Finding Integral Distinguishers with Ease

Zahra Eskandari[1], Andreas Brasen Kidmose[2], Stefan Kölbl[2,3], and Tyge Tiessen[2]
zahra.eskandari@mail.um.ac.ir, abki@dtu.dk, stek@mailbox.org

[1] Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
[2] DTU Compute, Technical University of Denmark, Denmark
[3] Cybercrypt, Denmark

Abstract. The division property method is a technique to determine integral distinguishers on block ciphers. While the complexity of finding these distinguishers is higher, it has recently been shown that MILP and SAT solvers can efficiently find such distinguishers. In this paper, we provide a framework to automatically find those distinguishers which solely requires a description of the cryptographic primitive. We demonstrate that by finding integral distinguishers for 30 primitives with different design strategies.
We provide several new or improved bit-based division property distinguishers for ChaCha, Chaskey, DES, GIFT, LBlock, Mantis, Qarma, RoadRunner, Salsa and SM4. Furthermore, we present an algorithm to find distinguishers with lower data complexity more efficiently.

Keywords: Integral Attacks, Division Property, Tool

1 Introduction

Block ciphers, stream ciphers, and hash functions are the fundamental symmetric cryptographic primitives that are at the base of almost all cryptographic protocols. One of the most successful set of techniques to evaluate their security are techniques based on higher-order derivatives.

Higher-order derivatives were first considered in the context of symmetric cryptography by Xuejia Lai [Lai94] and shown by Lars R. Knudsen [Knu95] to attack weaknesses not covered by differential cryptanalysis, and successfully used to break a cipher design [JK97]. A higher-order derivative in the context of cryptography is the discrete equivalent of higher-order derivatives of multivariate continuous functions. The cryptographic primitive can be seen as a vectorial Boolean function where a higher-order derivative evaluates this function at a given point with respect to some directions/subspace. Such a derivative can for example be used to find the coefficients of the monomials of the algebraic normal form (ANF) of a cryptographic primitive.

An important category of higher-order attacks is integral cryptanalysis. This type of cryptanalysis appeared first in the Square attack [DKR97a], and was later generalised to be apply to other ciphers as well ([KW02,BS10]). In integral cryptanalysis, the goal is to find a set of input bits and a set of output bits, such that when taking the sum over a set of input messages taking all possible values in the selected input bits and arbitrary but constant values in the other input bits, the sum will be balanced in the selected output bits. This can be described as a higher-order derivative that can be taken at any point and evaluates to zero in the specified output bits.

Originally such property was derived using arguments based on the structure of the primitive but Yosuke Todo demonstrated in his EUROCRYPT 2015 paper [Tod15b] a novel method to derive integral distinguishers using the so-called division property formalism whose effectiveness he demonstrated with an attack on full-round Misty [Tod15a]. The technique originally being used on words of at least four bits, has since been applied to bit-based designs as well, albeit at a higher computational cost [TM16].

Another type of higher-order attacks are so-called cube attacks [Vie07,DS09]. In these attacks the cryptographic primitive is viewed as a vectorial Boolean function in both public and secret input bits. By finding coefficients of terms in the public bits that are linear in the secret bits, it is possible to derive a set of linear equations that we can solve to extract the secret input bits. This technique has successfully been applied to stream ciphers and hash functions [DS11,DMP+15].

Contributions This paper presents a new framework to analyse the security of cryptographic primitives with respect to the bit-based division property by providing a simple way to find distinguishers and testing the number of rounds required for no such distinguisher to exist. We take a look at how finding division property distinguishers can be efficiently automated. To this end, we elaborate how the bit-based division property can be mapped to conditions on the state bits which in turn maps easily to a SAT problem.

Our tool focuses especially on the usability and allows to describe the cryptographic primitives at a high level by providing commonly used operations like S-boxes, linear layers, bit-permutations or modular addition. This completely removes the need of constructing any domain specific models like previous search strategies [XZBL16,SWW17,ZR17].

In order to demonstrate the usability of our tool we implemented 30 primitives following different design strategies. We then use our tool to find several new integral distinguishers, provide a bound for which number no such distinguishers exist in our model and also evaluate for which design strategies our approach becomes computationally infeasible.

In particular we find the following new results:

- We provide the first bit-based integral distinguishers for the permutations used in ChaCha (6 rounds), Chaskey (4 rounds) and Salsa (6 rounds). We further show that for one more round no distinguisher of this type exists.
- For DES we show that by using the bit-based division property we can improve upon the word-based division property distinguishers by Todo [Tod15b] and add one round. We also show that for 8 rounds no such distinguishers exist.
- We present the first integral distinguisher for both Mantis (3 forward, 2 backward rounds) and several variants of Qarma (2 forward, 2 backward rounds).
- For the SM4 block cipher we can show a distinguisher for 12 rounds and that no bit-based division property distinguisher exists for 13 rounds. This improves the best previously known integral distinguisher by 4 rounds [LJH+07].
- We find a distinguisher for 17 rounds of LBlock, which improves the best previously known results by one round [XZBL16].
- We present 9-round distinguishers for GIFT-64 which improve upon the data complexity of the distinguishers provided by the designers [BPP+17].
- For RoadRunner we are able to extend the distinguishers found by the authors [BS15] by one additional round.

For several other primitives we provide a bound at which no bit-based division property distinguishers exists in our model. Furthermore, we present an efficient algorithm to find distinguishers with reduced data complexity by only covering the search space which can actually lead to distinguishers.

Software. We place the tool developed for this paper into the public domain and it is available at `https://github.com/kste/solvatore`.

Related Work The division property has been applied to a large variety of cryptographic primitives and has led to significant improvements [Tod15b,Tod15a] over classical integral attacks in some cases. With the extension of the division property to bit-based designs [TM16] the technique can be applied to a larger class of cryptographic primitives.

However finding distinguishers with this approach is a difficult task and requires a lot of effort.

The first automated approach for finding bit-based division property distinguishers was presented in [SWW16] and is based on reducing the problem to mixed integer linear programming (MILP). This simplifies the search for distinguishers and allows to apply the bit-based division property to a larger class of cryptographic primitives. Another automated approach based on constraint programming has been proposed in [SGL+17] to find integral distinguishers for PRESENT. In the paper the authors show that this approach can have a better performance than the MILP based technique. The search for ARX and word-based division property has been dealt with in [SWW17] by using SAT resp. SMT solvers.

2 Division property and division trails

The methodology of division properties was devised by Yosuke Todo in his EUROCRYPT 2015 paper [Tod15b]. We elaborate this methodology here in the setting where the words are single bits, i.e., when applied as bit-based division property. While using the original formalism, we will look at it from a slightly different angle to simplify the discussion. For the division property over larger word sizes, we refer to the original paper.

2.1 Background

The formalism of division properties belongs to the family of attack vectors collectively named integral cryptanalysis. The goal of integral cryptanalytic techniques is to find a set of input texts such that the sum of the resulting output texts evaluates to zero in some of the bits. If such a property can be found it directly yields a distinguisher which often can be turned into a key recovery attack.

The most common sets of input texts that are used are those that are equal in some bit positions and take all possible combination of values in the remaining bit positions. The first attack that successfully used this attack vector is the Square attack [DKR97b] on the block cipher Square that is equally applicable to the Advanced Encryption Standard (AES).

There are two main methods that are used to derive an integral distinguisher: structural properties and algebraic degree bounds. In the Square attack and subsequent generalizations [BS01] the integral property could be derived by only looking at structural properties of the cipher such as the SPN or Feistel structure without taking much of the cipher details into consideration (such as concrete S-box, concrete linear layer).

Later it was recognised that these kinds of integral distinguishers correspond to discrete derivatives [Lai94] where the derivative is taken with respect to the active input bits, i.e., those that are varied. As such the structural techniques are a way to determine output bits whose polynomial representations do not contain terms that include all active input bits simultaneously. Taking the derivative with respect to these active input bits will thus necessarily evaluate to zero in these output bits.

The second major technique that is used to derive integral distinguishers uses this view of integral distinguishers as derivatives. By determining upper bounds on the algebraic degree of the polynomials of the output bits, we can determine that derivatives of sufficient degree have to evaluate to zero. Similar to the structural method, the methods used to bound the degree usually ignore large parts of the implementation details, for example by just looking at the degree of rounds and multiplying these.

The division property is an improvement with respect to this situation as it manages to take more implementation details of the cipher into consideration. The downside to this is an increased cost of finding the distinguishers.

2.2 Formalism of bit-based division properties

In the bit-based division property methodology, the goal is to find, given a set of chosen active input bits, those output bits whose polynomial representations do not contain terms that feature all of these active bits simultaneously. While this could principally be done by simply calculating the exact polynomial representations of the output bits, this is computationally infeasible in all but toy examples. With division properties we use an approximation instead that guarantees to only find valid distinguishers but might fail to find all distinguishers.

In this approximation, we continually track which bits of the state would need to be multiplied to generate a bit whose polynomial representation can contain terms of all active bits. Let us consider an initial state of four bits (x_0, x_1, x_2, x_3) where we activate bits x_1 and x_2, i.e., we are interested in which state bits we would need to multiply to create a term that contains both bits. For this initial state the minimal way of generating such a term is by multiplying those two bits directly. We write this combination as the choice vector $(0, 1, 1, 0)$.[4]

If we now add x_1 to x_3, we get the new state $(x_0, x_1, x_2, x_3 + x_1)$. Now we can generate a term that contains both x_1 and x_2 in two different minimal ways: first again by multiplying the second and third bit or by multiplying the third and the last bit. These correspond to the choice vectors $(0, 1, 1, 0)$ and $(0, 0, 1, 1)$.[5] The only original choice vector $(0, 1, 1, 0)$ has thus been transformed to two choice vectors by the application of the addition.

If we now applied another operation to this state, each of the choice vectors is transformed to other minimal choice vectors, and by iterating this process a tree of minimal choice vectors is spanned whose final nodes are the minimal choice vectors of output bits whose multiplication can create a term that contains all active input bits.

To determine whether a minimal choice vector can be reached from the initial choice vector of active bits, we need to determine whether a path exists in this tree from the initial choice vector to the output choice vector. We will refer to such path as a *division trail*. In particular, to determine whether a specific output bit is zero when evaluating the derivative with respect to the active bits, we need to determine whether the choice vector that only chooses this output bit is reachable. If it is not reachable, we know that this output bit cannot have terms in its polynomial representation that contain all active bits simultaneously and thus the derivative has to evaluate to zero. Should the choice vector be reachable though, nothing definite can be said about the derivative.

2.3 Rules of choice vector propagation

To trace a division trail of minimal choice vectors, we need to know how these minimal choice vectors of state bits are transformed to new choice vectors under the application of operations. In the following we will shortly discuss the application of XOR, AND, bit-copying and S-boxes. As the influence of the operations is local, it is sufficient to restrict the discussion to those bits involved in the operation.

Bit-Copying Let us take a look at the scenario where we have two state bits, and the value of the first bit is copied to the second bit. There are four possible original choice vectors: $(0, 0)$, $(1, 0)$, $(0, 1)$, and $(1, 1)$. The first choice vector implies that to generate a term that can contain all active bits, we don't need to multiply any of the two bits. So clearly we still do not need to multiply any of the bits after copying the first bit onto the second, leading to the transition $(0, 0) \to (0, 0)$.

[4] In the original paper, this was written slightly more verbosely as $\mathcal{D}^4_{(0,1,1,0)}$.
[5] In the original paper, this would be written as $\mathcal{D}^4_{(0,1,1,0),(0,0,1,1)}$.

In the case of $(1,0)$, we need the first bit in the product to generate a term with all active bits but the second one is not required. Thus after copying, we can choose either the first or the second bit (both would also be possible but not minimal). We thus have the two transitions: $(1,0) \to (1,0)$ and $(1,0) \to (0,1)$.

Now in the case of $(1,0)$ and $(1,1)$, the second bit is needed in the product to create a term with all active bits. As it is copied over, it is no longer possible after copying to create this term and thus no valid transitions exist.

XOR Now for the case where there are two state bits and the first is XORed onto the second. Again we have to look at the four cases $(0,0)$, $(1,0)$, $(0,1)$, and $(1,1)$. As with bit copying, in the case of $(0,0)$, the bits are not necessary in the product, so they are not necessary after the addition as well. This leads to the transition $(0,0) \to (0,0)$.

In the case of $(1,0)$, the first bit value is needed in the product. After the addition, the bit value is also present as part of the sum in the second bit. We can thus either choose the first or the second bit in the product, leading to the transitions $(1,0) \to (1,0)$ and $(1,0) \to (0,1)$.

When we have the case $(0,1)$, the second bit value is needed in the product. As it is still only present in the second bit after the addition, the only valid transition here is $(0,1) \to (0,1)$.

Finally, in the case of $(1,1)$, the product of both bits is needed to create a term with all active bits. Although the second bit contains both original bit values after the addition, it only does so as a sum while we need the product of both. Thus also after the addition, we have to choose both bits, leading to the transition $(1,1) \to (1,1)$.

AND If we now have again two state bits and we multiply the first onto the second, the situation is analogous to the case of the XOR except if the choice vector before the multiplication is $(1,1)$. In this case the product of both bit values is needed to create a term of all active bits. As the multiplication creates exactly this product in the second bit, the only minimal transition here is $(1,1) \to (0,1)$.

S-boxes The easiest way to see how choice vectors are transformed by an S-box is to look at the polynomial representation of the S-box, i.e., the algebraic normal form (ANF). It is tedious but straightforward to deduce the valid output choice vectors for a given input choice vector using the ANF. It can hence be easily automated and we only need to do this once for an S-box.

3 Solvatore - Automated Finding of Integral Properties

Finding integral distinguishers using division properties is a difficult task. Especially for bit-based designs the analysis often requires extensive manual work which is prone to errors. Automatic tools can be very useful and simplify the analysis of cryptographic primitives, allowing us to explore a larger set of attack vectors. On the other hand they can also be very useful in the design process of cryptographic primitives, to optimise parameters and quickly test different design strategies.

In the following, we present our automated tool Solvatore, which simplifies the search for bit-based division property distinguishers by providing a framework for implementing a large variety of cryptographic primitives. One of the main focuses of the framework is to not only automate finding the bit-based division property distinguishers, as done in previous work [XZBL16,SWLW16,SWW17], but also to completely abstract away the need for dealing with generating models for the primitives or requiring any domain specific knowledge. This makes it much simpler and less error-prone compared to other approaches

to add new primitives to the framework and in general it is far easier to implement a primitive in our tool than writing a standard C implementation as many details can be omitted.

Currently our framework supports the following operations to construct cryptographic primitives:

- Bit operations: bit-copying, **and**, and **xor**.
- Arbitrary S-boxes.
- Linear layers using matrix multiplication over arbitrary fields.
- Modular Addition.
- Bit-permutations.
- Generic cell permutations for ShiftRows or Midori-like constructions.

As an example the full description of Present is given in section A which only requires to define the S-box, bit-permutation and on which bits those are applied. In order to analyse the security of Present against the bit-based division property our tool provides functions for checking whether an output bit is balanced for a given choice vector.

In the following we show how we can reduce the problem of finding a division trail to a satisfiability problem. For this we have to construct a Boolean formula which is satisfiable if and only if it forms a valid division trail.

3.1 Modeling division property propagation with SAT

The *Boolean satisfiability problem* (SAT) is a well known problem from computer science. The problem is to decide whether there exists an assignment of variables in a Boolean formula in conjunctive normal form (CNF) such that the formula evaluates to `true`. While the problem is known to be NP-complete, the SAT instances we will construct here are very structured and can often be solved quickly in practice by modern SAT solvers. In the following we show how to reduce the problem of finding division trails to a SAT problem and how this can be useful in the cryptanalysis of cryptographic primitives.

First, we introduce a variable for each bit of the choice vector $S^i = (s_0, \ldots, s_{n-1})$ after the ith operation applied to the state where n is the size of the state. The next step is to define how the choice vector can propagate through different Boolean functions which occur in the round functions of cryptographic primitives. The rules for this have been explained in subsection 2.3 and have also been studied in [Tod15a,Tod15b]. We therefore focus here on how we can construct a Boolean formula in CNF which is SAT if and only if the assignment of the variables forms a valid transition of choice vectors.

Bit-Copying. The **copy** operation copies a bit a to an output bit b, and all valid transitions of choice vectors are given by

$$\begin{aligned} \mathbf{copy}(a_{\text{old}}, b_{\text{old}}) &\to \{(a_{\text{new}}, b_{\text{new}})\} \\ \mathbf{copy}(0,0) &\mapsto \{(0,0)\} \\ \mathbf{copy}(1,0) &\mapsto \{(1,0),(0,1)\}. \end{aligned}$$

The set of clauses C_{copy} which form a Boolean formula which is SAT iff $(a_{\text{old}}, b_{\text{old}}) \xrightarrow{\text{copy}} (a_{\text{new}}, b_{\text{new}})$ is given by

$$\begin{aligned} C_{\text{copy}} = \{&(\neg b_{\text{old}}), (\neg a_{\text{old}} \vee b_{\text{new}} \vee a_{\text{new}}), (a_{\text{old}} \vee \neg b_{\text{new}}), \\ &(a_{\text{old}} \vee \neg a_{\text{new}}), (\neg a_{\text{new}} \vee \neg b_{\text{new}})\}. \end{aligned} \tag{1}$$

And. The **and** operation corresponds to the result of $a \wedge b \to b$. The valid transitions are given by

$$\begin{aligned}\mathbf{and}(a_{\text{old}}, b_{\text{old}}) &\to \{(a_{\text{new}}, b_{\text{new}})\}\\ \mathbf{and}(0,0) &\mapsto \{(0,0)\}\\ \mathbf{and}(0,1) &\mapsto \{(0,1)\}\\ \mathbf{and}(1,0) &\mapsto \{(1,0),(0,1)\}\\ \mathbf{and}(1,1) &\mapsto \{(0,1)\}.\end{aligned}$$

Just as for the **copy** operation, translating this to a SAT sentence is straightforward and gives the following set of clauses

$$\begin{aligned}C_{\text{and}} = \{&(a_{\text{old}} \vee \neg a_{\text{new}}), (\neg b_{\text{old}} \vee b_{\text{new}}), (\neg b_{\text{new}} \vee \neg a_{\text{new}}),\\ &(\neg a_{\text{old}} \vee b_{\text{new}} \vee a_{\text{new}}), (a_{\text{old}} \vee b_{\text{old}} \vee \neg b_{\text{new}}).\end{aligned} \tag{2}$$

Xor. The **xor** operation corresponds to the result of $a \oplus b \to b$. The valid transitions are given by

$$\begin{aligned}\mathbf{xor}(a_{\text{old}}, b_{\text{old}}) &\to \{(a_{\text{new}}, b_{\text{new}})\}\\ \mathbf{xor}(0,0) &\mapsto \{(0,0)\}\\ \mathbf{xor}(0,1) &\mapsto \{(0,1)\}\\ \mathbf{xor}(1,0) &\mapsto \{(1,0),(0,1)\}\\ \mathbf{xor}(1,1) &\mapsto \{(1,1)\}\end{aligned}$$

which corresponds to the following clauses

$$\begin{aligned}C_{\text{xor}} = \{&(a_{\text{old}} \vee \neg a_{\text{new}}), (\neg b_{\text{old}} \vee b_{\text{new}}), (b_{\text{old}} \vee \neg b_{\text{new}} \vee \neg a_{\text{new}}),\\ &(\neg a_{\text{old}} \vee a_{\text{new}} \vee b_{\text{new}}), (b_{\text{old}} \vee a_{\text{old}} \vee \neg b_{\text{new}}),\\ &(\neg b_{\text{old}} \vee \neg a_{\text{old}} \vee a_{\text{new}}).\end{aligned} \tag{3}$$

S-boxes. As described in subsection 2.3, the transition rules for S-boxes can easily be deduced automatically. The rules create a truth table for involved variables which can be transformed to a CNF using standard methods.

Linear Layers. Many popular designs, like the AES, use a complex linear layer in order to get good diffusion. These linear layers are often represented as $d \times d$ matrices over some field $\mathbb{F}_2^k$. In order to model the trail propagation we can represent these transformations as $kd \times kd$ matrices over $\mathbb{F}_2$, which then can be decomposed into the basic **copy** and **xor** operations.

In order to simplify the description of such linear layers in our tool, we implemented this decomposition and it is only required to provide the irreducible polynomial for the field $\mathbb{F}_2^k$ and the matrix. From the irreducible polynomial it is possible to deduce the $k \times k$ matrices that represent the elements of $\mathbb{F}_k$ as matrices over $\mathbb{F}_2$. Substituting these matrices in the original matrix over $\mathbb{F}_k$ now creates the $nk \times nk$ binary matrix.

Modular Addition. Modular addition is used as a non-linear component in ARX-ciphers like HIGHT, LEA, and SPECK. We can use the same approach as [SWLW16] to decompose the modular addition into `xor` and `and`. Let z, x, y be n bit-variables with z_i, y_i, x_i as the ith bits, counting from the least significant bit, and $z = x \boxplus y$. The modular addition modulo 2^n is given by:

$$
\begin{aligned}
&z_i = x_i \oplus y_i \oplus c_i \\
&\text{where} \\
&c_i = x_{i-1}y_{i-1} \oplus (x_{i-1} \oplus y_{i-1})c_{i-1} \text{ for } i > 0 \\
&c_0 = 0
\end{aligned}
$$

So far we have assumed that both x, y are variables, however in some ciphers one of them is a constant, e.g. a round key. Since we can ignore `xor` and `and` with a constant we get the following expressions.

$$
\begin{aligned}
&z_i = x_i \oplus c_i \\
&\text{where} \\
&c_i = x_{i-1} \oplus x_{i-1}c_{i-1} \text{ for } i > 0 \\
&c_0 = 0
\end{aligned}
$$

Similar, if we want to find a distinguisher on a cipher like Bel-T or the inverse of an ARX-cipher we also need modular subtraction. To do modular subtraction we can use the fact that

$$x \boxminus y = x \boxplus (-y) = x \boxplus (2^n - y) = x \boxplus ((2^n - 1) - y) \boxplus 1 = x \boxplus \overline{y} \boxplus 1 \tag{4}$$

Since the NOT operation has no effect on whether a bit is balanced or not we can omit it to get $x \boxminus y = x \boxplus y \boxplus 1$. This means that we can do modular subtraction with one modular addition and one constant addition.

3.2 Finding integral distinguishers

In order to find useful integral properties of a cipher, we have to propagate an initial choice vector S^0 and check whether it is impossible to reach certain choice vectors S^r after r rounds. If we can show that an output choice vector that is everywhere zero except for a single **1** in one bit is unreachable, we know that this bit has to be balanced.

In particular we are often interested in whether any bit in the output will be balanced. This corresponds to showing that at least one of the vectors in the set

$$S^r \in \{w \in \mathbb{F}_2^n \mid \mathbf{hw}(w) = 1\}. \tag{5}$$

is unreachable, where $\mathbf{hw}(x)$ is the Hamming weight of the vector.

Contrarily, we can also use this approach to show the absence of a bit-based division property distinguisher in our model. Checking all possible options for the starting choice vector would be (for most primitives) computationally infeasible. Fortunately it is sufficient to show for all starting choice vectors in the set

$$S^0 \in \{w \in \mathbb{F}_2^n \mid \mathbf{hw}(w) = n - 1\}. \tag{6}$$

that all choice vectors in the set in Equation 5 are reachable. This works because the balancedness of the output bits is preserved when we exchange the input choice vector with any vector greater than it (with respect to the above ordering).

We will use the following notation to simplify the description of the distinguishers found later in the paper. The set of *active* bits will be denoted as

$$A = \{i \mid S_i^0 = 1, \;\; i = 0, \ldots, n-1\} \tag{7}$$

and correspondingly the set of *constant* bits as

$$\overline{A} = \overline{\{i \mid S_i^0 = 1,\ \ i = 0, \ldots, n-1\}} = \{i \mid S_i^0 = 0,\ \ i = 0, \ldots, n-1\}. \tag{8}$$

The set of bits which are balanced at the output is denoted as B. We can now describe a distinguisher, for a function f, as

$$A \xrightarrow{f} B. \tag{9}$$

If a valid division trail from A to B exists we will also use the more compact notation $\mathbf{DP}(A) = B$ if the function is clear from context.

Note that while the notation for the set of active bits at the input and the balanced bits at the output looks very similar it conveys a very different meaning in the context of the division property. For a range of bits $s_i, s_{i+1}, \ldots, s_j$ we will use the notation s_{i-j}.

4 Distinguishers and Bounds

We implemented a variety of cryptographic primitives in Solvatore to demonstrate the versatility of our tool and the ease of adding primitives with different design principles.

- **SPN**: Gift, LED, Midori, Photon, Present, Skinny, Spongent
- **ARX**: BelT, ChaCha, Chaskey, LEA, HIGHT, Salsa, Sparx, Speck
- **Feistel**: DES, LBlock, Misty, Roadrunner, Skipjack, SM4, Twine
- **Reflection**: Mantis, Prince, Qarma
- **Bit-sliced**: Ascon, Rectangle
- **LFSR-based**: Bivium, Trivium, Kreyvium

We will first go over the general methodology and after that over the results on the different primitive classes obtained using Solvatore. This includes both bit-based division property distinguishers and finding the number of rounds at which no such distinguisher exists anymore. All results have been obtained on an Intel Core i7-4770S running Ubuntu 17.10 using the Python interface to CryptoMiniSat 5.0.1. Several examples for distinguishers we found are given in section B.

4.1 Methodology

Finding a Bound. As a first step we try to find the number of rounds r^* at which no bit-based division property distinguisher in our model exists. This is done by testing all set of active bits of type

$$A_j = \{i \mid i \in \mathbb{Z}_n \setminus j\} \quad \forall j \in \mathbb{Z}_n\,. \tag{10}$$

This corresponds to all vectors where a single bit is constant. If for all possible choices the set of balanced bits $B_j = \mathbf{DP}(A_j)$ is empty we know that no such distinguisher exists for r^* rounds.

Reducing Data Complexity. In order to reduce the data complexity for the distinguishers covering the most rounds we use different strategies. The naive approach would be to increase the number of constant bits c, try out all possible combinations and check whether the resulting set of balanced bits B is not empty. This might work in some cases however the complexity increases very quickly as we have to test all $\binom{n}{c}$ possible choices.

This can be improved by only testing those combinations of constant bits which can actually lead to non-empty sets B. First, we compute the set of constant bits

$$G_1 = \{j \mid \mathbf{DP}(A_j) = B_j \wedge (|B_j| > 0) \quad \forall j \in \mathbb{Z}_n\} \tag{11}$$

for which at least one of the bits after r rounds is balanced, similar to the case where we try to find the bound. Next, we look at all combinations of two elements of G_1 which share at least one balanced bit

$$G_2 = \{\{i, j\} \mid (i \neq j) \wedge (|\,\mathbf{DP}(A_i) \cap \mathbf{DP}(A_j)|) > 0, \forall i, j \in G_1\}. \tag{12}$$

We can continue the last step in a similar way until G_i is empty by testing all combinations of the sets of bits in G_i repeatedly. Note that in the next step we would not have single indices but sets of indices and we therefore look whether the union of these sets of constant bits lead to a non-empty set B. Another advantage of this approach is that we only need to test those bits for the balancedness property which were already balanced in the last iteration.

In each step the elements in G_i are a set of constant bits which will have at least one balanced bit in the output after r rounds. This approach improves the complexity of finding distinguishers with lower data complexity significantly, but often it is still computationally infeasible to find an optimal distinguisher. For more structured designs it often helps to look at the word level and only look at maximizing the number of constant words as there are fewer combinations which we have to check.

4.2 SPN

Table 1. Results from the optimised search for SPONGENT-88. Combinations are the number of pairs (i, j) in the sets G_i which share bits in their corresponding sets B_i and B_j.

	G_1	G_2	G_3	G_4	G_5
Size ($\lvert G\rvert$)	43	40	25	1	0
Combinations	878	643	234	0	-

We will use 9 rounds of SPONGENT-88 as an example to show the benefits of the optimised search for a distinguisher with lower data complexity. In order to estimate the complexity we will count for how many choice vectors we would have to compute the set of balanced bits B. Using the optimised search we only have to test 1819 choice vectors (see Table 1) to find distinguishers with up to 4 constant bits and exclude any distinguisher with 5 constant bits. Using the naive approach we would have to test 679120 choice vectors to find all distinguishers up to 4 bits and check $\binom{128}{5}$ combinations to exclude the existence of any further distinguishers.

For SKINNY-64 we can find a distinguisher with the same data complexity as the one given by the authors [BJK+16] with one additional balanced bit and show that no distinguishers exist for 11 rounds.

For GIFT-64 we use our optimal approach and no better distinguisher exists. We can find a 9-round distinguisher similar to the one by the authors [BPP+17], but also distinguishers with a lower data complexity. For GIFT-128 finding distinguishers takes significantly longer and we were only able to find a distinguisher with high data complexity similar to the original one.

For several variants of PHOTON we can find distinguishers with low data complexity by searching for combinations of constant words. However for more rounds the search time increases quickly and we are not able to improve any results. The complex linear layer generates a large number of clauses which seems to be the main limiting reason.

Table 2. Overview of our distinguishers and bounds for SPN-based designs.

Cipher	Rounds	Active Bits	Balanced Bits
GIFT-64	9	61	5
	9	62	11
	9	63	30
	10	No Distinguisher	
GIFT-128	11	127	32
	12	No Distinguisher	
LED	5	60	64
	8	No Distinguisher	
Midori-64	6	48	16
	8	No Distinguisher	
Midori-128	5	104	128
PHOTON-100	4	12	100
	5	99	100
PHOTON-144	4	24	144
PHOTON-196	4	28	196
PHOTON-256	4	32	256
Present	9	60	1
	10	No Distinguisher	
Skinny-64	10	48	9
	11	No Distinguisher	
Spongent-88	9	84	3
	9	87	54
	10	No Distinguisher	
Spongent-136	10	132	8
	10	135	93
	11	No Distinguisher	
Spongent-176	12	No Distinguisher	

4.3 ARX

First we look at the permutation used in the Chaskey MAC [MMH+14]. We can find a distinguisher for 3 rounds with only two constant words, one with high complexity for 4 rounds and show that no bit-based division property distinguishers for 5 rounds exist. This confirms the claim by the authors that Chaskey is likely to resist this type of attacks. Considering the construction used for the MAC it seems infeasible to mount an attack based on the 4-round distinguisher.

The large state of Salsa and ChaCha make it difficult to adopt our approach for reducing the data complexity. We therefore keep whole words constants and try to find the maximum number. For 6 rounds of Salsa the only distinguisher which exists keeps the first word constant and the one for ChaCha has only a single constant bit. In both cases no distinguisher exists for 7 rounds. On the actual mode in which Salsa and ChaCha are used as a stream cipher we can only control the 64-bit nonce in a single block. In this setting there are no bit-based division property distinguisher for 4 rounds of Salsa and 2 rounds of ChaCha.

We can also confirm the results from [SWW17] using our optimal search algorithm for Hight, LEA and Speck. We noticed that Solvatore performs significantly better for finding these distinguishers even though we use the same SAT solver. It only took us 28/195/51 seconds compared to 15/30/6 minutes for finding the optimal distinguishers for

Table 3. Overview of our distinguishers and bounds for ARX-based designs.

Cipher	Rounds	Active Bits	Balanced Bits
ChaCha	6	511	138
	7	No Distinguisher	
Chaskey	3	64	6
	4	127	5
	5	No Distinguisher	
LEA	8	126	16
	8	118	1
	9	No Distinguisher	
HIGHT	18	63	2
	19	No Distinguisher	
Salsa	6	480	129
	7	No Distinguisher	
Speck-32	6	31	1
	7	No Distinguisher	
Speck-48	6	45	1
	7	No Distinguisher	
Speck-64	6	61	1
	7	No Distinguisher	
Speck-96	6	93	1
	7	No Distinguisher	
Speck-128	6	125	1
	7	No Distinguisher	
BelT	2	45	5
	3	No Distinguisher	
SPARX-64	3	32	32
	4	No Distinguisher	
SPARX-128	4	96	64
	5	No Distinguisher	

Hight/LEA/Speck. This gap could be explained by the slightly different model resp. using a better search strategy.

Bel-T is a block cipher which has been adopted as a national standard in the Republic of Belarus and combines S-boxes with modular addition. There is only a very limited amount of cryptanalysis available [JP15] (also provides an English description of the algorithm). We provide the first analysis with respect to integral attacks for Bel-T and can find a fairly efficient distinguisher for 2 rounds while showing that none exist for 3 rounds.

In the case of Sparx we can confirm the results by the authors [DPU+16]. The full summary of the results for ARX-based primitives can also be found in Table 3.

4.4 Feistel

For DES we improve the best bit-based division property distinguisher [Tod15b] by one round. The original distinguisher for DES also uses the division property but only word-based which makes this improvement possible.

One of the most successful applications of the division property is the full break of Misty [Tod17]. It is also based on the analysis on the word level so one might suspect that it can be improved by looking at the bit-based division property. We tried to find the

Table 4. Overview of our distinguishers and bounds for Feistel networks.

Cipher	Rounds	Active Bits	Balanced Bits
DES	7	60	8
	8	No Distinguisher	
LBlock	17	63	4
	18	No Distinguisher	
Misty	3	32	64
Roadrunner	5	58	8
	6	No Distinguisher	
SKIPJACK	19($A^8B^8A^3$)	47	16
	20($A^8B^8A^4$)	56	8
	21($A^8B^8A^5$)	No Distinguisher	
Simon32	14	31	16
	15	No Distinguisher	
Simon48	16	47	24
	17	No Distinguisher	
Simon64	18	63	22
	19	No Distinguisher	
Simon96	22	95	5
	23	No Distinguisher	
Simon128	26	127	3
	27	No Distinguisher	
Simeck32	15	31	7
	16	No Distinguisher	
Simeck48	18	47	5
	19	No Distinguisher	
Simeck64	21	63	5
	22	No Distinguisher	
SM4	12	126	32
	13	No Distinguisher	
TWINE	16	63	32
	17	No Distinguisher	

same distinguishers as in the original attack automatically however the complexity seems too high without further optimizations. We could only find a distinguisher for 3 rounds.

The best integral distinguisher on SM4 covers 8 rounds [LJH+07]. By using the bit-based division property we can improve those distinguishers to 12 rounds, although at a high complexity. We further can show that no such distinguishers exist for 13 rounds.

In the case of LBlock we are able to extend the distinguisher found with MILP [XZBL16] by one additional round and for Roadrunner we can find a 5-round distinguisher which also covers one more round than the best known distinguisher [BS15].

For all variants of Simon and Simeck we can reproduce the results from [XZBL16], show that these have the lowest data complexity and that there are no distinguisher in our model for more rounds.

4.5 Reflection

Block ciphers based on the reflection design strategy, introduced by PRINCE, are a popular choice for low-latency designs. We will denote the number of rounds as $f + b$,

Table 5. Results on reflection ciphers.

Cipher	Rounds	Active Bits	Balanced Bits
MANTIS	$2+2$	12	16
	$3+2$	32	16
	$3+3$	No Distinguisher	
PRINCE	$1+1$	12	64
	$2+1$	32	64
	$1+2$	32	64
	$2+2$	No Distinguisher	
QARMA-64/σ_0	$2+2$	48	16
	$3+3$	No Distinguisher	
QARMA-64/σ_1	$2+2$	52	64
	$3+3$	No Distinguisher	
QARMA-64/σ_2	$2+2$	52	64
	$3+3$	No Distinguisher	
QARMA-128/σ_0	$2+2$	96	128
	$3+3$	No Distinguisher	
QARMA-128/σ_1	$2+2$	96	128
	$3+3$	No Distinguisher	
QARMA-128/σ_2	$2+2$	120	128
	$3+3$	No Distinguisher	

Table 6. Results on bit-sliced ciphers.

Cipher	Rounds	Active Bits	Balanced Bits
Ascon	5	16	320
Rectangle	9	60	
	10	No Distinguisher	

where f are the rounds before the middle layer and b the rounds after the middle layer (see Table 5).

For PRINCE we can find a bit-based division property distinguisher with the same complexity as the best higher-order differential given in [RR16] and show that for one additional round none exist. Very similar distinguisher also exist for MANTIS with the only difference being that one can extend those by one round in forward and backwards direction. The distinguishers for QARMA can cover a similar number of rounds although at a much higher data complexity.

4.6 Bit-sliced

In this category we look at two LS-designs (see Table 6). The permutation used in the authenticated encryption scheme Ascon and the block cipher Rectangle. For Ascon we can improve the data complexity of the 5 round distinguisher [Tod15b] by a factor of 4, however for more rounds we could not improve any results as the computations takes too long. For Rectangle we are able to show that no distinguisher exists for 10 rounds and find the already known 9-round distinguisher from [XZBL16].

4.7 LFSR-based

We looked at three LFSR-based stream ciphers which share a similar structure. The active bits are taken over the choice of IV and our distinguishers here checks whether the output

Table 7. Results on LFSR-based stream ciphers.

Cipher	Rounds	Active Bits	Balanced Bits
Bivium	681	79	1
Trivium	707	79	1
Kreyvium	713	127	1

bit of the key stream is balanced after r rounds. It is very likely that there are more bits balanced in the state, but we can only distinguish the key stream if the resulting key stream bit is also balanced.

While we could find some distinguishers the time it takes to find a balanced output bit of the keystream quickly increases and other approaches seem to be more promising for constructing distinguisher based on the division property for this type of ciphers [TIHM17].

4.8 Overview

Using Solvatore we were able to demonstrate several new distinguishers, reduce the data complexity and show at which number of rounds a primitive becomes resistant against bit-based division property. In Figure 1 we give an overview of the number of rounds required before no bit-based division property distinguisher exists in relation to the full number of rounds of the primitive. It can be seen that most ciphers provide a fairly large security margin against these type of attacks and also for many of these designs there are indeed better distinguishers based on other techniques like differential and linear cryptanalysis.

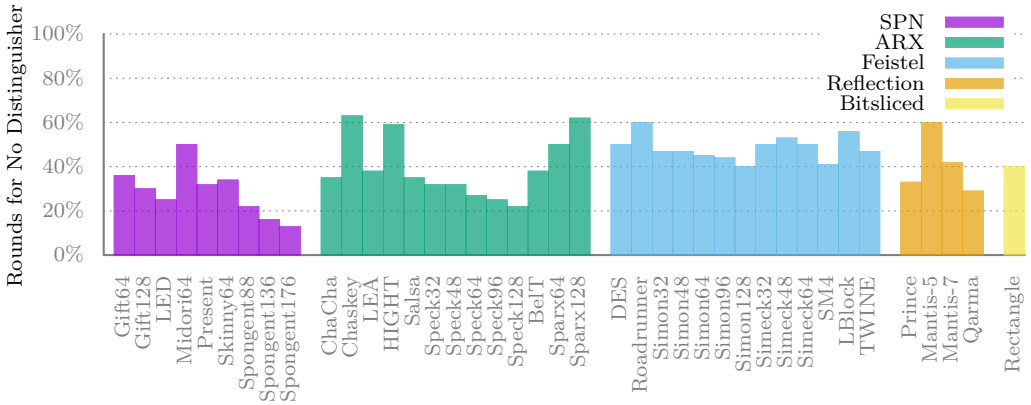

Fig. 1. Overview of the fraction of rounds required before we can show that no bit-based division property distinguishers exist in our model.

The performance of Solvatore varies a lot from the designs and for some it is not feasible to find good distinguishers. For instance we also implemented both AES and Keccak in our tool, but we could only obtain very limited results which could not improve upon the state-of-the-art.

5 Conclusion and Future Work

In this work we presented a new framework to automatically find division property distinguishers for a large class of cryptographic primitives by reducing the problem to SAT. We also provide a cryptanalysis tool implementing this approach, providing a simple way to describe primitives, allowing both designers and cryptanalysts to evaluate cryptographic primitives against this attack vector.

Using this tool we present several new or improved bit-based division property distinguishers for ChaCha, ChasKey, DES, GIFT, LBlock, Mantis, Qarma, RoadRunner, Salsa and SM4.

Furthermore, we provide an improved algorithm for finding distinguisher with an optimal data complexity and show for several primitives that no bit-based division property distinguisher can exist for more rounds.

References

BJK+16. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016. 10

BPP+17. Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 321–345. Springer, 2017. 2, 10

BS01. Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001. 3

BS10. Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. *Journal of Cryptology*, 23(4):505–518, 2010. 1

BS15. Adnan Baysal and Sühap Sahin. Roadrunner: A small and fast bitslice block cipher for low cost 8-bit processors. In *LightSec*, volume 9542 of *Lecture Notes in Computer Science*, pages 58–76. Springer, 2015. 2, 13

DKR97a. Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *Fast Software Encryption, FSE '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997. 1

DKR97b. Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *Fast Software Encryption, FSE '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997. 3

DMP+15. Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 733–761. Springer, 2015. 2

DPU+16. Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design strategies for ARX with provable bounds: Sparx and LAX. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 484–513, 2016. 12

DS09. Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009. 2

DS11. Itai Dinur and Adi Shamir. Breaking grain-128 with dynamic cube attacks. In Antoine Joux, editor, *Fast Software Encryption, FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 167–187. Springer, 2011. 2

JK97. Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Eli Biham, editor, *Fast Software Encryption, FSE '97*, volume 1267 of *Lecture Notes in Computer Science*, pages 28–40. Springer, 1997. 1

JP15. Philipp Jovanovic and Ilia Polian. Fault-based attacks on the bel-t block cipher family. In *DATE*, pages 601–604. ACM, 2015. 12

Knu95. Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop. 1994*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1995. 1

KW02. Lars R. Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption,FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2002. 1

Lai94. Xuejia Lai. Higher order derivatives and differential cryptanalysis. In Richard E. Blahut, Jr. Daniel J. Costello, Ueli Maurer, and Thomas Mittelholzer, editors, *Communications and Cryptography, Two Sides of One Tapestry*, pages 227–233. Kluwer Academic Publishers, 1994. 1, 3

LJH+07. Fen Liu, Wen Ji, Lei Hu, Jintai Ding, Shuwang Lv, Andrei Pyshkin, and Ralf-Philipp Weinmann. Analysis of the SMS4 block cipher. In *ACISP*, volume 4586 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2007. 2, 13

MMH+14. Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In *Selected Areas in Cryptography*, volume 8781 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2014. 11

RR16. Shahram Rasoolzadeh and Håvard Raddum. Faster key recovery attack on round-reduced PRINCE. In *LightSec*, volume 10098 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2016. 14

SGL+17. Siwei Sun, David Gerault, Pascal Lafourcade, Qianqian Yang, Yosuke Todo, Kexin Qiao, and Lei Hu. Analysis of aes, skinny, and others with constraint programming. *IACR Transactions on Symmetric Cryptology*, 2017(1), 2017. 3

SWLW16. Ling Sun, Wei Wang, Ru Liu, and Meiqin Wang. Milp-aided bit-based division property for arx-based block cipher. Cryptology ePrint Archive, Report 2016/1101, 2016. http://eprint.iacr.org/2016/1101. 5, 7

SWW16. Ling Sun, Wei Wang, and Meiqin Wang. Milp-aided bit-based division property for primitives with non-bit-permutation linear layers. *IACR Cryptology ePrint Archive*, 2016:811, 2016. 3

SWW17. Ling Sun, Wei Wang, and Meiqin Wang. Automatic search of bit-based division property for arx ciphers and word-based division property. Cryptology ePrint Archive, Report 2017/860, 2017. https://eprint.iacr.org/2017/860. 2, 3, 5, 11

TIHM17. Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In *CRYPTO (3)*, volume 10403 of *Lecture Notes in Computer Science*, pages 250–279. Springer, 2017. 15

TM16. Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, *Fast Software Encryption, FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 357–377. Springer, 2016. 1, 2

Tod15a. Yosuke Todo. Integral cryptanalysis on full MISTY1. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 413–432. Springer, 2015. 1, 2, 6

Tod15b. Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015. 1, 2, 3, 6, 12, 14

Tod17. Yosuke Todo. Integral cryptanalysis on full MISTY1. *J. Cryptology*, 30(3):920–959, 2017. 12

Vie07. Michael Vielhaber. Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack. *IACR Cryptology ePrint Archive*, 2007:413, 2007. 2

XZBL16. Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016. 2, 5, 13, 14

ZR17. Wenying Zhang and Vincent Rijmen. Division cryptanalysis of block ciphers with a binary diffusion layer. Cryptology ePrint Archive, Report 2017/188, 2017. https://eprint.iacr.org/2017/188. 2

A Implementation of Present

The following example shows how one can implement the PRESENT cipher in our framework to analyse its properties against bit-based division property attacks.

```
from cipher_description import CipherDescription

present_sbox = [0xC, 0x5, 0x6, 0xB, 0x9, 0x0, 0xA, 0xD,
                0x3, 0xE, 0xF, 0x8, 0x4, 0x7, 0x1, 0x2]
present_permutations = [\
  ['s1', 's16', 's4'], ['s2', 's32', 's8'], ['s3', 's48', 's12'],
  ['s5', 's17', 's20'],['s6', 's33', 's24'], ['s7', 's49', 's28'],
  ['s9', 's18', 's36'], ['s10', 's34', 's40'], ['s11', 's50', 's44'],
  ['s13', 's19', 's52'], ['s14', 's35', 's56'], ['s15', 's51', 's60'],
  ['s22', 's37', 's25'], ['s23', 's53', 's29'], ['s26', 's38', 's41'],
  ['s27', 's54', 's45'], ['s30', 's39', 's57'], ['s31', 's55', 's61'],
  ['s43', 's58', 's46'], ['s47', 's59', 's62']]

present = CipherDescription(64)
present.add_sbox('S-box', present_sbox)
for i in range(16):
    bits = ["s{}".format(4*i + 0),
            "s{}".format(4*i + 1),
            "s{}".format(4*i + 2),
            "s{}".format(4*i + 3)]
    present.apply_sbox('S-box', bits, bits)
for p in present_permutations:
    present.apply_permutation(p)
```

Using this description of the PRESENT block cipher we can mount our analysis. The following code checks whether no bit-based division property distinguisher exists for 10 rounds of PRESENT.

```
from itertools import combinations
from solvatore import Solvatore
from cipher_description import CipherDescription
from ciphers import present

cipher = present.present
rounds = 10

solver = Solvatore()
solver.load_cipher(cipher)
solver.set_rounds(rounds)

# Look over all combination for one non active bit
for bits in combinations(range(64), 1):
    nonactive_bits = bits
    active_bits = {i for i in range(64) if i not in nonactive_bits}

    # Find all balanced bits
    balanced_bits = []
    for i in range(cipher.state_size):
        if solver.is_bit_balanced(i, rounds, active_bits):
            balanced_bits.append(i)

    if len(balanced_bits) > 0:
        print("Found distinguisher!")
        print(active_bits, balanced_bits)
```

B Overview of Distinguishers

In the following we list some of the new distinguishers we found.

B.1 ChaCha

$$\overline{\{0\}} \xrightarrow{\text{6-round}} \{32-68, 192-223, 352-415, 424-428\} \tag{13}$$

B.2 Chaskey

$$\overline{\{64-127\}} \xrightarrow{\text{3-round}} \{80-85\} \tag{14}$$

$$\overline{\{96\}} \xrightarrow{\text{4-round}} \{80-81\} \tag{15}$$

B.3 DES

$$\overline{\{50-52,63\}} \xrightarrow{\text{7-round}} \{0,3,9,10,18,19,25,28\} \tag{16}$$

B.4 GIFT-64

$$\overline{\{0-2\}} \xrightarrow{\text{9-round}} \{3,7,27,43,59\} \tag{17}$$

B.5 LBlock

$$\overline{\{34\}} \xrightarrow{\text{17-round}} \{2,3,30,31\} \tag{18}$$

B.6 Mantis

$$\overline{\{0-7,16-23,40-47,56-63\}} \xrightarrow{\text{3 + 2 rounds}} \begin{aligned}\{&2,6,10,14,18,22,26,30\\ &34,38,42,46,50,54,58,62\}\end{aligned} \tag{19}$$

B.7 QARMA

QARMA-64/σ_0

$$\overline{\{0-3,20-23,40-43,60-63\}} \xrightarrow{\text{2 + 2 rounds}} \begin{aligned}\{&1,5,9,13,17,21,25,29,\\ &33,37,41,45,49,53,57,61\}\end{aligned} \tag{20}$$

QARMA-64/σ_1

$$\overline{\{0-3,20-23,40-43\}} \xrightarrow{\text{2 + 2 rounds}} \{0-63\} \tag{21}$$

QARMA-64/σ_2

$$\overline{\{0-3,20-23,40-43\}} \xrightarrow{\text{2 + 2 rounds}} \{0-63\} \tag{22}$$

QARMA-128/σ_0

$$\overline{\{0-15,32-47\}} \xrightarrow{\text{2 + 2 rounds}} \{0-127\} \tag{23}$$

QARMA-128/σ_1

$$\overline{\{0-15,32-47\}} \xrightarrow{\text{2 + 2 rounds}} \{0-127\} \tag{24}$$

QARMA-128/σ_2

$$\overline{\{0-7\}} \xrightarrow{\text{2 + 2 rounds}} \{0-127\} \tag{25}$$

B.8 RoadRunner

$$\overline{\{0,1,8,9,16,17\}} \xrightarrow{\text{5-round}} \{32,33,40,41,48,49,56,57\} \tag{26}$$

B.9 Salsa

$$\overline{\{0-31\}} \xrightarrow{\text{6-round}} \{128-255,295\} \tag{27}$$

B.10 SM4

$$\overline{\{96,97\}} \xrightarrow{\text{12-round}} \{0-31\} \tag{28}$$