Differential cryptanalysis with SAT, SMT, MILP, and CP: a detailed comparison for bit-oriented primitives

Emanuele Bellini[2][0000−0002−2349−0247], Alessandro De Piccoli[1][0000−0002−6399−3164], Mattia Formenti[2][0009−0001−0069−6146], David Gerault[2][0000−0001−8583−0668], Paul Huynh[2][0000−0002−6965−3427], Simone Pelizzola[1][0009−0006−3991−1161], Sergio Polese, and Andrea Visconti[1][0000−0001−5689−8575]

[1] Università degli Studi di Milano, Milano, Italia
{name.lastname}@unimi.it
[2] Technology Innovation Institute, Abu Dhabi, UAE
{name.lastname}@tii.ae

Abstract. SAT, SMT, MILP, and CP, have become prominent in the differential cryptanalysis of cryptographic primitives. In this paper, we review the techniques for constructing differential characteristic search models in these four formalisms. Additionally, we perform a systematic comparison encompassing over 20 cryptographic primitives and 16 solvers, on both easy and hard instances of optimisation, enumeration and differential probability estimation problems.

Keywords: Differential cryptanalysis · SAT · SMT · MILP · CP.

1 Introduction

The design and analysis of block ciphers is a time-consuming and error-prone task that involves tracing the propagation of bit-level or word-level patterns of all sorts, following intricate rules. Automatic tools have made such tasks significantly easier. In the case of differential cryptanalysis [9], one of the most widely used analysis technique, the studied patterns (differential characteristics) represent the propagation of a XOR difference between the inputs through the cipher, and are studied through the following methods: (1) *ad hoc* (include search algorithms implemented from scratch in general purpose programming languages, e.g. Matsui algorithm [34]); (2) *Boolean Satisfiability* and *Satisfiability Modulo Theory* (SAT/SMT); (3) *Mixed-Integer Linear Programming* (MILP); (4) *Constraint Programming* (CP). In this paper, we provide an extensive review and performance comparison for the last three techniques for the search of differential characteristics for various ciphers.

Contributions Our contributions are twofold:

- We provide an extensive review of modeling techniques in SAT, SMT, MILP and CP for the search of differential characteristics, in Section 3;
- We extensively compare these 4 methods on 3 different tasks: finding one optimal differential characteristic, enumerating all optimal differential characteristics, and estimating the probability of a differential. These tests are performed with 7 SAT solvers, 3 SMT solvers, 2 MILP solvers, 4 CP solvers, on over 20 primitives, resulting in the largest scale comparison of differential cryptanalysis models to date. The results are presented in Section 4.

The research community stands to benefit greatly from this extensive review and comparison of techniques, which provides a further steps towards a better understanding of how to solve the instances that are still out of reach.

2 Preliminaries

A symmetric cryptographic primitive is usually a sequence of linear and nonlinear *components* transforming a plaintext (possibly with a key) into a ciphertext, usually by applying a simple *round function* to update the state for a number of *rounds*, each round using a *round key* derived from a *key schedule* algorithm.

Differential cryptanalysis focuses on studying the probability of *differentials*, which map an XOR difference in the plaintexts to a differences in the ciphertexts. This probability is usually bounded by the probability of a *differential characteristics*, *i.e.*, a sequence of expected differences at each round (as described in Section A); the probability of the corresponding differential is related to the combined probabilities of all differential characteristics sharing the corresponding input and output differences, but varying in the internal rounds. Finding the optimal (highest probability) differential characteristic, or enumerating differential characteristics with given properties, is a highly combinatorial problem. In recent years, it has increasingly been tackled through declarative approaches (Section B), where the cryptographer describes the problem and leaves its resolution to a solver, usually SAT, SMT (Satisfiability Modulo Theories), MILP (Mixed Integer Linear Programming) and CP (Constraint Programming).

The search typically involves one set of variables per round to hold the difference state after each component of the primitive, as well as a set of variable for the probabilities. These variables usually contain the weights (base 2 logarithm of the reciprocal of the probabilities) for practical reasons. The problem of finding an optimal differential characteristic can then be expressed as assigning values for all state variables, such that known difference propagation rules are satisfied, and the sum of the probability weights is minimised, following the Markov cipher assumption of independent rounds.

The representation of these variables, and the expression of the propagation rules, vary between SAT, SMT, MILP and CP.

The propagation rules for linear components are simple, as differences propagate deterministically through them:

Proposition 1. *Let $f : \{0,1\}^m \to \{0,1\}^n$ be a linear function and let $\Delta\vec{x} \in \{0,1\}^m$ be an input difference; then $\Delta\vec{y} = f(\Delta\vec{x})$. (Proof: $f(\vec{x} + \Delta\vec{x}) = f(\vec{x}) + f(\Delta\vec{x})$)*

On the other hand, propagation through non-linear operations are stochastic, and represent the main difficulty of the problem, due to the resulting combinatorial explosion. In Section 3, we detail the models used for propagation of the linear and nonlinear components used by the analyzed ciphers.

Related Work Differential cryptanalysis using declarative frameworks (SAT, SMT, MILP or CP) was introduced through MILP in [37], and has since then been an active research field (a review of techniques is given in Section 3). It is known [53] that the modeling choices for the search problem, independently of the chosen declarative framework, have a significant impact on the performance of the search. Additionally, within a given framework, it is difficult to predict what specific solver performs best: competitions such as the SAT competition [27] or the MiniZinc challenge for Constraint Programming solvers [36] pit existing solvers against each other on vast ranges of problems, but rarely cryptography-related ones. The choice of a model and a solver having such drastic impact on the ability to solve relevant differential cryptanalysis problems, research comparing the available options is important.

In [51] and [24], the authors use Constraint Programming tools to test the effectiveness of four solvers on PRESENT and AES, showing that for best differential characteristic search Chuffed is the best-performing solver on small instances, while Gurobi and Picat-SAT scale better. In [50], different SAT solvers are compared against a divide-and-conquer-based MILP model from [54] on a wide range of ciphers. In [18], the authors compare different models for the search of the best differential trails of SKINNY, including one for MILP, one for SAT, and one for CP. Following a two-stage search, their analysis showed that, in this case, this search is better performed with a MILP model in the first stage (enumerate the truncated trails with the smallest number of active S-box). CP performed best for the second stage, in which the truncated trails of the first stage are instantiated.

Despite extensive research in the area, many problems, such as the probabilities of differential characteristics for over 9 rounds of SPECK128 [50], are still out of reach. It is our hope that our large-scale comparison between solvers and modeling techniques will help chosing the right techniques to solving these.

3 Cipher components models

In this section, we review existing techniques to model different operations, in each of the studied declarative frameworks.

We use the following notation: x denotes inputs, y outputs and w weight; superscripts denote input numbers and subscripts bit positions. If no input number is given, the input is only one; if no bit position is given, the variable is intended to be a single bit. Finally, we will use the vector notation $\vec{x} = (x_0, \ldots, x_{n-1})$ to

denote the whole input, using 0 as the index of the Most Significant Bit (MSB). The models described in this section are *bit-based*, rather than *word-based*.

3.1 XOR component

XOR is a linear function and Proposition 1 applies, so that we can directly apply the bitwise model $\Delta y = \bigoplus_{i=0}^{n-1} \Delta x^i$.

- **SAT:** for $n = 2$, the CNF is

$$(\neg\Delta x^0, \Delta x^1, \Delta y)\wedge(\Delta x^0, \neg\Delta x^1, \Delta y)\wedge(\Delta x^0, \Delta x^1, \neg\Delta y)\wedge(\neg\Delta x^0, \neg\Delta x^1, \neg\Delta y). \tag{1}$$

  When $n > 2$, one can operate in the following two ways: the first consists of the direct encoding without any additional variables; the second consists of performing a sequence of only two inputs XORs using intermediate variables that we will call d^i in the following way:

$$d^0 = \Delta x^0 \oplus \Delta x^1, \quad d^i = \Delta x^{i+1} \oplus d^{i-1} \text{ for } 1 \leq i \leq n-3, \quad \Delta y = \Delta x^{n-1} \oplus d^{n-3}. \tag{2}$$

  Note that the CNF in Equation 1 represents every possible assignment verifying $\Delta y = \Delta x^0 \oplus \Delta x^1$. Therefore, a direct encoding of an XOR involving n variables will have 2^n clauses. In our analysis, when $n > 2$, we have preferred to use a sequential XOR, as depicted in Equation 2, keeping the number of clauses linear in the number of variables, i.e. $4(n-1)$ clauses [50].
- **SMT:** a XOR theory is natively present for $n = 2$ or more.
- **MILP:** 2-input XOR is commonly modeled with four inequalities:

$$\{\Delta x^0 + \Delta x^1 \geq \Delta y\},\ \{\Delta x^0 + \Delta y \geq \Delta x^1\},\ \{\Delta x^1 + \Delta y \geq \Delta x^0\}, \{\Delta x^0 + \Delta x^1 + \Delta y \leq 2\}. \tag{3}$$

  We also considered an alternative, with a dummy variable, which can easily be generalized to any arbitrary number of inputs:

$$\{\Delta x^0 + \cdots + \Delta x^{n-1} + \Delta y = 2d\} \tag{4}$$

  While this results in a smaller and constant number of inequalities, the *LP-relaxation* of the resulting problem—that is, the same optimization problem without integrality constraint on the variables—is weaker than the one obtained with Equation 3. Indeed, any fractional solution of Equation 3 is also a solution of Equation 4. However, the converse is not true. For instance, for $n = 2$, $\Delta x^0 = \Delta x^1 = \frac{1}{5}, \Delta y = \frac{1}{2}$ is a solution for Equation 4 when $d = \frac{9}{20}$ but does not satisfy Equation 3. For this reason, we favored Equation 3 over the more concise expression of Equation 4. This was also backed by our experiments Midori64, whose linear layer contains several n-XORs: even though both expressions seemed to yield similar performance for 2 and 3 rounds, a difference started to be noticeable for 4 rounds as the search for the optimal trail with Gurobi took less than 2 minutes using Equation 3, while it took more than 30 minutes with Equation 4.

– **CP:** the XOR can be seen as the addition modulo 2, i.e. $\Delta y = \Delta x^0 + \Delta x^1 \pmod 2$. The same can be applied when dealing with more than 2 inputs:

$$\Delta y = \Delta x^0 + \Delta x^1 + \ldots + \Delta x^{n-1} \pmod 2.$$

3.2 Rotation and shift components

Rotation and shift are linear functions to which Proposition 1 directly applies.

– **SAT:** an equality can be translated in an if-and-only-if logic, so, the model that we have used is $(\Delta y_i \vee \neg f(\Delta x_i)) \wedge (\neg \Delta y_i \vee f(\Delta x_i))$.
– **MILP:** the equality is expressed as two inequalities: $\{\Delta y_i \geq f(\Delta x_i),\ \Delta y_i \leq f(\Delta x_i)\}$.
– **SMT, CP:** both formalisms natively include equality constraints.

3.3 Linear layer component

For the linear layer, Proposition 1 directly applies. Considering the linear function f represented as a vector-matrix product, the linear layer is simply a set of equalities of the form $\Delta y = \Delta x^0 \oplus \Delta x^1 \oplus \ldots \oplus \Delta x^{n-1}$.

If $n = 1$, then, we have no XOR and we can directly encode the equality. If $n \geq 2$, we refer to the XOR component for encoding the equality.

3.4 S-box component

An S-box is a nonlinear vectorial Boolean function that transforms an m-bit input into an n-bit output. Commonly, $m = n$ and usual values for n are up to 8. For instance, we take the 3-bit S-box defined as $S = (S_0, S_1, \ldots, S_7) = (3, 2, 7, 0, 4, 1, 6, 5)$, meaning that $S(i) = S_i$.

In order to study the differential of the S-box, it is usually affordable to consider its Difference Distribution Table (DDT). We start from a $m \times n$ table filled with zeros and for each input pair (i, j), we compute $\Delta\vec{x} = i \oplus j$ and $\Delta\vec{y} = S_i \oplus S_j$ and increase the $(\Delta\vec{x}, \Delta\vec{y})$ entry by one. Our SAT, SMT and MILP models also operate on other tables related to the DDT:

– $*$-DDT, using the same notation of [1], a truncated DDT, in which all the non-zero entries of the DDT are replaced by 1.
– w-DDT, which contains the weights[3] of the probability of the $(\Delta\vec{x}, \Delta\vec{y})$ entry.

Considering the previous 3-bit S-box S, we show its DDT in Table 1a and the associated w-DDT and $*$-DDT in Table 1b and Table 1c respectively.

[3] It should be noted that the entries of this table are not always integers, as a DDT might contain entries that are not powers of 2.

Table 1: DDT of the S-box $S = (3, 2, 7, 0, 4, 1, 6, 5)$ and its associated tables.

(a) DDT

$\Delta\vec{x}$ \ $\Delta\vec{y}$	0	1	2	3	4	5	6	7
0	8	0	0	0	0	0	0	0
1	0	2	0	2	0	2	0	2
2	0	0	4	0	4	0	0	0
3	0	2	0	2	0	2	0	2
4	0	2	0	2	0	2	0	2
5	0	0	4	0	0	0	4	0
6	0	2	0	2	0	2	0	2
7	0	0	0	0	4	0	4	0

(b) w-DDT

$\Delta\vec{x}$ \ $\Delta\vec{y}$	0	1	2	3	4	5	6	7
0	0	·	·	·	·	·	·	·
1	·	2	·	2	·	2	·	2
2	·	·	1	·	1	·	·	·
3	·	2	·	2	·	2	·	2
4	·	2	·	2	·	2	·	2
5	·	·	1	·	·	·	1	·
6	·	2	·	2	·	2	·	2
7	·	·	·	·	1	·	1	·

(c) $*$-DDT

$\Delta\vec{x}$ \ $\Delta\vec{y}$	0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1
2	0	0	1	0	1	0	0	0
3	0	1	0	1	0	1	0	1
4	0	1	0	1	0	1	0	1
5	0	0	1	0	0	0	1	0
6	0	1	0	1	0	1	0	1
7	0	0	0	0	1	0	1	0

– **SAT:** we will refer to the S-box presented above for concrete examples, thus, in the following, we will use the bit representation of values, i.e. $\Delta\vec{x} = (\Delta x_0, \Delta x_1, \Delta x_2)$, $\Delta\vec{y} = (\Delta y_0, \Delta y_1, \Delta y_2)$ and $\vec{w} = (w_0, w_1)$. The value for the *weight* has only two bits since from Table 1b, it is clear that the maximum weight $w_{\max}$ here is 2, so two bits will be enough to represent the weight. Generally speaking, we need $\lceil \log_2(w_{\max}) \rceil$ bits to encode the weight.
Ankele and Kölbl presented a method to compute the CNF representing the w-DDT of an S-box [3]. Basically, they compute the $*$-DDT and, for every $(\Delta\vec{x}, \Delta\vec{y})$ having the relative entry equal to 0, they encode the constraint

$$\neg(\Delta\vec{x} \wedge \Delta\vec{y} \wedge w) \quad \Rightarrow \quad \neg\Delta\vec{x} \vee \neg\Delta\vec{y} \vee \neg w$$

for every possible weight. For instance, for the pair (2, 3) in w-DDT, we use

$$\neg(\neg\Delta x_0 \wedge \Delta x_1 \wedge \neg\Delta x_2 \wedge \neg\Delta y_0 \wedge \Delta y_1 \wedge \Delta y_2 \wedge w_0 \wedge w_1)$$

to avoid the triplet $(\Delta\vec{x}, \Delta\vec{y}, \vec{w}) = (2, 3, 2)$. The procedure must be repeated for every triplet that is not present in Table 1b. Summing up, we can say that they build the complementary set of the possible triplets shown in Table 1b. For a high number of cipher rounds, this method results in a number of constraints, i.e. clauses, which is not handy for SAT solvers.
In order to reduce the number of constraints, we model the w-DDT as a sum of products. In this way, we directly encode only all allowed triplets. For instance, considering the triplet (2, 4, 1) in w-DDT, we use as a model

$$\begin{aligned}(&\neg\Delta x_0 \wedge \Delta x_1 \wedge \neg\Delta x_2 \wedge \Delta y_0 \wedge \neg\Delta y_1 \wedge \neg\Delta y_2 \wedge \neg w_0 \wedge w_1) \\ &\vee (\neg\Delta x_0 \wedge \Delta x_1 \wedge \neg\Delta x_2 \wedge \neg\Delta y_0 \wedge \Delta y_1 \wedge \Delta y_2 \wedge w_0 \wedge \neg w_1)\end{aligned}$$

Clearly, a SAT solver can not handle a sum of products. Therefore we have used the heuristic *Espresso* algorithm [11] in order to reduce it to a product-of-sum, i.e. a CNF. As already pointed out in [3], this technique is only applicable to DDTs containing entries that are powers of 2.

– **SMT:** we use the same model presented for SAT.
– **MILP:** The bitwise modeling of a differential propagation through an S-box of size greater than 6 bits remained a hard problem until the work of Abdelkhalek *et al.* was published [1]. Their approach relies on logical condition modeling, already introduced by Sun *et al.* [52], and uses the product-of-sums representation of the indicator function of the $*$-DDT, as in SAT and SMT. Taking the example again from Table 1a, let f be the 6-bit to 1-bit boolean function associated with the $*$-DDT shown in Table 1c. That is, $f(\Delta\vec{x}, \Delta\vec{y}) = 1$ only if the propagation is possible, where $\Delta\vec{x} = (\Delta x_0, \ldots, \Delta x_{n-1})$ and $\Delta\vec{y} = (\Delta y_0, \ldots, \Delta y_{n-1})$ denote the input and output difference, respectively. The product-of-sums representation of f is as follows:

$$f(\Delta\vec{x}, \Delta\vec{y}) = (\Delta x_0 \vee \Delta x_1 \vee \Delta x_2 \vee \Delta y_0 \vee \Delta y_1 \vee \overline{\Delta y_2}) \\ \wedge \cdots \wedge (\overline{\Delta x_0} \vee \overline{\Delta x_1} \vee \overline{\Delta x_2} \vee \overline{\Delta y_0} \vee \overline{\Delta y_1} \vee \overline{\Delta y_2}),$$

where $\overline{\Delta a}$ is the negation of Δa. Each term of the product represents one impossible transition in the $*$-DDT. For instance, the first term $(\Delta x_0 \vee \Delta x_1 \vee \Delta x_2 \vee \Delta y_0 \vee \Delta y_1 \vee \overline{\Delta y_2})$ corresponds to the impossible propagation `0x0` $\rightarrow$ `0x1`. This means that the number of terms corresponds to the number of null entries in the $*$-DDT, which can be rather high for an 8-bit S-box. For this reason, finding a minimal, equivalent set of inequalities is a crucial step in the modeling of large S-boxes. Several algorithms have been described for the Boolean function minimization problem, such as the Quine-McCluskey algorithm [44,45,35] or the heuristic *Espresso* algorithm, already mentioned for SAT. Once a simplified product-of-sum is returned, each term can be rewritten as a linear inequality. For instance, $(\Delta x_0 \vee \Delta x_1 \vee \Delta x_2 \vee \Delta y_0 \vee \Delta y_1 \vee \overline{\Delta y_2}) = 1$ becomes:

$$\Delta x_0 + \Delta x_1 + \Delta x_2 + \Delta y_0 + \Delta y_1 + (1 - \Delta y_2) \geq 1.$$

After removing all impossible propagation for a given $*$-DDT table, the actual probabilities of the differential transitions of the S-box need to be taken into account. To do so, [1] proposed to separate the $*$-DDT into multiple w_k-DDT tables, such that w_k-DDT only contains entries with the same weight w_k, that is: w_k-DDT$[i, j] = 1$ if w-DDT$[i, j] = w_k$ and 0 otherwise. The use of *indicator constraints* (such as the *big-M method*) ensures that only a single w_k-DDT is active:
  - for each S-box, we introduce a binary variable Q equal to 1 if the S-box is active, 0 otherwise;
  - similarly, for each w_k-DDT, a binary variable Q_{w_k} that equals 1 when the set of inequalities representing the w_k-DDT need to be effective.

  Setting $\sum Q_{w_k} = Q$ ensures that whenever an S-box is active, only one w_k-DDT is effective; and the weight of the S-box can be modeled as $\sum w_k \cdot Q_{w_k}$.
– **CP:** *table constraints* allow for a straightforward representation of the S-box component. Indeed, they enforce a tuple of variables to take its value among a list of allowed tuples, explicitly defined as the rows of a table. In particular,

each row will contain the following three elements concatenated: an input difference, an output difference, and the weight of the probability for the input/output difference pair. In our bitwise representation, the input and output differences are the concatenations of m and n single-bit variables, respectively. An entry of the table is thus a $m + n + 1$ tuple.

Remark 1. We highlight that the S-box constraints represent a considerable amount of the constraints in SAT, SMT and MILP formalisms. In fact, the PRESENT S-box (4 bits) constraints are roughly one-half of the total constraints.

3.5 AND/OR component

As the AND and OR are bitwise operations, one can easily build their DDTs. Indeed, they can be seen as 2-to-1 S-boxes repeated in parallel for as many times as the bit length of the inputs. This is equivalent to the approach explained in [2, Section 3].

- **SAT:** we reuse the techniques described in Subsection 3.4 obtaining:

$$(\neg \Delta y \vee w) \wedge (\Delta x^0 \vee \Delta x^1 \vee \neg w) \wedge (\neg \Delta x^0 \vee w) \wedge (\neg \Delta x^1 \vee w).$$

- **SMT:** since satisfying a sum-of-products is easier than satisfying a product-of-sum, we encoded the AND component with the following model for a single bit:

$$(\neg \Delta x^0 \wedge \neg \Delta x^1 \wedge \neg \Delta y \wedge \neg w) \vee (\Delta x^0 \wedge w) \vee (\Delta x^1 \wedge w).$$

- **MILP, CP:** we reuse the techniques described in Subsection 3.4 to model its DDT.

3.6 Modular addition component

Due to the intractable size of the DDT, even if using wordsize equal to 32 bits, the method adopted for the modular addition is the Lipmaa Moriai algorithm [31], based on two conditions:

1. $\mathsf{eq}(\Delta\vec{x}^0 \ll 1, \Delta\vec{x}^1 \ll 1, \Delta\vec{y} \ll 1) \wedge (\Delta\vec{x}^0 \oplus \Delta\vec{x}^1 \oplus \Delta\vec{y} \oplus (\Delta\vec{x}^1 \ll 1)) \neq 0$
2. $2^{-\mathsf{hw}(\neg\,\mathsf{eq}(\Delta\vec{x}^0, \Delta\vec{x}^1, \Delta\vec{y}) \wedge \mathsf{mask}(n-1))}$

with $\mathsf{eq}(x, y, z) := (\neg x \oplus y) \wedge (\neg x \oplus z)$, that is, $\mathsf{eq}(x, y, z) = 1 \Leftrightarrow x = y = z$, and for any n, $\mathsf{mask}(n) := 2^n - 1$.

- **SAT:** first of all, observe that $\mathsf{eq}(\Delta x_i^0, \Delta x_i^1, \Delta y_i)$ for $1 \leq i \leq n-1$ is used in both conditions, therefore, using w for the Hamming weight variable, we model

$$w_i = \neg\,\mathsf{eq}(\Delta x_i^0, \Delta x_i^1, \Delta y_i) \tag{5}$$

using the following CNF

$$(\Delta x_i^0 \vee \neg \Delta y_i \vee w_i) \wedge (\Delta x_i^1 \vee \neg \Delta x_i^0 \vee w_i) \wedge (\Delta y_i \vee \neg \Delta x_i^1 \vee w_i) \\ \wedge (\Delta x_i^0 \vee \Delta x_i^1 \vee \Delta y_i \vee \neg w_i) \wedge (\neg \Delta x_i^0 \vee \neg \Delta x_i^1 \vee \neg \Delta y_i \vee \neg w_i)$$

which is exhaustive for the second condition. By only considering the Least Significant Bit, the first condition can be encoded as

$$\Delta x_{n-1}^0 \oplus \Delta x_{n-1}^1 \oplus \Delta y_{n-1} = 0 \quad \Rightarrow \quad \Delta y_{n-1} = \Delta x_{n-1}^0 \oplus \Delta x_{n-1}^1 \tag{6}$$

for which we refer to the XOR component. Finally, taking the advantage of Equation 5 and using a dummy variable, for $0 \leq i \leq n-2$, we need

$$(\neg w_i \wedge (d_i \oplus \Delta x_i^1) = 0) \wedge (d_i = \Delta x_{i+1}^0 \oplus \Delta x_{i+1}^1 \oplus \Delta y_{i+1}) \tag{7}$$

which turns into the following CNF

$$(\Delta x_i^1 \vee \neg d \vee w) \wedge (\neg \Delta x_i^1 \vee d \vee w) \\ \wedge (\Delta x_{i+1}^0 \vee \Delta x_{i+1}^1 \vee d \vee \neg \Delta y_{i+1}) \wedge (\Delta x_{i+1}^0 \vee \Delta x_{i+1}^1 \vee \neg d \vee \Delta y_{i+1}) \\ \wedge (\Delta x_{i+1}^0 \vee \neg \Delta x_{i+1}^1 \vee d \vee \Delta y_{i+1}) \wedge (\neg \Delta x_{i+1}^0 \vee \Delta x_{i+1}^1 \vee d \vee \Delta y_{i+1}) \\ \wedge (\Delta x_{i+1}^0 \vee \neg \Delta x_{i+1}^1 \vee \neg d \vee \neg \Delta y_{i+1}) \wedge (\neg \Delta x_{i+1}^0 \vee \Delta x_{i+1}^1 \vee \neg d \vee \neg \Delta y_{i+1}) \\ \wedge (\neg \Delta x_{i+1}^0 \vee \neg \Delta x_{i+1}^1 \vee d \vee \neg \Delta y_{i+1}) \wedge (\neg \Delta x_{i+1}^0 \vee \neg \Delta x_{i+1}^1 \vee \neg d \vee \Delta y_{i+1})$$

Note that this is a different approach from the one in [50]. Indeed, although our model has two more clauses in comparison, the number of variables per clause is reduced and can thus speed up the SAT solving process.

- **SMT:** since SMT has more expressive capability, we have encoded a bitwise model in a similar way to SAT. We simply report the implementation details:
  - we have used $\neg w_i = (\Delta x_i^0 = \Delta x_i^1 = \Delta y_i)$ instead of Equation 5;
  - we have directly used $\Delta y_i \oplus \Delta x_i^0 \oplus \Delta x_i^1 = 0$ in Equation 6;
  - we have used $w_i \vee \neg(\Delta x_{i+1}^0 \oplus \Delta x_{i+1}^1 \oplus \Delta y_{i+1} \oplus \Delta x_i^1)$ instead of Equation 7.
- **MILP:** implementing the Lipmaa-Moriai as is in MILP would be rather inefficient, as expressing simple if-then-else statements requires extra variables and constraints. Instead, it is possible to directly derive a small set of linear constraints by listing all valid patterns for $(\Delta \vec{x}^0, \Delta \vec{x}^1, \Delta \vec{y}, \Delta \vec{x}^0 \ll 1, \Delta \vec{x}^1 \ll 1, \Delta \vec{y} \ll 1)$ that satisfy the conditions imposed by the Lipmaa-Moriai algorithm, as done by Fu *et al.* [21]. In their paper, the authors obtained 65 linear inequalities for each bit. This set of constraints was then reduced by using a greedy algorithm or the Espresso minimizer. As such, the differential behavior of addition modulo 2^n could be represented using $13(n-1)+5$ linear inequalities in total.
- **CP:** in the CP model, the constraints for modular addition involve the preliminary step of declaring three shifted arrays representing the carry (the shifts in the first condition) and an additional array `eq` with the results of the `eq` function. The constraint is then a straightforward implementation of the Lipmaa-Moriai algorithm. The `eq` function is easily defined thanks to the `all_equal()` global constraint. Then, the output difference constraints are derived from the first condition:

- if the `eq` constraint is satisfied, then the difference propagation is deterministic and its constraint is given by the second part of the condition, i.e. $\Delta\vec{x}^0 \oplus \Delta\vec{x}^1 \oplus \Delta\vec{y} \oplus (\Delta\vec{x}^1 \ll 1) = 0$. In other words, the output difference is the XOR of the inputs and carry differences;
- otherwise, no more constraints are needed, and the transition will have weight 1. The weight variable is constrained to be $n - sum(\texttt{eq})$.

4 Experimental results

In this section, we present a comparison of formalisms and solvers for differential cryptanalysis problems. In particular, we examine the 3 following tasks:

1. **Task 1** the search for an optimal differential trail (easy and difficult instances);
2. **Task 2** the enumeration of all optimal trails;
3. **Task 3** the estimation of the probability of a differential.

For these three tasks we will present the results we obtained on different ciphers, based on the data available in literature and how accurately the corresponding graph would present the experimental comparison between the best solvers for each formalism.

It has been observed in previous works, such as [17], that the fastest solver on small instances does not always scale up to more difficult instances of the same problem; therefore, we study both cases for the search of an optimal trail.

In the first two cases, no constraints are imposed on the input and output; in the third case, the weight, or objective function, is fixed to the optimal value; in the last case, the input and output differences are fixed, and all trails with a probability greater than a fixed lower bound are enumerated.

Optimization is natively supported for CP and MILP, whereas increasing objective values are tested until the problem is satisfiable for SAT and SMT. The enumeration of solutions is performed natively in CP, by adding constraints forbidding each new solution after it is found for the other formalisms.
All tests were run on a server with the following configuration, on which no more than half the threads were used at any given time:

- `CPU`: 2 x Intel(R) Xeon(R) Gold 6258R;
- `Number of Cores/Threads`: 2 x 28 Cores/2 x 56 Threads
- `Base/Max CPU frequency achievable`: 2.7 GHz / 4.0 GHz
- `Cache`: 38.5 Mb
- `Memory`: 768GB @2933 MHz;
- `Operating System`: Ubuntu 18.04.5 LTS.

In this framework, many algorithms are taken into account, considering block ciphers, stream ciphers and hash functions. In particular, the following families of ciphers have been analyzed:

- Block ciphers: Simon and Speck, Threefish, LEA, DES, Midori, PRESENT, TEA, XTEA;
- Permutations: Gift, Gimli, Keccak, Ascon, ChaCha, Xoodoo,
- Hash functions: SHA1, SHA-224, SHA-256, SHA-384, SHA-512, Blake, Blake2, MD5.

For each cipher, we tested several rounds. We did not use results found in smaller rounds for the higher round case.

4.1 Choice of solvers

In our testing activities, we not only compare formalisms but also try to identify which solver performs best for a given formalism and a given problem. Below is the list of solvers we used for each formalism.

- **SAT Solvers**: CaDiCal (1.5.3) [7]; CryptoMiniSat (5.11.4) [49]; Glucose Syrup (4.1) [4]; Kissat (3.0.0) [7]; MathSAT (5.6.9) [13]; Minisat (2.2.1) [20]; Yices2 (2.6.4) [19]. All solvers were run with their default parameters and options.
- **SMT Solvers**: MathSAT (5.6.9) [13]; Yices2 (2.6.4) [19]; Z3 (4.8.12) [38]. All solvers were run with their default parameters and options. Note that the SMT models developed in Section 3 need the `QF_UF` logic in SMT-LIB standard, therefore we excluded Boolector [40] and STP [22].
- **MILP Solvers**: GLPK [41], Gurobi [26]. SCIP [6] was considered, but since our MILP models were written using the solver interfaces provided by the `SageMath` MILP module, which do not include SCIP, it was not included.
- **CP Solvers**: Chuffed [12], Gecode [47], OR-tools [25], Choco [43]. Our model are written in the MiniZinc [39] language, which interfaces to these solvers.

4.2 Comparison for Task 1

The first problem of this comparison is that of the optimal objective value (and a satisfying trail).

We considered representatives of block ciphers, permutations and hash functions and fixed the number of rounds with two different ideas in mind: we wanted to compare the performances of the different formalisms and solvers on easier problems, obtained by considering instances of various ciphers on a low number of rounds (2 to 6). To make our results meaningful we set a minimum time threshold of 2 seconds: if any solver is able to finish the 6-round instance in less than that, we repeat the test for a higher number of rounds, until this threshold is crossed. These will be called *quick tests*. In addition, we ran a comparison on *slow tests*, composed of more difficult instances of Simon, Speck, and PRESENT.

For each test we measured the *solving time* (time to solve the model) and the *building time* (time to build the model). The sum of building and solving time will be referred to as the *combined time*.

Quick Tests In this section, we present a comparison of solvers on easy cryptographic instances for all the primitives mentioned in Section 4. The solver with the lowest combined time for a given instance is awarded a *win*. The best solver for each cipher is the one with the highest number of wins. The winner of our *competition* (for every formalism) is the solver that performs best for the highest number of ciphers (more than 20, each from round 2 to 6).

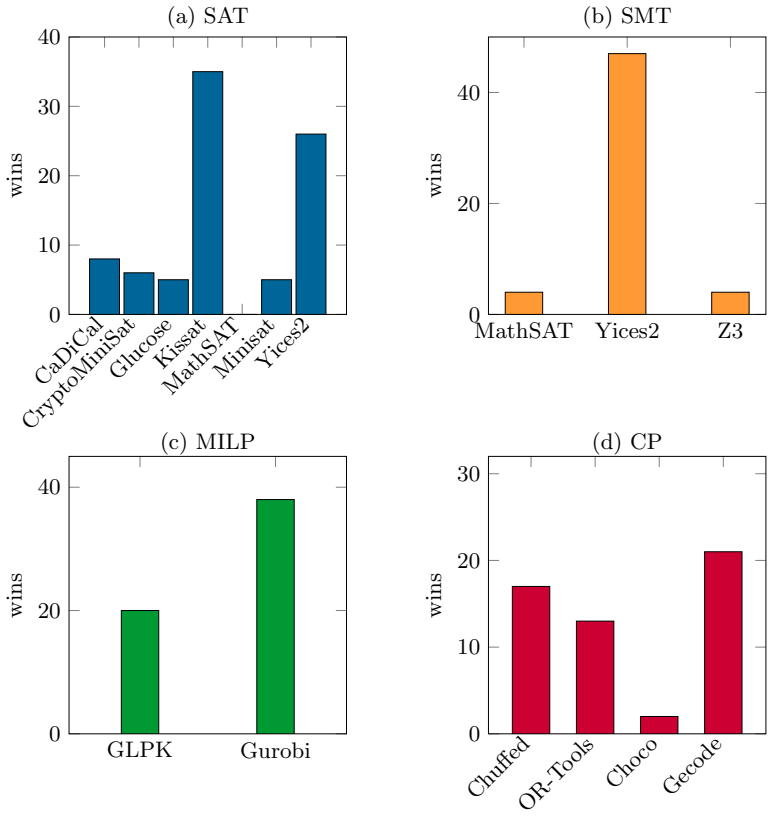


Fig. 1: Comparison of the number of victories of each solver, per formalism, on the set of easy instances.

The graphs in Figure 1 report the results of these competitions:

- Among SAT solvers, Kissat and Yices2 emerge as the clear winners. It should also be noted that the timings reported from Glucose are computed taking multithreading into account, and thus do not faithfully represent the real time needed to obtain the results;
- In the SMT solvers category, Z3 and MathSAT are always inferior to Yices2, which is thus clearly the best SMT solver in our testing;

– In CP and MILP, the difference between different solvers is not as clear cut: while Gecode and Gurobi are the fastest solvers overall, Chuffed and GLPK often manage to be at least equal to them in their respective models.

In Figure 2, we present the results of the quick tests for Simon32, Speck32, PRESENT, Gimli, and BLAKE[4], for the best solver of each formalism we found before. These tests were run with a timeout of 10 minutes, which was extended by another 10 minutes if no solver returned within the first time slot. We refer to Section C for the exhaustive list of timings. In all these cases, SAT consistently appears as the superior option.

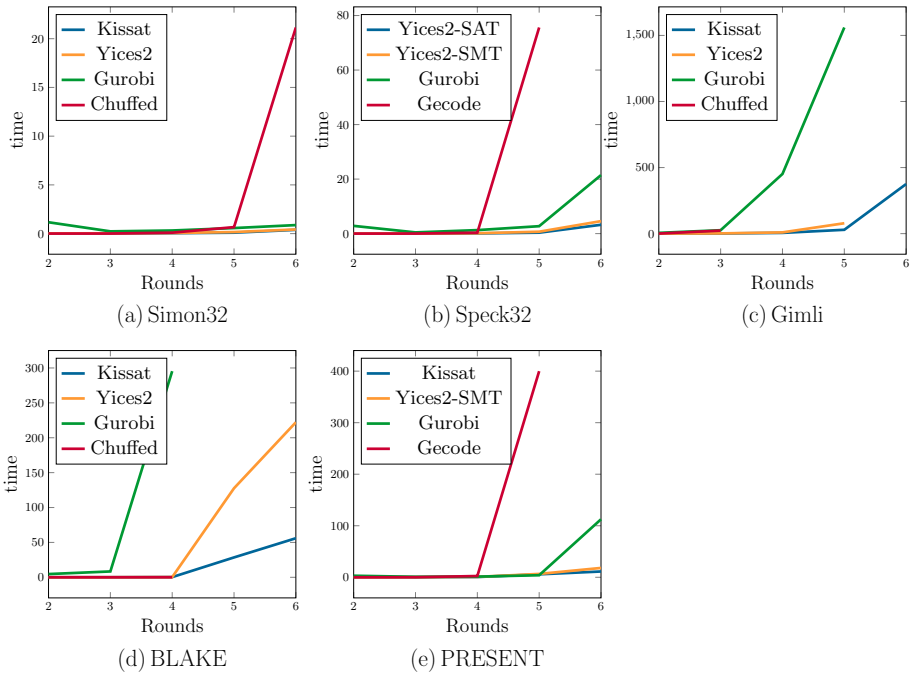


Fig. 2: Graph comparisons between the best solvers for each formalism on different ciphers testing the function `find_lowest_weight_trail`.

Figure 2a and Figure 2b show very similar performances between SAT and SMT for Simon and Speck; the detailed times are given in Table 5 and Table 6 of Section C. On the other hand, SAT dominated on a primitive with a larger state, Gimli, as shown in Figure 2c and Figure 2d: SAT is the only formalism to complete the 6-round test within the 10 minutes time limit.

[4] As an example, we selected, respectively, three small state block ciphers, (one AndRX, one ARX, one S-Box based), one large state permutation (384 bits) and one large state ARX hash (512 bits)

Slow tests In this section, we run a comparison on longer instances, described in Table 2, with a timeout of 24 hours.

Table 2: The instances of our long tests set; optimal weight for a fixed number of rounds is found and compared to known results for correctness.

Cipher	Rounds	Weight	Reference
PRESENT	18	78	[50]
Simon32	12	34	[33]
Simon64	19	64	[33]
Speck32	9	30	[5]
Speck64	13	55	[5]

The results are reported in Table 3; solving and building time are expressed in seconds, while the memory used is in megabytes. In the table, *inf* is reported when the solver does not provide a reliable way to measure its memory usage.

These tests were ran for all paradigms, but the solvers that returned within the 24 hours timeout were mostly SAT, showing a clear advantage on this problem; MILP only finished within the timeout once (and came out on top) for SIMON32. We ran all tests with the best current known techniques for each for each formalism, except for MILP for which we use techniques from [1], even though we are aware of the improvements from [10,30] and plan to add them in the future. Chances are that the improvements from [10,30] will yield better performances for MILP solvers.

For 9 rounds of SPECK32, the known best trail was retrieved, but only SAT and SMT solvers finished within the time limit. For PRESENT and SPECK64, only SAT solvers finished within the time limit, with a clear advantage for Kissat.

These results contrast with the quick tests: Yices2, which was the best overall solver on the quick tests, is not able to find the Speck32 or Present64 trail, while CaDiCal, CryptoMiniSat and Glucose can.

We also see a notable increase in time when the state size is increased: while some SAT solvers can find the lowest known trail for Speck64 on 13 rounds, we can see that the time needed is much higher than the one needed for Speck32, and no solver among all formalisms is able to find the lowest weight trail for Speck128 within the timeout of 24 hours.

4.3 Comparison for Task 2

It has been shown that a solver being fast at finding one solution is not always as fast for enumerating solutions with fixed variables, such as the objective value; for instance, in [23], a SAT solver is used to find solution patterns, which are then explored with a CP solver. In this experiment, we only tested the solvers that returned within the timeout in the `find_lowest_weight_trail` experiment.

As we can see in Figure 3a and Figure 3b, SAT is still a suitable formalism for this problem, though with a different top performer (Yices over Kissat).

Table 3: Results on the optimization problems on the difficult instances, for the solvers that finished within the timeout of 24 hours.

(a) PRESENT 64/80, 18 rounds

Formalism	Building time	Solving time	Memory	Weight	Solver
SAT	**0.13**	**789.75**	**325.69**	**78**	**Kissat**
SAT	0.23	2761.93	311.17	78.0	CaDiCal
SAT	0.14	5757.36	163272.00	78.0	CryptoMiniSat
SAT	0.13	28624.79	inf	78.0	Glucose

(b) Simon 32/64, 12 rounds

Formalism	Building time	Solving time	Memory	Weight	Solver
MILP	**0.95**	**53.20**	**0**	**34.0**	**Gurobi**
SAT	0.03	86.43	208.72	34.0	CaDiCal
SAT	0.03	93.24	218.80	34.0	Kissat
SAT	0.03	132.63	inf	34.0	Glucose
SAT	0.03	432.77	14.39	34.0	Yices2
SAT	0.03	439.43	55.56	34.0	CryptoMiniSat
SMT	0.03	896.70	54.81	34.0	Z3
SAT	0.03	393369.00	56.82	34.0	MathSAT
SMT	0.03	469589.00	21277.00	34.0	Yices2
SMT	0.03	518824.00	100809.00	34.0	MathSAT

(c) Simon 64/128, 19 rounds

Formalism	Building time	Solving time	Memory	Weight	Solver
SAT	**0.11**	**533.09**	**257.62**	**64.0**	**Kissat**
SAT	0.13	64929.70	410.49	64.0	CaDiCal
SAT	0.07	346522.15	inf	64.0	Glucose

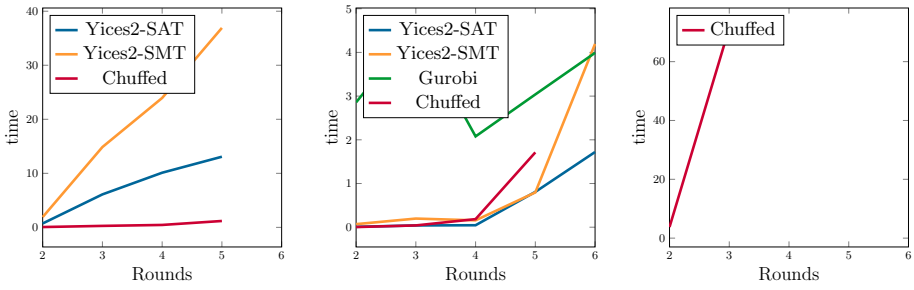
(d) Speck 32/64, 9 rounds

Formalism	Building time	Solving time	Memory	Weight	Solver
SAT	**0.04**	**99.01**	**220.45**	**30.0**	**Kissat**
SAT	0.03	764.28	209.79	30.0	CaDiCal
SAT	0.03	1963.10	inf	30.0	Glucose
SAT	0.03	3266.48	100.24	30.0	CryptoMiniSat
SMT	0.04	75977876.00	817426.00	30.0	MathSAT

(e) Speck 64/128, 13 rounds

Formalism	Building time	Solving time	Memory	Weight	Solver
SAT	**0.12**	**437.96**	**259.22**	**50.0**	**Kissat**
SAT	0.12	67051.43	300.97	50.0	CaDiCal

Furthermore, this time CP's performances improve greatly and in Figure 3c CP is actually the sole formalism to finish within the timeout.



(a) Simon32 (b) Speck32 (c) Gimli

Fig. 3: Graph comparisons between the best solvers for each formalism on different ciphers testing `find_all_trails` function.

4.4 Comparison for Task 3

Our final test compares the time taken to estimate the probability of a differential: the input and output differences were fixed, along with a bound on the probability, and all satisfying trails were enumerated. We used the differentials reported in [3]; in particular, we tested the ones reported for 7 rounds of Speck64, and 14 rounds of Simon32. In addition, we ran this test on 4 rounds of Midori128 to evaluate the influence of a large S-box, which typically favors CP and its table constraints. The results, under a timeout of 6 hours, are reported in Table 4.

For the case of SIMON, we were not able to enumerate all the weights reported in [28] within the timeout, so we only enumerated trails with weights between 38 and 49. As expected, due to the 8-bit S-box, CP was the fastest for Midori128, with all 4 solvers finishing under 12 seconds (Table 4a), followed by SAT solvers, from CryptoMiniSat, which runs in about 13 minutes to MiniSAT (1h24m). Lastly, SMT solvers exhibit even slower performance. With the exception of Yices2, where the performance difference between using SAT or SMT as a formalism is relatively small, all other solvers take over 2 hours to complete, and MathSAT even times out. SPECK, an ARX block cipher, behaves differently: in Table 4b, the only formalism to finish the tests within the timeout is SAT, with the fastest being CaDiCal, which takes around 46 minutes. Due to the inherently boolean nature of ARX operations, an advantage for SAT was expected.

4.5 Speeding up CryptoMiniSat

In a final batch of experiments, we tested the differential probability estimation experiment using the `maxsol` option in CryptoMiniSat; this option lets CryptoMiniSat enumerate solutions up to a given maximum number. In this set of experiments, we set this number of solutions to 10^6, which is arbitrarily higher than the highest number of solutions we observed.

For the 3 ciphers under study, CryptoMiniSat becomes the fastest solver of all the tested ones with this strategy and finishes within 3 seconds for Midori, 40

Table 4: Timing results on the differential probability estimation experiments.

(a) MIDORI 64/128
4 rounds
$\Delta\vec{x} =$ `0x0002002000002000`
$\Delta\vec{y} =$ `0x0000022222022022`
896 trails, $-\log_2(p) = 23.7905$

Formalism	Time	Solver
CP	10.00	Chuffed
CP	10.28	Gecode
CP	10.49	OR Tools
CP	11.26	Choco
SAT	795.71	CryptoMiniSat
SAT	846.49	Yices2
SMT	874.31	Yices2
SAT	941.33	Kissat
SAT	960.07	CaDiCal
SAT	1168.56	Glucose
SAT	1206.09	MathSAT
SAT	5092.36	MiniSAT
SMT	8366.91	Z3

(b) SPECK 64/128
7 rounds
$\Delta\vec{x} =$ `0x4000409210420040`
$\Delta\vec{y} =$ `0x8080a0808481a4a0`
75 trails, $-\log_2(p) = 20.9538$

Formalism	Time	Solver
SAT	2789.58	CaDiCal
SAT	3400.27	Kissat
SAT	3416.21	Glucose
SAT	8785.10	CryptoMiniSat

minutes for SPECK64, and 30 minutes for SIMON32. The significant increase in speed allows us to test larger weights, with a maximum of 58 in this experiment. As a result, we can enumerate a significantly greater number of trails than in our previous experiments, while still maintaining much faster solving times.

5 Conclusion

Differential cryptanalysis is one of the main techniques when testing the strength of symmetric ciphers, and fast evaluation helps designers set the parameters of new primitives; this paper reviews the existing modeling techniques for SAT, SMT, MILP and CP, and compares their performances through different solvers.

In the comparison, solvers from all categories were tested on finding an optimal differential trail, enumerating optimal trails, and estimating the probability of a differential, for block ciphers, permutations and hash functions.

Overall, SAT solvers were the winners of this comparison for ARX primitives and SPNs, such as PRESENT or Midori. In terms of solvers, Kissat dominated the SAT category, Yices2 the SMT pool, Gurobi in MILP and Chuffed won CP.

Even though SAT was the winner in most cases, CP obtained a victory when enumerating the trails of a differential for Midori, in line with previously observed results. On the other hand, when using the `maxsol` option, CryptoMiniSat took the win for enumeration problems.

This work is one further step towards a better understanding of what methods to use for solving differential cryptanalysis problems. A systematic study,

with more primitives and more problems, would be extremely beneficial to the community. Indeed, in future works, we plan to extend similar comparisons for (1) other families of ciphers (such as SPNs or ciphers with large state) and (2) for other types of cryptanalysis, such as linear, differential-linear, and rotational-xor cryptanalysis.

References

1. Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP modeling for (large) s-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. **2017**(4), 99–129 (2017)
2. Abed, F., List, E., Lucks, S., Wenzel, J.: Differential cryptanalysis of round-reduced simon and speck. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 525–545. Springer (2014)
3. Ankele, R., Kölbl, S.: Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In: Cid, C., Jr., M.J.J. (eds.) Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. LNCS, vol. 11349, pp. 163–190. Springer (2018)
4. Audemard, G., Simon, L.: Glucose and syrup: Nine years in the sat competitions. Proceedings of SAT Competition pp. 24–25 (2018)
5. Bellini, E., Gérault, D., Protopapa, M., Rossi, M.: Monte carlo tree search for automatic differential characteristics search: Application to SPECK. In: Isobe, T., Sarkar, S. (eds.) Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings. LNCS, vol. 13774, pp. 373–397. Springer (2022)
6. Bestuzheva, K., Besançon, M., Chen, W.K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S.J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M.E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D., Witzig, J.: The SCIP Optimization Suite 8.0. ZIB-Report 21-41, Zuse Institute Berlin (December 2021)
7. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
8. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
9. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptol. **4**(1), 3–72 (1991)
10. Boura, C., Coggia, D.: Efficient MILP modelings for sboxes and linear layers of SPN ciphers. IACR Trans. Symmetric Cryptol. **2020**(3), 327–361 (2020)
11. Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni-Vincentelli, A.L.: Logic Minimization Algorithms for VLSI Synthesis, The Kluwer International Series in Engineering and Computer Science, vol. 2. Springer (1984)
12. Chu, G., Stuckey, P.J., Schutt, A., Ehlers, T., Gange, G., Francis, K.: Chuffed, a lazy clause generation solver, https://github.com/chuffed/chuffed, last accessed March 19th, 2023

13. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The mathsat5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 93–107. Springer (2013)
14. Dantzig, G.B.: Maximization of a linear function of variables subject to linear inequalities. Activity analysis of production and allocation **13**, 339–347 (1951)
15. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
16. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
17. Delaune, S., Derbez, P., Huynh, P., Minier, M., Mollimard, V., Prud'homme, C.: SKINNY with scalpel - comparing tools for differential analysis. IACR Cryptol. ePrint Arch. p. 1402 (2020)
18. 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.) Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part II. LNCS, vol. 12727, pp. 184–207. Springer (2021)
19. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. LNCS, vol. 8559, pp. 737–744. Springer (2014)
20. Eén, N., Sörensson, N.: An extensible sat-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer (2003)
21. Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: Milp-based automatic search algorithms for differential and linear trails for speck. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 268–288. Springer (2016)
22. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings. LNCS, vol. 4590, pp. 519–531. Springer (2007)
23. Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Revisiting AES related-key differential attacks with constraint programming. Inf. Process. Lett. **139**, 24–29 (2018)
24. Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Computing AES related-key differential characteristics with constraint programming. Artif. Intell. **278** (2020)
25. Google: Or-tools - google optimization tools, https://developers.google.com/optimization, last accessed March 19th, 2023
26. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023)
27. Heule, M., Iser, M., Jarvisalo, M., Suda, M., Balyo, T.: Sat competition 2022, https://satcompetition.github.io/2022/results.html, last accessed March 2nd, 2023
28. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON block cipher family. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 161–185. Springer (2015)
29. Land, A.H., Doig, A.G.: An automatic method for solving discrete programming problems. In: Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.) 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art, pp. 105–132. Springer (2010)
30. Li, T., Sun, Y.: Superball: A new approach for MILP modelings of boolean functions. IACR Trans. Symmetric Cryptol. **2022**(3), 341–367 (2022)

31. Lipmaa, H., Moriai, S.: Efficient algorithms for computing differential properties of addition. IACR Cryptol. ePrint Arch. p. 1 (2001)
32. Liu, Y.: Techniques for Block Cipher Cryptanalysis. Ph.D. thesis, KU Leuven, Faculty of Engineering Science (9 2018), available at: https://www.esat.kuleuven.be/cosic/publications/thesis-306.pdf
33. Liu, Z., Li, Y., Wang, M.: Optimal differential trails in simon-like ciphers. IACR Cryptol. ePrint Arch. p. 178 (2017)
34. Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 81–91. Springer (1992)
35. McCluskey, E.J.: Minimization of boolean functions. Bell System Technical Journal **35**, 1417–1444 (1956)
36. MiniZinc: Minizinc challenge 2022 results, https://www.minizinc.org/challenge2022/results2022.html, last accessed March 2nd, 2023
37. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C., Yung, M., Lin, D. (eds.) Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers. LNCS, vol. 7537, pp. 57–76. Springer (2011)
38. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
39. 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.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer (2007)
40. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0. J. Satisf. Boolean Model. Comput. **9**(1), 53–58 (2014)
41. Oki, E.: Glpk (gnu linear programming kit) (2012)
42. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM Rev. **33**(1), 60–100 (1991)
43. Prud'homme, C., Godet, A., Fages, J.G.: choco-solver, https://github.com/chocoteam/choco-solver, last accessed March 19th, 2023
44. Quine, W.V.: The problem of simplifying truth functions. American Mathematical Monthly **59**, 521–531 (1952)
45. Quine, W.V.: A way to simplify truth functions. American Mathematical Monthly **62**, 627–631 (1955)
46. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2. Elsevier (2006)
47. Schulte, C., Tack, G., Lagerkvyst, M.Z.: Gecode, https://www.gecode.org/index.html, last accessed March 19th, 2023
48. Silva, J.P.M., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Trans. Computers **48**(5), 506–521 (1999)
49. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 244–257. Springer (2009)
50. Sun, L., Wang, W., Wang, M.: Accelerating the search of differential and linear characteristics with the SAT method. IACR Trans. Symmetric Cryptol. **2021**(1), 269–315 (2021)
51. 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)

52. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer (2014)
53. Xu, S., Feng, X., Wang, Y.: On two factors affecting the efficiency of MILP models in automated cryptanalyses. IACR Cryptol. ePrint Arch. p. 196 (2023)
54. Zhou, C., Zhang, W., Ding, T., Xiang, Z.: Improving the MILP-based security evaluation algorithm against differential/linear cryptanalysis using A divide-and-conquer approach. IACR Trans. Symmetric Cryptol. **2019**(4), 438–469 (2019)

A Differential cryptanalysis

Differential cryptanalysis, first proposed by Biham and Shamir in 1990 [9], is a statistical cryptanalysis technique, very effective against many cryptographic primitives, such as block or stream ciphers or hash functions. Given two inputs to the primitive with difference Δx through a chosen operation (we use the XOR, the most common) the technique studies how this value propagates through the iterated operations to reach an output difference Δy.

The *differential probability* of a given input/output pair of differences for a vectorial Boolean function is the probability for that pair to yield over all the possible pairs of inputs with said input difference. For a function f and two differences Δx and Δy, we will denote this probability with $\mathsf{dp}^f(\Delta x \to \Delta y)$.

It is currently infeasible to compute the output difference for a block cipher for all the possible pairs of inputs, considering its large size, and building the table with all the frequencies for each pair of input/output difference (that is called *Difference Distribution Table*, in short DDT). To facilitate the analysis, we can use the fact that block ciphers are often *iterative functions*, i.e. they are the composition $f_{r-1} \circ \ldots \circ f_0$ of simpler keyed round functions f_i's.

We define a r-round *differential trail* (or *characteristic*) for an iterative function $f = f_{r-1} \circ \ldots \circ f_1 \circ f_0$, as a sequence of differences

$$\Delta_0 \xrightarrow{f_0} \Delta_1 \xrightarrow{f_1} \ldots \to \Delta_{r-1} \xrightarrow{f_{r-1}} \Delta_r$$

and a *differential* as a pair of input/output differences. In the case of the whole composite primitive, the differential

$$\Delta x \xrightarrow{f_0 \circ \ldots \circ f_{r-1}} \Delta y.$$

has probability equal to the sum of the probabilities of all the differential characteristics with $\Delta_0 = \Delta x$ and $\Delta_r = \Delta y$, where the probability of the characteristic is usually computed as the product of the probabilities of each intermediate differential of the chain. In particular, one can rely on the assumption of independence between each differential so that the resulting probability, when considering the composition of vectorial Boolean functions, is computed by the following:

Proposition 2. *Let f_1 and f_2 be two vectorial Boolean functions*

$$f_1 : \{0,1\}^l \to \{0,1\}^m\,, \quad f_2 : \{0,1\}^m \to \{0,1\}^n.$$

and let $\Delta\vec{x} \in \{0,1\}^l$, $\Delta\vec{y} \in \{0,1\}^m$ and $\Delta\vec{z} \in \{0,1\}^n$ be three differences such that

$$\mathsf{dp}^{f_1}(\Delta\vec{x} \to \Delta\vec{y}) = p_1 \qquad \mathsf{dp}^{f_2}(\Delta\vec{y} \to \Delta\vec{z}) = p_2.$$

Then, we have

$$\mathsf{dp}^{f_2 \circ f_1}(\Delta\vec{x} \to \Delta\vec{z}) = p_1 \cdot p_2.$$

To simplify the search for the most probable differential trail, it is common to search for the best differential characteristic instead, assuming its probability to be a good approximation of the target one, even if this is not always true [3].

In general, there is no efficient way to compute the precise probability of a differential characteristics. To do so, some fundamental assumptions on block ciphers are commonly used, such as the *Markov cipher* assumption, the *Hypothesis of stochastic equivalence* and the *Hypothesis of independent round keys* (see e.g. [32, Section 2.2.1]).

B Formalisms

In order to search for differential trails having the highest possible probability, we will make use of several constraints problems solvers adopting 4 different formalisms. The problem underlying the search of differential trails can be set from a general point of view.

Problem 1. Given a set of *variables* (unknown elements with a fixed domain) and a set of *constraints* (e.g. relations representing the propagation of the difference through the cipher), it is required to find an *assignment* of the variables to values in their domains, that is a mapping associating to each variable a value in its domain, that satisfies all the constraints.

We will call the resolution process *procedure*. In the following, we specialize the general terminology for each of the 4 formalisms we have used.

B.1 Satisfiability (SAT)

The terminology is as follows:

- *variables* are Boolean unknowns; a literal is either an unknown Boolean quantity v_i or its negation $\neg v_i$;
- *constraints* are clauses; a clause is a disjunction of literals, $\bigvee_{i=0}^{n-1} x_i$; the set of clauses is called Conjunctive Normal Form (CNF) and it is the conjunction of all the clauses, $\bigwedge_{j=0}^{m-1} \left(\bigvee_{i=0}^{n_j} x_{ij}\right)$;
- the main *procedures* are DPLL [16,15] or CDCL [48], improved in the actual implementations.

B.2 Satisfiability Modulo Theories (SMT)

The terminology is as follows:

- *variables* are unknown Booleans x_i coming from the quantifier free theory, i.e. the Boolean logic;
- *constraints* are formulae in the chosen theory involving Boolean symbols;
- the main *procedures* are Lazy or Eager [8]; due to the simplicity of implementation, Lazy is the most widely implemented.

B.3 Mixed-Integer Linear Programming (MILP)

The terminology is as follows:

- *variables* are unknown quantities x_i that can either be booleans, integers ($\mathbb{Z}$) or continuous ($\mathbb{R}$);
- *constraints* are linear inequalities of the form $a_0x_0+a_1x_1+\cdots+a_{n-1}x_{n-1} \leq b$ with $a_i, b \in \mathbb{Q}$; moreover we have an objective function of the form $z = c_0x_0 + c_1x_1 + \cdots + c_{n-1}x_{n-1}$ to be maximized or minimized, with $c_i \in \mathbb{Q}$;
- the main *procedures* are the Simplex algorithm [14], Branch-and-bound [29] and Branch-and-cut [42].

B.4 Constraint Programming (CP)

The terminology is as follows:

- *variables* are unknown quantities belonging to a specific domain, i.e. pairs (x_i, D_i). In our models we will either have Boolean variables ($D_i = \{0, 1\}$) or more generic integer variables ($D_i \subseteq \mathbb{N}$);
- *constraints* are relations which involve a subset of the variables. There are several types of constraints that can be used to model CP problems; in our models we used linear equations of integer variables (eventually modulo 2), logical combinations of linear equations of integer variables through the usual operators (AND, OR, NOT) and table constraints.
- the main *procedures* are Backtracking search, Local Search and Dynamic programming [46].

C Experimental results tables

In Table 5, we use the following notation: BT = Building Time, ST = Solving Time, NR = Number of Rounds, W = Weight.

		NR=2, W=2			NR=3, W=4			NR=4, W=6			NR=5, W=8			NR=6, W=12		
Formalism	Solver	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory
SAT	Kissat	0.00	0.00	213.81	0.00	0.02	213.44	0.00	0.04	214.22	0.01	0.11	214.72	0.01	0.36	215.47
SAT	CaDiCal	0.00	0.00	203.40	0.00	0.02	203.61	0.00	0.07	204.23	0.01	0.16	205.01	0.02	0.45	206.13
SAT	CryptoMiniSat	0.00	0.00	5.59	0.00	0.03	5.80	0.01	0.06	5.82	0.01	0.15	6.12	0.01	0.51	6.84
SAT	MiniSAT	0.00	0.04	10.56	0.00	0.07	10.57	0.01	0.21	10.71	0.01	0.42	11.01	0.02	1.25	11.30
SAT	Yices2	0.00	0.00	3.50	0.00	0.01	3.63	0.00	0.04	3.76	0.01	0.08	3.76	0.01	0.23	4.02
SAT	MathSAT	0.00	0.01	8.60	0.01	0.03	8.60	0.00	0.06	9.11	0.01	0.14	9.37	0.01	0.41	10.14
SAT	Glucose	0.00	0.01	inf	0.01	0.04	inf	0.00	0.14	inf	0.01	0.25	inf	0.02	0.85	inf
SMT	Yices2	0.00	0.02	6.76	0.00	0.04	6.95	0.01	0.09	7.24	0.01	0.16	7.56	0.01	0.40	8.02
SMT	MathSAT	0.00	0.05	15.52	0.00	0.10	16.81	0.01	0.15	18.10	0.00	0.27	19.91	0.01	0.71	23.00
SMT	Z3	0.00	0.05	18.63	0.00	0.12	19.04	0.01	0.22	19.63	0.01	0.44	20.55	0.01	1.42	21.77
CP	Chuffed	0.00	0.00	0.12	0.00	0.01	0.19	0.00	0.09	0.28	0.00	0.63	0.28	0.00	20.46	0.42
CP	Gecode	0.00	0.00	inf	0.00	0.02	inf	0.00	0.16	inf	0.00	1.22	inf	0.00	30.34	inf
CP	Choco	0.00	0.03	inf	0.00	0.12	inf	0.00	0.49	inf	0.00	2.76	inf	0.00	63.79	inf
CP	OR Tools	0.00	0.02	inf	0.00	0.03	inf	0.00	0.20	inf	0.00	1.29	inf	0.00	33.13	inf
MILP	GLPK	0.06	0.11	0.00	0.07	1.03	0.00	2.85	0.19	0.00	2.96	0.41	0.00	-	-	-
MILP	Gurobi	2.78	0.01	0.00	2.92	0.09	0.00	0.08	7.60	0.00	0.15	38.71	0.00	3.07	0.66	0.00

Table 5: Comparison results on Simon 32/64

		NR=2, W=1			NR=3, W=3			NR=4, W=5			NR=5, W=9			NR=6, W=13		
Formalism	Solver	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory
SAT	Kissat	0.00	0.00	213.46	0.00	0.02	214.79	0.01	0.07	215.38	0.01	0.47	216.47	0.02	2.46	218.04
SAT	CaDiCal	0.01	0.00	204.06	0.00	0.03	203.95	0.01	0.09	205.85	0.01	0.54	206.74	0.02	1.92	207.89
SAT	CryptoMiniSat	0.00	0.01	5.79	0.00	0.04	5.87	0.01	0.07	6.02	0.02	0.71	7.50	0.02	4.26	11.65
SAT	MiniSAT	0.00	0.03	10.56	0.01	0.10	10.73	0.01	0.27	10.84	0.01	1.94	10.30	0.02	15.88	11.54
SAT	Yices2	0.01	0.00	3.63	0.00	0.02	3.63	0.01	0.06	3.89	0.02	0.33	4.16	0.02	3.17	4.72
SAT	MathSAT	0.00	0.01	8.60	0.00	0.04	9.11	0.01	0.10	9.37	0.01	0.63	10.66	0.02	3.43	13.23
SAT	Glucose	0.00	0.01	inf	0.00	0.04	inf	0.01	0.19	inf	0.01	0.69	inf	0.02	3.42	inf
SMT	Yices2	0.00	0.01	6.79	0.01	0.04	7.07	0.01	0.10	7.55	0.01	0.66	8.27	0.02	4.38	10.22
SMT	MathSAT	0.00	0.03	16.04	0.01	0.14	19.39	0.01	0.41	25.58	0.01	1.53	35.89	0.02	9.25	65.57
SMT	Z3	0.00	0.04	18.67	0.01	0.12	19.30	0.01	0.36	20.27	0.01	2.17	22.66	0.02	12.88	26.92
CP	Chuffed	0.00	0.00	0.05	0.00	0.04	0.12	0.00	0.81	0.19	0.00	132.69	0.28	-	-	-
CP	Gecode	0.00	0.00	inf	0.00	0.01	inf	0.00	0.33	inf	0.00	74.62	inf	-	-	-
CP	Choco	0.00	0.04	inf	0.00	0.24	inf	0.00	17.41	inf	-	-	-	-	-	-
CP	OR Tools	0.00	0.02	inf	0.00	0.06	inf	0.00	0.47	inf	0.00	28.44	inf	-	-	-
MILP	GLPK	0.04	14.92	-	-	-	-	-	-	-	-	-	-	-	-	-
MILP	Gurobi	2.70	0.05	0.00	2.93	0.29	0.00	3.02	1.33	0.00	2.84	3.27	0.00	3.09	21.44	0.00

Table 6: Comparison results on Speck 32/64

		NR=2, W=0			NR=3, W=0			NR=4, W=1			NR=5, W=6			NR=6, W=7		
Formalism	Solver	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory
SAT	Kissat	0.04	0.02	173.08	0.05	0.03	181.31	0.06	0.21	191.15	0.08	18.40	225.68	0.11	55.87	237.44
SAT	CaDiCal	0.03	0.02	162.30	0.05	0.03	171.22	0.06	0.12	184.59	0.08	47.38	208.79	0.11	62.59	230.71
SAT	CryptoMiniSat	0.03	0.02	10.18	0.04	0.02	12.15	0.06	0.13	16.20	0.09	73.82	112.40	0.11	83.68	97.73
SAT	MiniSAT	0.04	0.21	14.57	0.05	0.31	16.07	0.06	1.01	21.82	0.09	280.28	103.45	0.11	367.98	235.89
SAT	Yices2	0.04	0.00	6.29	0.05	0.00	7.69	0.06	0.20	10.02	0.09	84.74	40.92	0.11	150.09	51.24
SAT	MathSAT	0.03	0.04	15.43	0.05	0.06	18.27	0.06	0.30	23.42	0.09	162.17	134.20	0.11	284.66	257.83
SAT	Glucose	0.04	0.04	119.50	0.05	0.04	124.34	0.06	0.18	inf	0.09	110.01	inf	0.11	166.23	inf
SMT	Yices2	0.02	0.05	12.22	0.03	0.07	15.21	0.05	0.21	23.92	0.09	127.53	71.81	0.11	222.02	94.72
SMT	MathSAT	0.02	0.28	37.68	0.03	0.81	56.50	0.05	1.01	92.64	0.10	174.88	268.01	0.12	331.54	307.70
SMT	Z3	0.02	0.21	37.20	0.03	0.34	68.50	0.05	2.12	76.78	0.09	448.57	163.91	0.13	666.45	254.68
CP	Chuffed	0.01	0.01	3.16	0.02	0.03	4.75	0.02	0.07	4.75	-	-	-	-	-	-
CP	Gecode	0.01	0.03	inf	0.01	0.05	inf	0.02	0.26	inf	-	-	-	-	-	-
CP	Choco	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CP	OR Tools	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MILP	GLPK	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MILP	Gurobi	0.94	0.44	0.00	3.02	382.40	0.00	-	-	-	-	-	-	-	-	-

Table 7: Comparison results on Blake 512

		NR=2, W=4			NR=3, W=6			NR=4, W=10			NR=5, W=16			NR=6, W=28		
Formalism	Solver	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory
SAT	Kissat	0.04	0.14	225.93	0.05	0.88	233.74	0.09	5.47	245.46	0.08	27.92	262.10	0.12	355.12	302.89
SAT	CaDiCal	0.04	0.20	215.73	0.07	1.00	222.27	0.08	4.57	233.96	0.09	23.03	251.19	0.12	484.81	288.71
SAT	CryptoMiniSat	0.04	0.42	9.55	0.05	1.61	9.55	0.07	11.40	20.57	0.09	70.18	48.39	-	-	-
SAT	MiniSAT	0.03	1.14	13.90	0.05	3.53	16.64	0.07	24.27	21.23	0.09	217.81	45.50	-	-	-
SAT	Yices2	0.04	0.14	5.70	0.05	1.34	7.70	0.07	12.17	12.64	0.09	81.29	18.93	-	-	-
SAT	MathSAT	0.04	0.34	14.68	0.05	2.06	19.42	0.04	14.44	32.57	0.09	103.20	56.91	-	-	-
SAT	Glucose	0.04	0.21	inf	0.05	0.99	inf	0.07	7.21	inf	-	-	-	-	-	-
SMT	Yices2	0.03	0.34	12.98	0.05	1.62	18.63	0.07	9.16	27.50	0.11	75.54	42.49	-	-	-
SMT	MathSAT	0.04	0.87	52.13	0.05	2.89	80.23	0.07	14.35	125.35	0.11	99.44	208.36	-	-	-
SMT	Z3	0.04	1.47	39.06	0.05	5.10	72.12	0.07	33.32	136.04	0.11	569.40	149.982	-	-	-
CP	Chuffed	0.02	0.30	0.94	0.01	22.01	1.41	-	-	-	-	-	-	-	-	-
CP	Gecode	0.02	0.15	inf	0.03	22.39	inf	-	-	-	-	-	-	-	-	-
CP	Choco	0.02	46.04	inf	-	-	-	-	-	-	-	-	-	-	-	-
CP	OR Tools	0.02	0.72	inf	-	-	-	-	-	-	-	-	-	-	-	-
MILP	GLPK	0.57	75.56	0.00	-	-	-	-	-	-	-	-	-	-	-	-
MILP	Gurobi	4.50	1.45	0.00	5.92	22.49	0.00	7.62	452.57	0.00	-	-	-	-	-	-

Table 8: Comparison results on Gimli 384

		NR=2, W=4			NR=3, W=8			NR=4, W=12			NR=5, W=20			NR=6, W=24		
Formalism	Solver	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory	BT	ST	Memory
SAT	Kissat	0.00	0.03	218.14	0.02	0.14	219.83	0.03	0.46	222.53	0.03	5.64	226.42	0.03	11.02	231.10
SAT	CaDiCal	0.02	0.04	207.43	0.02	0.19	209.20	0.02	1.08	212.66	0.03	5.78	216.67	0.04	11.25	220.55
SAT	CryptoMiniSat	0.02	0.04	6.08	0.02	0.04	6.08	0.03	0.81	7.99	0.03	9.37	13.23	0.03	31.09	21.76
SAT	MiniSAT	0.02	0.21	11.07	0.02	0.21	11.07	0.03	3.43	12.58	0.03	21.28	14.02	0.03	62.39	15.38
SAT	Yices2	0.01	0.01	3.89	0.02	0.13	4.29	0.03	0.49	4.73	0.03	5.74	6.04	0.03	18.47	7.27
SAT	MathSAT	0.02	0.05	9.37	0.02	0.89	33.57	0.03	0.80	11.95	0.03	7.08	16.59	0.04	18.93	18.91
SAT	Glucose	0.02	0.08	inf	0.02	0.42	inf	0.02	1.29	inf	0.03	8.12	inf	0.03	15.87	inf
SMT	Yices2	0.02	0.09	7.84	0.02	0.27	9.18	0.03	0.73	11.03	0.04	6.16	14.11	0.05	16.91	18.37
SMT	MathSAT	0.02	0.29	25.06	0.02	0.89	33.57	0.03	2.32	47.23	0.04	9.97	71.47	0.05	27.06	91.30
SMT	Z3	0.02	0.50	20.94	0.02	1.66	23.75	0.03	4.13	27.24	0.04	18.51	42.50	0.05	42.97	73.78
CP	Chuffed	0.01	0.00	0.19	0.01	0.11	0.28	0.01	3.13	0.42	-	-	-	-	-	-
CP	Gecode	0.01	0.00	inf	0.01	0.08	inf	0.01	2.43	inf	0.01	399.94	inf	-	-	-
CP	Choco	0.01	0.04	inf	0.01	0.31	inf	0.01	7.12	inf	-	-	-	-	-	-
CP	OR Tools	0.01	0.25	inf	0.01	0.47	inf	0.01	4.80	inf	-	-	-	-	-	-
MILP	GLPK	0.20	1.01	0.00	-	-	-	-	-	-	-	-	-	-	-	-
MILP	Gurobi	3.08	0.07	0.00	3.20	0.53	0.00	3.37	0.92	0.00	3.44	3.81	0.00	3.64	111.07	0.00

Table 9: Comparison results on Present 64/80