.org/10.1587/transfun.2019CIL0004, http://search.ieice.org/bin/summary.php?id=e103-a_1_212
30. Sun, S., Gérault, D., Lafourcade, P., Yang, Q., Todo, Y., Qiao, K., Hu, L.: Analysis of AES, SKINNY, and Others with Constraint Programming. IACR Trans. Symmetric Cryptol. **2017**(1), 281–306 (2017)
31. Teh, J.S., Biryukov, A.: Differential cryptanalysis of WARP. J. Inf. Secur. Appl. **70**, 103316 (2022). https://doi.org/10.1016/j.jisa.2022.103316
32. Udovenko, A.: MILP modeling of boolean functions by minimum number of inequalities. IACR Cryptol. ePrint Arch. p. 1099 (2021), https://eprint.iacr.org/2021/1099
33. Ullmann, J.R.: Partition search for non-binary constraint satisfaction. Inf. Sci. **177**(18), 3639–3678 (2007). https://doi.org/10.1016/j.ins.2007.03.030