IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2020, No. 4, pp. 173–194. DOI:10.46586/tosc.v2020.i4.173-194

Increasing Precision of Division Property

Patrick Derbez* and Pierre-Alain Fouque

Univ Rennes, Centre National de la Recherche Scientifique (CNRS), Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, France
{patrick.derbez,pierre-alain.fouque}@irisa.fr

Abstract. In this paper we propose new techniques related to division property. We describe for the first time a practical algorithm for computing the propagation tables of 16-bit Super-Sboxes, increasing the precision of the division property by removing a lot of false division trails. We also improve the complexity of the procedure introduced by Lambin *et al.* (Design, Codes and Cryptography, 2020) to extend a cipher with linear mappings and show how to decrease the number of transitions to look for. While search procedures for integral distinguishers most often rely on MILP or SAT solvers for their ease of programming the propagation constraints, such generic solvers can only handle small 4/8-bit Sboxes. Thus we developed an ad-hoc tool handling larger Sboxes and all the improvements described in the paper. As a result, we found new integral distinguishers on `SKINNY-64`, `HIGHT` and `Midori-64`.

Keywords: Division property · SKINNY · Midori · HIGHT · Tools

1 Introduction

Integral cryptanalysis exploits distinguishers computing the sum of ciphertexts corresponding to a set of plaintexts spanning a linear subspace. This technique was originally introduced by Knudsen in [DKR97] as a specific attack against the byte-oriented structure of the block cipher `SQUARE`. In 2000, Ferguson *et al.* [FKL+00] presented at FSE powerful attacks based on integral distinguishers against round-reduced versions of `AES`, named *Partial Sum attacks*. In particular they described a practical attack against 6 rounds which is still one of the best known attacks against `AES`. Integral distinguishers were found by propagating through the round functions simple properties on words composing the internal states: ALL (the word takes all the possible values once), BALANCED (the word sums to zero), CONSTANT (the value of the word is constant).

The so-called division property, introduced by Todo at Eurocrypt'15 [Tod15], is a method to find more sophisticated integral distinguishers. The idea behind the division property technique is actually quite simple. Let f and g be two n-bit functions and assume the goal is to find an integral distinguisher on $g \circ f$ without computing it explicitly. Let $y_i = f_i(x_0, \ldots, x_{n-1})$ and $z_i = g_i(y_0, \ldots, y_{n-1})$ be the intermediate and final expressions of the coordinate functions of f and of g, and let m_z be a monomial in the z_i's, and so m_z is a polynomial in some m_y monomials. Division property actually captures that if for a subset $\mathcal{X}$ of $\mathbb{F}_2^n$ each monomial m_y appearing in m_z satisfies $\bigoplus_{x\in\mathcal{X}} m_y(x) = 0$ then $\bigoplus_{x\in\mathcal{X}} m_z(x) = 0$. Several variants of this property were used to find integral distinguishers. For instance, in [TM16], Todo and Morii used that if all monomials m_y but one sum to zero then $\bigoplus_{x\in\mathcal{X}} m_z(x) = 1$. And more recently, in both [HLM+20] and [HLLT20], the exact relation was used: $\bigoplus_{x\in\mathcal{X}} m_z(x) = 0$ if and only if the number of monomials m_y for which $\bigoplus_{x\in\mathcal{X}} m_y(x) = 1$ is even.

*Patrick Derbez and Pierre-Alain Fouque were supported by the French Agence Nationale de la Recherche through the CryptAudit project under Contract ANR-17-CE39-0003.

Received: 2020-09-01 Accepted: 2020-11-01 Published: 2020-12-10

In practice we cannot try all possible sets $\mathcal{X}$ nor compute the corresponding sums for all monomials involved in the description of a cryptographic primitive. Furthermore we typically want integral distinguishers independent from the key, adding an extra complexity to the problem. However it is easy to show that if P is a polynomial in variables $(x_1, \ldots, x_n)$ then $\bigoplus_{(x_1,\ldots,x_i)\in\mathbb{F}_2^i} P(x_1, \ldots, x_n) = 0$ for each value of $(x_{i+1}, \ldots, x_n)$ if and only if P does not involve a monomial containing all the variables $x_1, \ldots, x_i$. This property can be understood more easily using higher-order differential and means that if we derive i times w.r.t. to the i first variables, a multivariate polynomial P that does not contain a monomial involving the $x_1 x_2 \ldots x_i$ monomial, then we get the 0 polynomial. Thus integral distinguishers are highly related to the maximal monomials involved in a polynomial and division property can be seen as a method to track them through an iterated function.

Searching for integral distinguishers. The main difficulty is to efficiently modelize the propagation of division property through the round functions of a cipher. Except in [TM16] where Todo and Morii used an ad-hoc tool to exhaust division trails on `SIMON`-32, searching for integral distinguishers usually relies on generic solvers for MILP, SAT or SMT models. In [XZBL16] Xiang *et al.* show that it is possible to describe transitions through small Sboxes with inequalities by computing the convex hull of points. This work has been extended by Zhang and Rijmen [ZR19] to binary linear mapping. Eskandari *et al.* in [EKKT18] have built a tool called Solvatore to find such division property trails using a SAT solver and found many new integral distinguishers. The difficulty of the search procedure depends on the cipher and on the variant of division property implemented. The original variant is the simplest to search for but is also the less accurate as it may miss some cancellations of monomials and thus miss distinguishers. In [HLLT20], Hebborn *et al.* worked with the exact variant and described a new method dedicated to (small) block cipher aiming at proving that for each linear combination of the ciphertext bits and for each degree $n-1$ monomial in the plaintexts bits, there is at least one key (considering independent round keys) for which the monomial appears in the ANF of the linear combination. They used a heuristic approach to find round keys for which evaluating the parity of division trails is the cheapest. As a result they found that 13-round `SKINNY-64`, 11-round `Gift` and 11-round `PRESENT` are all immune to integral distinguishers if considering independent round keys.

Our Contributions.

In this paper, our contributions are three-fold.

i) Our main idea is to increase the precision of the original division property. To this end, we want to handle larger Sboxes than the typical 4/8-bit Sboxes block ciphers are usually composed of. More precisely, we want to handle Super-Sboxes to cover two layers of Sboxes in one operation. Hence, in this paper we propose a new algorithm to compute the so-called *propagation table* associated to a Super-Sbox. Our algorithm computes the propagation table corresponding to a collection of k n-bit permutations in $\mathcal{O}(nk2^{2n} + 2^{3n})$ simple operations while applying k times the classical algorithm would lead to a complexity in $\mathcal{O}(k2^{3n})$.

ii) MILP and SAT solvers seem to be unable to efficiently handle such large propagation tables and we decided to implement an ad-hoc tool to this end, based on a classical branch-and-bound. To the best of our knowledge, this is the first time an ad-hoc approach can practically search for division trails on 64-bit block ciphers without relying on generic solvers. It is well-known that MILP and SAT solvers make easier the development of tools, but they also have caveats such as it is hard to predict their running time and it is also hard to reverse engineer the algorithmic techniques they used to speedup the running time. For cryptanalysts, such generic tools give a first

Table 1: Summary of the best integral distinguishers found by our tool. Note that in our case, a distinguisher against a cipher E means we found two linear mappings L_{in} and L_{out} and a distinguisher against $L_{out} \circ E \circ L_{in}$.

Cipher	Rounds	Data	Ref.
Midori-64	6	2^{48}	[EKKT18]
		2^{15}	6.1
	7	2^{63}	[ZR19]
		2^{45}	6.1
	9	2^{63}	6.1
SKINNY-64	8	2^{15}	6.2
	10	2^{48}	[EKKT18]
		2^{47}	6.2
	11	2^{63}	6.2
HIGHT	19	2^{63}	[FTIM19]
	20	2^{63}	6.3

good approximation or solution, but it is always interesting to better understand how they have been able to find the solutions.

iii) We also provide several new algorithms which may improve all previous models related to division property. First we show how to remove some unnecessary elements from a chain of propagation tables describing a cipher. This restricts the search space and decreased the running time of our tool up to factor 3. We provide as well new algorithms to add linear mappings around the cipher to extend the search space of division trails. Indeed, contrary to differential or linear cryptanalysis, integral division property attack are not invariant under linear mapping and Lambin *et al.* in [LDF20] have shown that considering linear mappings at the beginning and end of the cipher may allow to find integral distinguishers covering more rounds. In particular our new algorithms have a much better time complexity than the ones introduced in [LDF20] since for an m-bit Sbox we only have to consider 2^m mappings while Lambin *et al.* had to consider $O\left(2^{m^2}\right)$ of them.

As a result, we found new integral distinguishers against the three blockciphers `SKINNY-64` [BJK+16], `Midori-64` [BBI+15] and `HIGHT` [HSH+06], increasing the number of rounds covered compared to previously best known integral distinguishers. We also experimentally verified some distinguishers found on smaller instances in order to validate our tool. For instance, we searched for low data distinguishers by fixing some input bits of the Super-Sboxes to constant and we found an integral distinguishers requiring only 2^{15} chosen plaintexts against both 8-round `SKINNY-64` and 6-round `Midori`-64. All the results found by our tool are given in Table 1.

The C++ code of our ad-hoc tool is available at

`https://gitlab.inria.fr/pderbez/divlin`

For now it handles any 64-bit function which can be written as $f_R \circ \sigma \circ \ldots \circ \sigma \circ f_0$ where σ is a permutation operating at the nibble level and where the f_i's are parallel applications of four 16-bit to 16-bit functions, eventually depending on a round key. This includes a large set of ciphers as `TWINE`, `Gift`, `HIGHT`, ...

Organization of the paper.

Section 2 contains the notations and definitions and Section 3 some related works. In Section 4 we present new techniques, including an algorithm to compute the propagation table of a Super-Sbox. In Section 5 we describe our ad-hoc tool to search for integral distinguishers. Finally, Section 6 contains our new results against the block ciphers `SKINNY-64`, `Midori-64` and `HIGHT`.

2 Preliminaries

In this section, we give the notations and definitions we will use in this paper. We also introduce division property based distinguishers on block cipher.

2.1 Notations and Definitions

We denote $\boldsymbol{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{F}_2^n$ an n-bit vector, where x_0 is the least significant bit and will often write $x_0 x_1 \ldots x_{n-1}$ instead of $(x_0, \ldots, x_{n-1})$. There is a trivial mapping from n-bit vectors to monomials in variables $(X_0, \ldots, X_{n-1})$ and we will often refer to $\boldsymbol{x}$ as a monomial.

Definition 1. We say that a monomial $\boldsymbol{m_0}$ contains a monomial $\boldsymbol{m_1}$ if and only if all the variables of $\boldsymbol{m_1}$ belong to $\boldsymbol{m_0}$, i.e. if and only if $\boldsymbol{m_1}$ is a divisor of $\boldsymbol{m_0}$. In that case we will write $\boldsymbol{m_1} \preceq \boldsymbol{m_0}$. For instance, $x_0 \preceq x_0 x_1$ but $x_2 \not\preceq x_0 x_1$.

Definition 2. Given a set s of monomials, we denote by $\max(s)$ (resp. $\min(s)$) the set of the maximal (resp. minimal) monomials of s.

Note that given a set s of n monomials, building $\max(s)$ (resp. $\min(s)$) requires at most $n \times |\max(s)|$ (resp. $n \times |\min(s)|$) comparisons and is upper bounded by n^2. Furthermore, both operators min and max can be easily extended to polynomials over $\mathbb{F}_2$ since they can be seen as set of monomials.

Definition 3 (Bit-product)**.** For $\boldsymbol{x}, \boldsymbol{u} \in \mathbb{F}_2^n$, we denote by $\boldsymbol{x}^{\boldsymbol{u}}$ the bit product

$$\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=0}^{n-1} x_i^{u_i}.$$

Definition 4 (Bit-based Division Property [TM16])**.** A set $\mathbb{X} \subset \mathbb{F}_2^n$ has the division property $D_{\mathbb{K}}^n$, where $\mathbb{K} \subset \mathbb{F}_2^n$ is a set, if for all $\boldsymbol{u} \in \mathbb{F}_2^n$, we have

$$\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = \begin{cases} unknown & \text{if there is } \boldsymbol{k} \in \mathbb{K} \text{ s.t. } \boldsymbol{u} \succeq \boldsymbol{k} \\ 0 & \text{otherwise} \end{cases}$$

2.2 Integral Distinguishers

Basically, the division property is a tool to track the monomials through the successive applications of a round function. Given a block cipher, let $P_b(X_0, \ldots X_{n-1}, K_0, \ldots, K_{m-1})$ be the polynomial describing the b-th bit of the ciphertext as a function of the plaintext (X) and the master key (K). If no monomial greater than or equal to $X_0 X_1 \ldots X_{i-1}$ appears in P_b then for any value $\boldsymbol{y}$ of $(X_i, \ldots, X_{n-1}, K_0, \ldots, K_{m-1})$ we have that

$$\bigoplus_{\boldsymbol{x} \in \{0,1\}^i} P_b(\boldsymbol{x}, \boldsymbol{y}) = 0,$$

which is a property a random function should not have. However, in practice we cannot computationally obtain the polynomial expression of all the bits of the ciphertext because the number of terms is too huge. Hopefully the division property tackles down this problem. Let f and g be two n-bit functions and let $y_i = f_i(x_0, \ldots, x_{n-1})$ and $z_i = g_i(y_0, \ldots, y_{n-1}) = g_i \circ f(x_0, \ldots, x_{n-1})$ be the intermediate and final expressions of the coordinate functions of f and g respectively. Division property captures that if for all monomials $\boldsymbol{y}^{\boldsymbol{v}}$ appearing in $\boldsymbol{z}^{\boldsymbol{u}}$, $\boldsymbol{y}^{\boldsymbol{v}}$ does not involve a monomial greater than $\boldsymbol{x}^{\boldsymbol{w}}$ then $\boldsymbol{z}^{\boldsymbol{u}}$ (now seen as a function of the x_i's) does not involve a monomial greater than $\boldsymbol{x}^{\boldsymbol{w}}$ too. Hence, a common way to study division property for a block cipher is to study the *division trails* of this cipher, which show the propagation of the division property through the basic operations composing the block cipher.

Definition 5 (Division Trails [XZBL16])**.** Let f denote the round function of an iterated block cipher. Assume the input set to the block cipher has initial division property $D^n_{\{\boldsymbol{k}\}}$, and denote the division property after propagating through i rounds of the block cipher (i.e. i applications of f) by $D^n_{\mathbb{K}_i}$. Thus, we have the following chain of division property propagations :

$$\{\boldsymbol{k}\} \triangleq \mathbb{K}_0 \xrightarrow{f} \mathbb{K}_1 \xrightarrow{f} \mathbb{K}_2 \xrightarrow{f} \ldots \xrightarrow{f} \mathbb{K}_r.$$

Moreover, for any vector $\boldsymbol{k}_i$ in $\mathbb{K}_i$ ($i \geq 1$), there must exist a vector $\boldsymbol{k}_{i-1}$ in $\mathbb{K}_{i-1}$ such that $\boldsymbol{k}_{i-1}$ can propagate to $\boldsymbol{k}_i$ by the division property propagation rules, i.e. $f^{\boldsymbol{k}_i}$ contains a monomial $\boldsymbol{m}$ such that $\boldsymbol{m} \succeq \boldsymbol{k}_{i-1}$. Furthermore, for $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r$, if $\boldsymbol{k}_{i-1}$ can propagate to $\boldsymbol{k}_i$ for all $i \in \{1, 2, \ldots, r\}$, $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r)$ is called an r-round division trail.

In the rest of the paper, we will denote $\boldsymbol{k} \xrightarrow{f} \boldsymbol{k'}$ if the vector $\boldsymbol{k} \in \mathbb{F}_2^n$ can propagate to a vector $\boldsymbol{k'} \in \mathbb{F}_2^m$ through the n-bit to m-bit function f.

Proposition 1 ([XZBL16])**.** *Assume $\mathbb{X}$ is a set with division property $D^n_{\mathbb{K}}$, then $\mathbb{X}$ does not have integral property if and only if $\mathbb{K}$ contains all the n unit vectors. As a result, if the i-th unit vector does not belong to $\mathbb{K}$, then the i-th bit is balanced.*

This proposition is the core of the division property technique. Given an n-bit to n-bit function f and $\boldsymbol{x} \in \mathbb{F}_2^n$, if there is no division trail through f from $\boldsymbol{x}$ to the i-th unit vector then it means that monomial $\boldsymbol{x}$ does not divide any of the monomials involved in the anf of the i-th coordinate function f_i and thus there is an integral distinguisher on f_i.

3 Related Works

In this section we recall some previous works our paper is based on.

3.1 Propagation Table

At ASIACRYPT'16, Xiang *et al.* [XZBL16] proposed an algorithm to compute the *propagation table* of an n-bit to n-bit function f. The propagation table of f is a table T such that for any $\boldsymbol{m} \in \mathbb{F}_2^n$, $T[\boldsymbol{m}]$ contains all possible monomials $\boldsymbol{m'}$ such that the transition $\boldsymbol{m} \xrightarrow{f} \boldsymbol{m'}$ is valid, fully describing the propagation rules through f. In other words, the propagation table of a function f is a table mapping any monomial $\boldsymbol{m}$ to the list of vectors $\boldsymbol{u} \in \{0,1\}^n$ such that the polynomial $f^{\boldsymbol{u}}$ contains a monomial $\boldsymbol{m'}$ satisfying $\boldsymbol{m} \preceq \boldsymbol{m'}$.

The algorithm building the propagation table is quite simple. It computes all the product $f^{\boldsymbol{u}}$, finds all the monomials included in at least one monomial of $f^{\boldsymbol{u}}$ and adds $\boldsymbol{u}$ to the corresponding lines. At the end, each line of the table is reduced based on the fact that minimal monomials are sufficient to characterize the division property.

Algorithm 1: Building propagation table

```
Data: an n-bit to n-bit function f
Result: T the propagation table associated to f
init T as empty
foreach u ∈ {0,1}^n do                                          // 2^n
    compute the anf of f^u                                      // O(n2^n)
    foreach monomial m contained in a monomial of f^u do        // O(2^{2n})
        T[m] = T[m] ∪ {u}
    end
end
foreach m ∈ {0,1}^n do                                          // 2^n
    T[m] = min(T[m])                                            // O(2^{2n})
end
return T
```

The exact complexity of the algorithm to produce the propagation table is hard to compute, as it depends on the function f and more precisely on the number of terms each coordinate function is composed of. In the worst case it is $\mathcal{O}(2^{3n})$ simple operations. It is depicted in Algorithm 1.

3.2 Linear Mappings at Input and Output

In [LDF20], Lambin *et al.* show that for a given block cipher E, we should consider $L_{out} \circ E \circ L_{in}$, where both L_{out} and L_{in} are linear mappings, since division property is not linearly invariant. This may lead to new distinguishers but the drawback is that the search space is greatly increased. For instance, let f_k be the encryption function

$$f_k(x, y) = (p_0(k)x \oplus p_1(k)y, p_2(k)x \oplus p_3(k)y)$$

where $p_0, \ldots, p_3$ are non-zero polynomials. In that case classical application of division property would conclude that no output bit is balanced. But if either $p_0 = p_2$ or $p_1 = p_3$ then the xor of both output bits is balanced.

The idea proposed by Lambin *et al.* is, given an n-bit function f, to generate all the possible invertible $n \times n$ matrices and to compose them with f. Then all the corresponding propagation tables are built. From there, several matrices may lead to the same propagation table and so one may consider classes of equivalence to reduce the search space. However, the number of invertible matrices is around $\mathcal{O}(2^{n^2})$, and it seems very complicated to go much further than $n = 6$. Hence they restrict themselves to cases where the linear mapping L_{in} (resp. L_{out}) is applied in front (resp. back) of a 4-bit sbox.

As a result, they show that 10-round `RECTANGLE` [ZBL+15] can be distinguished while previous best known distinguishers could not target more than 9 rounds.

Remark. Actually they do not run $2^{n^2} \times 2^{n^2} = 2^{2n^2}$ times the search procedure. For each invertible matrix they combine it to the sbox and compute its propagation table. Then if two matrices lead to the same propagation table they only have to try one of them since both would lead to the same result.

4 Advanced Division Property Search

In this section we present our new ideas to improve division property based search procedures for integral distinguishers.

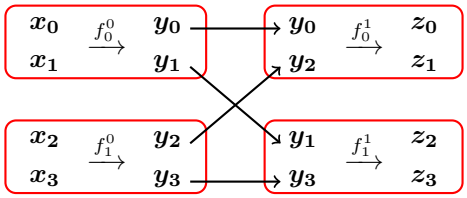
4.1 Reducing Propagation Tables

Since searching for integral distinguisher boils down to exhausting all the possible trails which reach a unit vector, its complexity directly depends on the number of possible trails. We saw in previous sections that if both trails $\boldsymbol{m_0} \xrightarrow{f} \boldsymbol{m_1}$ and $\boldsymbol{m_0} \xrightarrow{f} \boldsymbol{m'_1}$ are valid and if $\boldsymbol{m'_1} \preceq \boldsymbol{m_1}$, then we can consider only the second one. This property allows to reduce the number of elements stored in the propagation tables and to decrease the number of possible trails to try in order to find integral distinguishers. Note that this property is *local* to a function. However, in practice, we search for trails through the composition of many functions and we propose to go further.

Let $f = f^1 \circ \sigma \circ f^0$ be a $4n$-bit function where for $i \in \{0, 1\}$ we have:

- $f^i(a_0, a_1, a_2, a_3) = (f_0^i(a_0, a_1), f_1^i(a_2, a_3))$, f_0^i and f_1^i being two $2n$-bit functions,
- σ the permutation $\sigma(a_0, a_1, a_2, a_3) = (a_0, a_2, a_1, a_3)$.

Our objective is to study division trails through f:



Our idea is to remove unnecessary elements from both propagation tables $T_{f_0^0}$ and $T_{f_1^0}$ of f_0^0 and f_1^0 respectively. Let Y_2 and Y_3 be the sets containing all the possible values for $\boldsymbol{y_2}$ and $\boldsymbol{y_3}$ respectively as output of $T_{f_1^0}$ and let $(\boldsymbol{y_0}, \boldsymbol{y_1})$ and $(\boldsymbol{y'_0}, \boldsymbol{y'_1})$ be two outputs of $T_{f_0^0}$. We say that $(\boldsymbol{y_0}, \boldsymbol{y_1})$ is *smaller* than $(\boldsymbol{y'_0}, \boldsymbol{y'_1})$ if and only if both the following conditions are satisfied:

- for all $\boldsymbol{y_2} \in Y_2$, for all $\boldsymbol{u'} \in T_{f_0^1}[\boldsymbol{y'_0} \parallel \boldsymbol{y_2}]$, there is $\boldsymbol{u} \in T_{f_0^1}[\boldsymbol{y_0} \parallel \boldsymbol{y_2}]$ such that $\boldsymbol{u} \preceq \boldsymbol{u'}$
- for all $\boldsymbol{y_3} \in Y_3$, for all $\boldsymbol{u'} \in T_{f_1^1}[\boldsymbol{y'_1} \parallel \boldsymbol{y_3}]$, there is $\boldsymbol{u} \in T_{f_1^1}[\boldsymbol{y_1} \parallel \boldsymbol{y_3}]$ such that $\boldsymbol{u} \preceq \boldsymbol{u'}$

This means that for all trails going through $(\boldsymbol{y'_0}, \boldsymbol{y'_1})$, there is a trail going through $(\boldsymbol{y_0}, \boldsymbol{y_1})$ reaching a smaller output after f^1. Hence the propagation table of f_0^0 can be reduced using this (partial) order by keeping only the minimal elements on each line.

This order can be easily extended recursively to more rounds added at the beginning. Constructing the sets Y_i is free as it can be done while constructing the propagation tables. Then at each step we only need to remember whether $\boldsymbol{y_i}$ is *smaller* than $\boldsymbol{y'_i}$ or not.

In practice we found this technique to be very efficient to remove elements in the propagation tables. For instance on `SKINNY`-64 using this technique decreases the running time of our tool up to a factor 3.

4.2 Larger Tables and Super-Sboxes

For many SPN-based block ciphers, the internal state is a 4×4 matrix of *cell* (typically 4 or 8 bits) and the round function is the composition of:

- a **SubCells (SC)** operation, applying the same Sbox to each cell independently;
- a **MixColumns (MC)** operation, applying a linear transformation on each column independently;
- a **AddRoundKey (ARK)** operation, xoring the round key to the internal state;

- a **CellsPerm (CP)** operation, permuting the cells of the internal state.

Typically, the search of integral distinguishers is done by exhausting trails going through each layer successively. But given an n-bit function, Algorithm 1 produces the propagation table in roughly $\mathcal{O}(2^{3n})$ operations which is practical up to $n \approx 16$. Hence it seems possible to improve the precision of the propagation by considering 16-bit Sboxes, and more precisely Super-Sboxes. Since all operations except the cell permutation act on column and since $\mathbf{SC} \circ \mathbf{CP} = \mathbf{CP} \circ \mathbf{SC}$, two rounds can be rewritten as $\mathbf{CP} \circ \mathbf{ARK} \circ \mathbf{MC} \circ \mathbf{CP} \circ \mathbf{SSC}$ where **SSC** acts on each column independently.

Our idea is to build the propagation table for each of the 4 parts of the **SSC** operation. Because of the key addition between the two layers of Sboxes, a naive approach would require to run Algorithm 1 for all possible values of the (part of) round key used in the Super-Sbox and then merge the propagation tables. This would quickly make the computation untractable. Instead we propose a new version of Algorithm 1, taking as input a collection of k n-bit functions and outputing the propagation table containing all the valid transitions for at least one of the function. This is described in Algorithm 2 and the time complexity of this algorithm is in $\mathcal{O}(kn2^{2n} + 2^{3n})$. Note that typical value for k is 2^n and so our algorithm has complexity $\mathcal{O}(n2^{3n})$, to be compared to $\mathcal{O}(2^{4n})$, the cost of calling 2^n times Algorithm 1.

Algorithm 2: Building propagation table of a collection of functions

Data: a collection F of k n-bit to n-bit functions
Result: T the propagation table associated to F
init T as empty
foreach $f \in F$ **do** // k
  foreach $\boldsymbol{u} \in \{0,1\}^n$ **do** // 2^n
    compute the anf of $f^{\boldsymbol{u}}$ // $\mathcal{O}(n2^n)$
    foreach *monomial* $\boldsymbol{m}$ *of* $f^{\boldsymbol{u}}$ **do** // $\mathcal{O}(2^n)$
      $T[\boldsymbol{m}] = T[\boldsymbol{m}] \cup \{\boldsymbol{u}\}$
    end
  end
end
for d *from* $n-1$ *to* 0 **do**
  foreach *monomial* $\boldsymbol{m}$ *of degree* d **do** // 2^n
    foreach *monomial* $\boldsymbol{m'}$ *of degree* $d+1$ *such that* $\boldsymbol{m} \preceq \boldsymbol{m'}$ **do** // $\mathcal{O}(n)$
      $T[\boldsymbol{m}] = T[\boldsymbol{m}] \cup T[\boldsymbol{m'}]$
    end
    $T[\boldsymbol{m}] = \min(T[\boldsymbol{m}])$ // $\mathcal{O}(2^{2n})$
  end
end
return T

The core idea of our algorithm is to first compute all the products for all the n-bit functions and to store $\boldsymbol{u}$ in $T[\boldsymbol{m}]$ only if the monomial $\boldsymbol{m}$ *appears* in one of the polynomials $f^{\boldsymbol{u}}$ while, in Algorithm 1, $\boldsymbol{u}$ is stored whenever one monomial *contains* $\boldsymbol{m}$. Only then the table is completed and reduced. A naive approach to do so would be to go through each monomial $\boldsymbol{m'}$, to add $T[\boldsymbol{m'}]$ to $T[\boldsymbol{m}]$ for each monomial $\boldsymbol{m} \preceq \boldsymbol{m'}$ and finally to reduce the table. But we can notice that if three monomials $\boldsymbol{m}$, $\boldsymbol{m'}$ and $\boldsymbol{m''}$ satisfy the relation $\boldsymbol{m} \preceq \boldsymbol{m'} \preceq \boldsymbol{m''}$ we would do:

1. $T[\boldsymbol{m}] \leftarrow T[\boldsymbol{m}] \cup T[\boldsymbol{m'}]$, $T[\boldsymbol{m}] \leftarrow T[\boldsymbol{m}] \cup T[\boldsymbol{m''}]$, $T[\boldsymbol{m'}] \leftarrow T[\boldsymbol{m'}] \cup T[\boldsymbol{m''}]$
2. $T[\boldsymbol{m}] \leftarrow \min(T[\boldsymbol{m}])$, $T[\boldsymbol{m'}] \leftarrow \min(T[\boldsymbol{m'}])$, $T[\boldsymbol{m''}] \leftarrow \min(T[\boldsymbol{m''}])$

Organizing the computation according to the degree of monomials we can remove one step, and perform operations on smaller sets:

1. $T[\boldsymbol{m''}] \leftarrow \min(T[\boldsymbol{m''}])$
2. $T[\boldsymbol{m'}] \leftarrow T[\boldsymbol{m'}] \cup T[\boldsymbol{m''}]$
3. $T[\boldsymbol{m'}] \leftarrow \min(T[\boldsymbol{m'}])$
4. $T[\boldsymbol{m}] \leftarrow T[\boldsymbol{m}] \cup T[\boldsymbol{m'}]$
5. $T[\boldsymbol{m}] \leftarrow \min(T[\boldsymbol{m}])$

We successfully ran this algorithm to generate the propagation tables associated to Super-Sboxes of many block ciphers, including for instance `SKINNY-64` [BJK+16], `Midori-64` [BBI+15] and `PRESENT` [BKL+07].

Remark 1. Note that the complexity $n2^{3n}$ is an upper bound which is never reached in practice. Furthermore all operations are very simple, sequential (cache-friendly) and easy to vectorize and parallelize. Hence the algorithm is practical for 16-bit Super-Sboxes and for instance it requires less than an hour on a 128-core server to build the table for `Midori-64`.

Remark 2. We emphasize our construction does not lead to weak-key distinguishers. Indeed, the table contains a transition if and only if it is valid for at least one key and, for division property, there exists a distinguisher if and only if there is no valid trail.

4.3 Linear Mapping at the Output

As explained in Section 3.2, in [LDF20] Derbez *et al.* suggested to compose the last round with an invertible matrix to extend the distinguishers we can search for. But since we are looking for integral distinguishers, we are only interested in knowing whether the i-th bit of the output is balanced or not. Hence there is no reason to consider invertible matrices, linear combinations are enough, reducing the number of mappings to try from $\mathcal{O}(2^{n^2})$ to $\mathcal{O}(2^n)$.

To illustrate this, let consider the following example:

$$\begin{cases} b_0 &= x \oplus y \oplus xy \oplus xz \\ b_1 &= x \oplus z \oplus xz \oplus yz \\ b_2 &= y \oplus xy \oplus yz \end{cases}.$$

Assuming this is the ANF of the last round, the bits 0, 1 and 2 are balanced if and only if there is no trail reaching at least one monomial of $\{x, y, z, xy, xz\}$, $\{x, y, z, xz, yz\}$ and $\{x, y, z, xy, yz\}$ respectively.

Let now look at all the linear combinations of b_0, b_1 and b_2:

$$\begin{cases} b_0 \oplus b_1 &= y \oplus z \oplus xy \oplus yz \\ b_0 \oplus b_2 &= x \oplus xz \oplus yz \\ b_1 \oplus b_2 &= x \oplus y \oplus z \oplus xy \oplus xz \\ b_0 \oplus b_1 \oplus b_2 &= z \end{cases}$$

We can observe the linear combination $b_0 \oplus b_1 \oplus b_2$ is balanced if and only if there is no trail reaching z. Furthermore to check whether there is an integral distinguisher on the cipher it is enough to check whether $b_0 \oplus b_1 \oplus b_2$ is balanced or not. Indeed, if $b_0 \oplus b_1 \oplus b_2$ is not balanced then there is a trail reaching z and so all other linear combinations of b_0, b_1 and b_2 are not balanced since they all depend on z.

4.4 Linear Mapping at the Input

As for linear combinations at the output, it is not required to try all invertible matrices to cover the whole search space. Actually, what matters for integral distinguishers is the vector space spawned by constant (linear combinations of) bits. Indeed, let $P(x_1, \dots x_n)$ be a polynomial and let $\mathcal{H}(i,j)$ be the property that a polynomial does not contain any monomial greater than or equal to $x_i \dots x_j$. We know there exists two polynomials P_1 and Q_1 such that $P(x_1, \dots, x_n) = x_1 P_1(x_2, \dots, x_n) \oplus Q_1(x_2, \dots, x_n)$. In particular, for any $k \in \{1, \dots, n\}$, P satisfies $\mathcal{H}(1,k)$ if and only if P_1 satisfies $\mathcal{H}(2,k)$. Now let be $j \in \{2, \dots, n\}$ and consider polynomial $P'(x_1, \dots, x_n) = P(x_1 \oplus x_j, x_2, \dots, x_n)$. We have the following equalities:

$$\begin{aligned} P'(x_1, \dots, x_n) &= P(x_1 \oplus x_j, x_2, \dots, x_n) \\ &= (x_1 \oplus x_j) P_1(x_2, \dots, x_n) \oplus Q_1(x_2, \dots, x_n) \\ &= x_1 P_1(x_2, \dots, x_n) \oplus (x_j P_1(x_2, \dots, x_n) \oplus Q_1(x_2, \dots, x_n)) \\ &= x_1 P_1(x_2, \dots, x_n) \oplus Q'_1(x_2, \dots, x_n) \end{aligned}$$

As a consequence, P' satisfies $\mathcal{H}(1,k)$ if and only if P_1 satisfies $\mathcal{H}(2,k)$ and thus P' satisfies $\mathcal{H}(1,k)$ if and only if P satisfies $\mathcal{H}(1,k)$. Hence, any invertible matrix that does not modify the vector space of constant bits does not modify the integral distinguisher. In particular, when looking only for the existence of an integral distinguisher *i.e.* without optimizing on the data complexity, it is enough to exhaust the only linear combinations of bits that will be constant, reducing the number of mappings to test from $\mathcal{O}(2^{n^2})$ to $\mathcal{O}(2^n)$.

5 Search Algorithm

In [TM16], Todo and Morii proposed a way to look for integral distinguishers based on the division property, with a complexity upper bounded by 2^n, where n is the block size of the block cipher. In practice, they said that their algorithm is not suitable for block ciphers with block size beyond 32 bits, and thus the number of possible targets is very limited. However, a lot of work has been done towards efficiently searching such distinguishers, based on either MILP or SAT/SMT solvers.

Regarding MILP-based search algorithms, the main point is to generate sets of inequalities describing all the propagation tables involved in the decomposition of the cipher. But the number of inequalities required to describe a 16-bit propagation table seems too large to be handled efficiently by any MILP solver. For instance, the propagation table of the Super-Sbox of `Midori-64` contains approximately 2^{23} elements. Hence we developed a dedicated algorithm to search for integral distinguishers. To the best of our knowledge, this is the first time one shows a practical algorithm to search for division trails on 64-bit block ciphers not relying on generic solvers for MILP, SAT or SMT models.

5.1 Dedicated Tool to Search for Integral Distinguishers

We aimed at developing a tool to search for integral distinguishers and able to handle large propagation tables to increase the precision compared to previous approaches. To simplify the implementation process, and to be as fast as possible, we restrict ourselves in answering the following question: is there an integral distinguisher? Indeed, we believe this is the most important question and most often the only one interesting designers. Hence we did not try to improve on the data complexity nor the number of balanced bits. Actually, finding the integral distinguisher with the smallest possible data is a very hard task and seems completely out of reach if we consider linear mappings at the input.

Decomposition of the cipher. Our tool takes as input a block cipher, given as a sequence of $R \times 4$ 16-bit functions $\{f_0^0, f_1^0, f_2^0, f_3^0, f_0^1, \dots, f_3^{R-1}\}$ and a nibble permutation

Algorithm 3: Overview of our tool

Data: Sequence of $R \times 4$ 16-bit functions $\{f_0^0, f_1^0, f_2^0, f_3^0, f_0^1, \ldots, f_3^{R-1}\}$ and a nibble permutation σ of $\{0, 1, \ldots, 15\}$.
Result: List of integral distinguishers, if the list is empty there is no integral distinguisher against the cipher

```
L ⟵ ∅
for c_i from 0 to 3 do
    for c_o from 0 to 3 do
        V_i ⟵ MINIMALINPUTS(f^0_{c_i})
        V_o ⟵ MINIMALOUTPUTS(f^{R-1}_{c_o})
        for v_i ∈ V_i and v_o ∈ V_o do
            flag ⟵ false
            for i ∈ v_i and o ∈ v_o do
                if isThereATrail(i, o, {f^1_0, f^1_1, ..., f^{R-2}_3}, σ) then flag ⟵ true
            end
            if not flag then L ⟵ (v_i, v_o)
        end
    end
end
return L
```

σ of $\{0, 1, \ldots, 15\}$. We denote by x_i^j the i-th nibble at the input of round j and by y_i^j the i-th nibble at output of round j. The relation between those variables is as follows:

$$\begin{array}{cccc}
\boxed{\begin{array}{cc} x_0^j & \\ x_1^j & \\ x_2^j & \\ x_3^j & \end{array} \xrightarrow{f_0^j} \begin{array}{c} y_0^j \\ y_1^j \\ y_2^j \\ y_3^j \end{array}} & & y_{\sigma(0)}^j & x_0^{j+1} \\
 & & y_{\sigma(1)}^j & x_1^{j+1} \\
 & \xrightarrow{\sigma} & y_{\sigma(2)}^j & x_2^{j+1} \\
 & & y_{\sigma(3)}^j & x_3^{j+1} \\
\vdots & & \vdots & \vdots \\
\boxed{\begin{array}{cc} x_{12}^j & \\ x_{13}^j & \\ x_{14}^j & \\ x_{15}^j & \end{array} \xrightarrow{f_3^j} \begin{array}{c} y_{12}^j \\ y_{13}^j \\ y_{14}^j \\ y_{15}^j \end{array}} & & y_{\sigma(12)}^j & x_{12}^{j+1} \\
 & & y_{\sigma(13)}^j & x_{13}^{j+1} \\
 & & y_{\sigma(14)}^j & x_{14}^{j+1} \\
 & & y_{\sigma(15)}^j & x_{15}^{j+1}
\end{array}$$

The functions may depend on some key bits so Super-Sboxes are handled. However we do not take care of the key-schedule and key bits involved in different functions are considered as independent. This representation is generic enough to handle most of SPN-based block ciphers with 64-bit internal state. Note that here

$$(x_{4i}^j, x_{4i+1}^j, x_{4i+2}^j, x_{4i+3}^j) \xrightarrow{f_i^j} (y_{4i}^j, y_{4i+1}^j, y_{4i+2}^j, y_{4i+3}^j)$$

means that the transition from $(x_{4i}^j, x_{4i+1}^j, x_{4i+2}^j, x_{4i+3}^j)$ to $(y_{4i}^j, y_{4i+1}^j, y_{4i+2}^j, y_{4i+3}^j)$ should be valid regarding the propagation table of f_i^j.

First layer. This corresponds to function MINIMALINPUTS in Algorithm 3 and aims at finding which linear mapping to add in front of the cipher. Here *first layer* means the first

application of the four Super-Sboxes (one for each column). We only want to check for the existence of an integral distinguisher so our input division property has an Hamming weight of 63 (*i.e.* the input of one Super-Sbox has Hamming weight 15 while the 3 other ones have Hamming weight 16). For each column, we try the $2^{16}-1$ possible linear combinations for the constant bit and keep only the *minimal* ones. Here, minimal means the smallest set of inputs such that if there is no integral distinguisher from the inputs of the set then there is no integral property on the cipher.

Note that this step of the algorithm is equivalent to generating $2^{16}-1$ invertible matrices L_{in} such that the first line takes all the possible non-zero values, then computing the propagation table of $f_i^0 \circ L_{in}$ and extracting the line $01\ldots1$ of the table. However, once we compute all the products of the components of f_i^0, getting all the sets we want is straightforward. Let us consider the following example:

$$\begin{cases} f^{000} = 1 & f^{011} = y \oplus xy \oplus xz \oplus yz \\ f^{001} = x \oplus y \oplus yz & f^{101} = xz \\ f^{010} = y \oplus xz & f^{110} = xy \oplus xz \\ f^{100} = z \oplus xy \oplus yz & f^{111} = xz \oplus xyz \end{cases}$$

The only information we need to remember is the monomials of degree at least $n-1$ (in this example $n=3$) and then we only work with the simplified following version:

$$\begin{cases} f^{001} = yz & f^{011} = xy \oplus xz \oplus yz \\ f^{010} = xz & f^{101} = xz \\ f^{100} = xy \oplus yz & f^{110} = xy \oplus xz \end{cases} \qquad f^{111} = xz \oplus xyz$$

Now, combining f with a linear mapping $L_{in} = (\alpha_{i,j})$ modifies monomials as follows:

$$\begin{aligned} xy &\longrightarrow \alpha_{0,0}xy \oplus \alpha_{0,1}xz \oplus \alpha_{0,2}yz \oplus p_0(x,y,z) \\ xz &\longrightarrow \alpha_{1,0}xy \oplus \alpha_{1,1}xz \oplus \alpha_{1,2}yz \oplus p_1(x,y,z) \\ yz &\longrightarrow \alpha_{2,0}xy \oplus \alpha_{2,1}xz \oplus \alpha_{2,2}yz \oplus p_2(x,y,z) \\ xyz &\longrightarrow xyz \oplus q(x,y,z) \end{aligned}$$

where the p_i's have degree at most 1 and q at most 2. But we are only interested in the $\boldsymbol{u}$'s such that $f^{\boldsymbol{u}}$ contains a monomial containing yz. Hence we can restrict ourselves to:

$$\begin{cases} f^{001} = \alpha_{2,2}yz & f^{011} = (\alpha_{0,2} \oplus \alpha_{1,2} \oplus \alpha_{2,2})yz \\ f^{010} = \alpha_{1,2}yz & f^{101} = \alpha_{1,2}yz \\ f^{100} = (\alpha_{0,2} \oplus \alpha_{2,2})yz & f^{110} = (\alpha_{0,2} \oplus \alpha_{1,2})yz \end{cases} \qquad f^{111} = xyz$$

Now, we can try the 2^3-1 possible linear combinations and we find that the only sets we have to try are $\{001, 110\}$, $\{010, 101\}$ and $\{100, 011\}$ corresponding respectively to $x \oplus z$, y and z being constant. Indeed, all other sets have smaller elements and so will not lead to a distinguisher if those ones do not.

Last layer. This corresponds to function MINIMALOUTPUTS in Algorithm 3. As for the first layer, the last one is handled separately. For each column, we try the $2^{16}-1$ possible linear combinations for the balanced bit and keep only the *minimal* ones *i.e.* the smallest set of outputs required to check whether there is an integral distinguisher against the cipher or not. This was illustrated in Section 4.3.

Middle layers. We begin by constructing the propagation tables for each of the f_i^j's. Then we reduce the tables using the improvement described Section 4.1. Then for each pair of input/output constructed in the previous steps, we exhaust all trails with a classical

branch-and-bound approach. At each step we look at all the unset variables, guess the one with the less possible values and propagate this information which may reduce the number of possible values for other variables. If we find a trail then there is no integral distinguisher for the pair. Otherwise the pair is saved and will be returned by the algorithm. An example illustrating the behavior of Algorithm 4 is given Section A.

Algorithm 4: isThereATrail

Data: a 64-bit input i, a 64-bit output o, a sequence of $R \times 4$ 16-bit functions $\{f_0^0, f_1^0, f_2^0, f_3^0, f_0^1, \ldots, f_3^{R-1}\}$, a nibble permutation σ of $\{0, 1, \ldots, 15\}$, the set of variables X

Result: boolean indicating whether there is a trail from i to o

Keep for f^0 only transitions from i

Keep for f^{R-1} only transitions to o

if a variable from X has no possible value **then return** false

$x \longleftarrow$ selectNextVariable(X, $\{f_0^0, \ldots, f_3^{R-1}\}$)

for *all possible value* k *of* x **do**

  $X' \longleftarrow X$

  set variable x in X' to value k

  $s \longleftarrow \{f_0^0, \ldots, f_3^{R-1}\}$

  Remove from propagation tables in s transitions which do not go through k in variable x

  if isThereATrail(i, o, s, σ, X') **then return** true

  Remove from propagation tables of $\{f_0^0, \ldots, f_3^{R-1}\}$ transitions which do go through k in nibble x

end

return *false*

Complexity. It is quite hard to evaluate the time complexity as it depends on too many parameters. First it is important to notice that we have less transitions than in the classical approach. For instance if we would like to know whether a transition $x \rightarrow y$ is valid through a Super-Sbox with our approach we only have to look in the propagation table of the Super-Sbox. But in the classical approach we would have to find u and v such that both $x \rightarrow u$ and $v \rightarrow y$ are valid transitions through the **SubCells** layer and such that $u \rightarrow v$ is a valid transition through the linear layer. Furthermore there may be many couples (u, v) satisfying those conditions.

It also seems that sometimes searching for integral distinguishers is much easier than we would expect. Our tool was designed to focus on trails with inputs of Hamming weight $n - 1$ and outputs of Hamming weight 1 which highly restricts the possible values of intermediate variables.

In practice, the running time of our tool was reasonable for all the ciphers we tried except for one: LED [GPPR11]. We believe this due to the complex linear layer of LED which leads to a very high number of possible transitions.

Remarks. We tried several heuristics for the selection of the next variable to guess and selecting the one with the less possible values gave the best results. Note that in our algorithm a variable is a nibble. We tried to guess the internal state variables bit by bit instead of nibble by nibble but that was much worst for most of our targets. We believe this is because for SPN with a linear layer operating at the word level, bit-based division property is not that far to word-based division property.

The most expensive operation in our algorithm is the propagation which aims at restricting the number of possible values for all variables. But overall, the main difficulty is

the high number of trails to try. Hence, we believe the improvement presented Section 4.1 is of great importance to improve the efficiency of any approach searching for integral distinguishers.

Note also that Algorithm 3 could be easily extended to bigger internal states but limitation comes from Algorithm 2 which cannot be used to build propagation table for functions on more than 16 bits.

6 Results

In this section we give the results of our new algorithm combined with the improvements presented in Section 4. We also used our tool to search for low data distinguishers by setting at the input each Super-Sbox to an Hamming weight of 0, 15 or 16. While this does not cover all the possible inputs, we obtained some very interesting results.

6.1 Midori-64

`Midori` is a lightweight block cipher designed by Banik *et al.* and presented at ASIACRYPT'15. It has a classical SPN structure but the **MixColumns** matrix is a binary non-MDS matrix. An overview of the cipher is depicted on Figure 1, and we refer the interested readers to [BBI+15] for the specification of `Midori`.

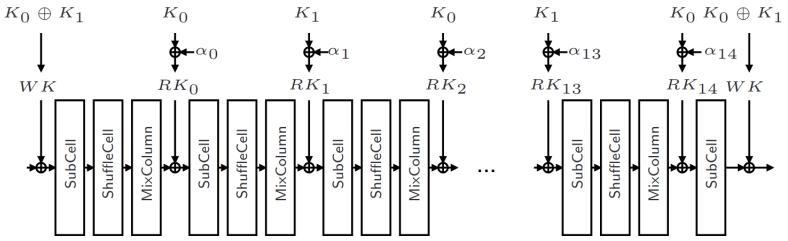


Figure 1: Overview of `Midori-64` [BBI+15]

We decomposed the cipher by alternating Super-Sboxes and linear layer. For the last part of the cipher, if the number of rounds targeted is odd then we consider the 16-bit sboxes obtained by combining both the linear layer and the **SubCell** operations.

We were able to study up to 10 rounds of `Midori`. As a result, we found new integral distinguishers against 9-round `Midori-64` and showed there is no such distinguisher against 10 rounds. Note that the previously known best integral distinguisher against `Midori-64` reached only 7 rounds and the technique used was described in [ZR19] by Zhang and Rijmen. Furthermore, they claimed that *there is no potential of improvement of the result on the attack by distinguishers after using our method* which our results invalidate. The fact is that Zhang and Rijmen compute exactly the Sbox propagation tables and the linear layer, but the composition on two rounds is not exact and it is precisely what we exploit.

Integral distinguishers against 6-round `Midori-64`. We found several distinguishers on 6 rounds requiring only 2^{15} data and the search procedure took 569 CPU-minutes (32 minutes on our 2× AMD EPYC 7742 64-Core server). For instance, if bits of indices from $\mathbf{ShuffleCell}^{-1}(\{9, 16, \dots 63\})$ are constant while we sum on the other ones, then the xor of bits 1, 5, 9 and 13 of the state right after the 6-th application of the **SubCell** operation is balanced.

Integral distinguishers against 7-round `Midori-64`. We used our tool to search for integral distinguishers on 7 rounds. We were able to find distinguishers requiring

only 2^{45} chosen plaintexts. For instance, if at the input the linear combination of bits $b_1 \oplus b_5 \oplus b_9 \oplus b_{13}$ is constant for three first Super-Sboxes and if the whole fourth one is also constant while summing on a complementary space of dimension 45 then the xor of bits 1, 5, 9 and 13 of the state right after the 7-th application of the **SubCell** operation is balanced.

However our tool shown its limits on this example. Exhausting distinguishers (inside our restricted search space) with data 2^{48} and 2^{47} took only few minutes on our server. But we manually stopped the search for distinguishers requiring 2^{46} after two weeks. For distinguishers with data 2^{45} we had to manually force an input/output pair we believed would lead to a distinguisher and it took a full day to our tool to confirm it. This shows how hard it is to search for low data integral distinguishers, at least with our method.

Integral distinguishers against 9-round `Midori-64`. Our search algorithm took $5,183$ CPU-minutes to exhaust minimal distinguishers (89 minutes on our server). As a result, we found several distinsguishers requiring 2^{63} data. At the input they require that, for one Super-Sbox, one of the following linear combinations of bits is constant:

- $b_1 \oplus b_5 \oplus b_9 \oplus b_{13}$
- $b_0 \oplus b_1 \oplus b_2 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_8 \oplus b_9 \oplus b_{10} \oplus b_{12} \oplus b_{13} \oplus b_{14}$

In that case, and if summing on a complementary space of dimension 63, for all columns on the state right after the 9-th application of the **SubCell** operation the following linear combinations of bits are balanced:

- $b_1 \oplus b_5 \oplus b_9 \oplus b_{13}$
- $b_0 \oplus b_2 \oplus b_4 \oplus b_6 \oplus b_8 \oplus b_{10} \oplus b_{12} \oplus b_{14}$
- $b_0 \oplus b_1 \oplus b_2 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_8 \oplus b_9 \oplus b_{10} \oplus b_{12} \oplus b_{13} \oplus b_{14}$

Integral distinguishers against 10-round `Midori-64`. Our search algorithm exhausted the search space in 509 CPU-minutes (11 minutes on our server) and did not find any distinguisher.[1]

Propagation tables. We compared the propagation tables of both the Super-Sbox and the classical approach. We found that as soon as the Hamming weight of the input is at least 9 then all lines of the table present a difference:

0	:	0%	4	:	19%	8	:	99.8%	12	:	100%
1	:	0%	5	:	38.2%	9	:	100%	13	:	100%
2	:	0%	6	:	85%	10	:	100%	14	:	100%
3	:	2.9%	7	:	97.8%	11	:	100%	15	:	100%

This table indicates the percentage of inputs of Hamming weight hw for which the classical approach leads to at least one transition not in the propagation table of the Super-Sbox, *i.e.* a false transition. In particular, it shows that for low Hamming weight inputs there is no benefit in considering the Super-Sbox (in term of precision). Furthermore, over the 7,815,842 elements in the propagation table obtained with the classical approach, only 3,860,001 actually belong to the propagation table of the Super-Sbox. Hence, with the classical approach, approximately half of the transitions are not valid, showing how our technique increases the precision of the division property.

Remark. Since we did not consider the key-schedule nor the round constants (seen as belonging to the key-schedule), we were able to reduce the search space by a factor 4, using the inner symmetries of the cipher.

[1] Note that it does not mean there are no integral distinguishers against 10 rounds but only that we can not conclude on their existence.

6.2 SKINNY-64

SKINNY [BJK+16] is a family of very lightweight tweakable block ciphers designed by Beierle *et al.* and presented at CRYPTO'16. The round function of SKINNY is very similar to the one of Midori. It also has a classical SPN structure and a non-MDS binary matrix is used for the **MixColumns** operation. But the matrix has a low weight (only half of the coefficient are non-zero) and the tweakey is xored to the two first rows only. The round function of SKINNY is depicted on Figure 2 and we refer the interested reader to [BJK+16] for more details.

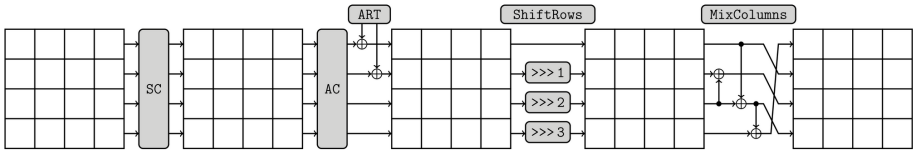


Figure 2: Round function of SKINNY [BJK+16]

We focused on SKINNY-64, the internal state of the 128-bit version being too big to be handled by our tool. As for Midori, we decomposed the cipher by alternating Super-Sboxes and linear layers. However, we had to generate two different propagation tables for the Super-Sboxes, one for the first column, and one for the other columns. Indeed, the third nibble of the first column has to be xored with the round constant but not to the key. Hopefully, the same constant (0x02) is used for all rounds and we do not need to build a different propagation table for each round nor taking care of the rounds we are analysing.

Many integral distinguishers against 10-round SKINNY-64 were published in the last years (e.g. [BJK+16], [ZR19] or [EKKT18]). While the designers of SKINNY wrote in the original paper that maybe the division property could be used to slightly extend those results, Zhang and Rijmen claimed in [ZR19] there is *no space for improvement of the result on this type of distinguishers*. Actually, all the previous approaches led to the conclusion that there is no integral distinguisher against 11 or more rounds of SKINNY-64. But with our new technique, we were able to find distinguishers against 11 rounds, showing the benefit in precision obtained by considering Super-Sboxes.

Integral distinguishers against 8-round SKINNY-64. We found several distinguishers on 8 rounds requiring only 2^{15} data. For instance, if bit 1 as well as all bits of indices in $\mathbf{ShiftRows}^{-1}(\{16,\dots,63\})$ are constant while we sum on the other ones, then both bit 7 and bit 11 of the state right after the 8-th application of the **SubCells** operation are balanced.

The search procedure took 125 CPU-minutes to finish (5 minutes on our server).

Integral distinguishers against 10-round SKINNY-64. We found several distinguishers on 10 rounds requiring only 2^{47} data. For instance, if we take as constant bits of indices in $\mathbf{ShiftRows}^{-1}(\{13,32,\dots,47\})$ while we sum on the other ones, then bit 39 of the state right after the 10-th application of the **SubCells** operation is balanced.

The search procedure took 160 CPU-minutes to finish (7 minutes on our server).

Integral distinguishers against 11-round SKINNY-64. The search algorithm took 683 CPU-minutes (22 minutes on our server) to exhaust all the minimal distinguishers. As a result we found if bit 1 is constant while summing on all the 63 other bits, then bits b_6, b_7, $b_7 \oplus b_{15}$, b_{22}, b_{23}, $b_{18} \oplus b_{30}$, $b_{23} \oplus b_{31}$, b_{38}, b_{29}, $b_{34} \oplus b_{46}$, $b_{39} \oplus b_{47}$, b_{54} and b_{55} of the state right after the 11-th application of the **SubCells** operation are balanced.

Note that from those results we can deduce more balanced bits as for instance b_{15}. Our algorithm does not output it because it only tries a *minimal* set of (linear combinations of) bits as explained Section 5.

Integral distinguishers against 12-round SKINNY-64. Our search algorithm exhausted the search space in 152 CPU-minutes (4 minutes on our server) and did not find any distinguisher[2].

Propagation tables. As for Midori, we compared the propagation tables of both the Super-Sbox and the classical approach. Results are also surprising since even for inputs of hamming weight 2 there are differences:

0	:	0%	4	:	49.1%	8	:	92.5%	12	:	98.6%
1	:	0%	5	:	62.9%	9	:	95.8%	13	:	98.4%
2	:	13.3%	6	:	76.2%	10	:	97.8%	14	:	98.3%
3	:	28.9%	7	:	86.5%	11	:	98.8%	15	:	93.7%

More precisely, over the 7,632,808 elements in the propagation table constructed by the classical approach, 3,043,045 do not belong to the propagation table of the Super-Sbox and thus correspond to false transitions. It shows again how our technique is much more accurate than the classical approach.

6.3 HIGHT

HIGHT [HSH+06] is block cipher designed by Hong *et al.* and presented at CHES'06. It is a generalized Feistel Network with 8 branches of 8 bits each. The resistance of HIGHT comes from applying XOR and addition mod 2^8 alternatively. The round function of HIGHT is depicted on Figure 3 and we refer interested readers to [HSH+06] for the complete specification.

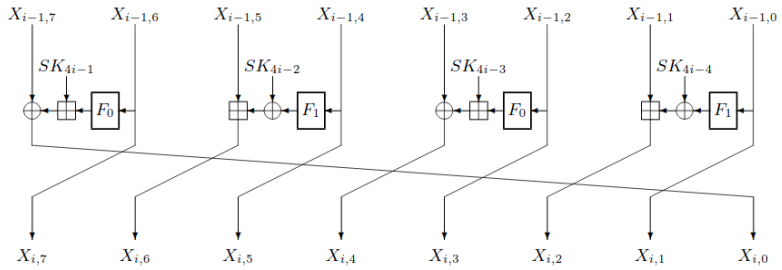


Figure 3: Round function of HIGHT

We computed the propagation tables of both the functions:

- $(x, y) \rightarrow (x, y + (F_1(x) \oplus k) \mod 256)$
- $(x, y) \rightarrow (x, y \oplus (F_0(x) + k \mod 256))$.

One round of HIGHT is then composed of a parallel application of those functions followed by a permutation. This shows that Algorithm 2 as well as our tool are very versatile and not restricted to classical SPN and Super-Sboxes.

We ran our search algorithm up to 21 rounds of HIGHT. We found integral distinguishers up to 20 rounds while the previously best known integral distinguisher against HIGHT could cover only 19 rounds with a data complexity of 2^{63} [FTIM19].

[2] At first we made a mistake in our description of SKINNY in our tool and applied the tweakey to bits of even indices (instead of the two first lines). Interestingly in that case there are integral distinguishers on 12 rounds.

Integral distinguishers against 20-round HIGHT. The search algorithm took more than 13 days on our server to exhaust all the minimal distinguishers. As a result, it found 4 distinguishers:

constant (input)	balanced (output)
$b_2 \oplus b_4 \oplus b_5 \oplus b_8$	$b_{17} \oplus b_{22} \oplus b_{23} \oplus b_{24}$
$b_{34} \oplus b_{36} \oplus b_{37} \oplus b_{40}$	$b_{49} \oplus b_{54} \oplus b_{55} \oplus b_{56}$
$b_{17} \oplus b_{22} \oplus b_{23} \oplus b_{24}$	$b_{50} \oplus b_{52} \oplus b_{53} \oplus b_{56}$
$b_{49} \oplus b_{54} \oplus b_{55} \oplus b_{56}$	$b_{18} \oplus b_{20} \oplus b_{21} \oplus b_{24}$

Remark. Our distinguishers against 20-round HIGHT require linear mappings both in front and back of the cipher and we did not find other distinguishers. This shows the property *all maximal degree monomials* used in [HLLT20] does not ensure security against integral distinguishers as claimed by Hebborn *et al.*. Indeed, while they do consider the case of a linear mapping added at the end of the cipher, they do not consider the possibility of adding it at the beginning.

6.4 Low Data Distinguishers

We experimentally verified the low data distinguishers and the experiments never failed.[3] For each experiment all the round keys as well as the constant bits were drawn uniformly at random, showing our distinguishers are independent of the key schedule as we expected.

We do not give results for both 8-round Midori-64 nor 9-round SKINNY-64. This is because the best integral distinguishers we found have approximately the same data complexity than our distinguishers on 9 and 10 rounds respectively. This comes from the restriction to the search space we added to our tool: at the input of the cipher, for each Super-Sbox the input has Hamming weight 0, 15 or 16.

7 Conclusion

In this paper we showed how considering larger Sboxes and, especially, Super-Sboxes, makes the propagation more accurate. We discovered new integral distinguishers that previous approaches could not find, covering more rounds against 3 well-studied block ciphers. We also proposed several generic improvements regarding division property, including an algorithm to reduce the number of trails to try as well as much faster algorithms to add linear mappings around the cipher, reducing the number of mappings to try from $\mathcal{O}(2^{n^2})$ to $\mathcal{O}(2^n)$.

We also believe this work will challenge the community in handling such large propagation tables with generic solvers for MILP, SAT or SMT models. Furthermore our search algorithm is quite basic and we are sure there is room for improvement.

Future work. In this work we built the propagation tables for Super-Boxes by adding transitions valid for at least one key. However it could be interesting to investigate the weak-key setting, for instance by adding a transition only if it is valid for 50% of the keys.

References

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors,

[3] The distinguishers requiring 2^{45} and 2^{47} data were run only twice each.

Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[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 *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153, 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, pages 450–466, 2007.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher square. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.

[EKKT18] Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding integral distinguishers with ease. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 115–138. Springer, 2018.

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of rijndael. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2000.

[FTIM19] Yuki Funabiki, Yosuke Todo, Takanori Isobe, and Masakatu Morii. Improved integral attack on HIGHT. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 102-A(9):1259–1271, 2019.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[HLLT20] Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. Lower bounds on the degree of block ciphers. *IACR Cryptol. ePrint Arch.*, 2020:1051, 2020.

[HLM+20] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 466–495. Springer, 2020.

[HSH+06] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A new block cipher suitable for low-resource device. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006.

[LDF20] Baptiste Lambin, Patrick Derbez, and Pierre-Alain Fouque. Linearly equivalent s-boxes and the division property. *Des. Codes Cryptogr.*, 88(10):2207–2231, 2020.

[TM16] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon Family. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 357–377, 2016.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 287–314, 2015.

[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 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016.

[ZBL+15] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *SCIENCE CHINA Information Sciences*, 58(12):1–15, 2015.

[ZR19] Wenying Zhang and Vincent Rijmen. Division cryptanalysis of block ciphers with a binary diffusion layer. *IET Information Security*, 13(2):87–95, 2019.

A About the Search Procedure

In this section we detail our search procedure on an example extracted from our tool on SKINNY-64. The procedure is a classical branch-and-bound, guessing variables and propagating constraints to other ones. Let assume after s steps of the algorithm one branch arrives to the internal state described in Table 2. This table contains the possible values for each (nibble) variable involved in a division trail. Variables from x_i and y_i are related by the propagation tables of the Super-Sboxes while variables from y_i and x_{i+1} are related by the propagation tables of the linear layers. More precisely, for all i the following transitions should hold:

- $(x_i[\sigma(4j)], \ldots, x_i[\sigma(4j+3)]) \xrightarrow{SSB} (y_i[4j], \ldots, y_i[4j+3])$, $0 \leq j < 4$
- $(y_i[\sigma(4j)], \ldots, y_i[\sigma(4j+3)]) \xrightarrow{L} (x_{i+1}[4j], \ldots, x_{i+1}[4j+3])$, $0 \leq j < 4$
- $\sigma : (0\ 1\ \ldots\ 15) \rightarrow (0\ 13\ 10\ 7\ 4\ 1\ 14\ 11\ 8\ 5\ 2\ 15\ 12\ 9\ 6\ 3)$

Table 2: Internal state at Step s

	y_1	x_2	y_2	x_3	y_3	x_4	y_4	x_5
0	4	F	F	7	F	$1, 4, 6, C$	0	0
1	F	4	F	B	$0, 2, 4$	B	0	0
2	F	F	F	D	$1, 2, 4, 6, A, C$	7	0	6
3	6	F	F	1	$0, 2, 8$	$6, A, C, E$	8	0
4	F	F	F	F	$2, 3, 5, 6, 9, D, F$	$3, 5, 7, E$	C	0
5	F	F	2	F	$1, 2, 6, C$	$2, 3, 5, 6, 9, B$	0	C
6	F	F	3	F	$1, 2, 6, 9, C$	$3, A, C, E$	4	0
7	F	F	1	F	$1, 2, 4, 6, C$	0	0	0
8	F	F	F	F	$1, 2, 4, 6, 9, C$	0	0	0
9	F	F	F	$3, 6, A, E$	$1, 3, 6, C$	0	0	0
10	F	F	F	$7, B, F$	F	$1, 2, 6, C$	6	0
11	F	F	F	$3, 7, B, E$	$2, 4, 6$	$1, 2, 3, 4, 5, 6, 7, 9, A, C, E$	0	0
12	F	6	F	F	$1, 3, 6, C$	$1, 3, 6, C, E$	2	E
13	F	F	1	7	2	0	0	0
14	F	F	F	F	F	$1, 3, 5, 6, 7, C, E$	0	0
15	F	F	F	B	0	0	0	0

At this step, our algorithm first searches for the unset variable with this least possible values. If several variables reach this minimum our tool would select in priority the furthest from the *middle* (y_3 in this example). Here $x_3[10]$ is selected. It can assume only 3 different values: 7, B and F. At this point the algorithm creates two branches: a first one with $x_3[10] = 7$ and a second one with $x_3[10] \in \{B, F\}$. Variable $x_3[10]$ is involved in two transitions:

- $(x_3[0], x_3[13], x_3[10], x_3[7]) \xrightarrow{SSB} (y_3[0], y_3[1], y_3[2], y_3[3])$
- $(y_2[8], y_2[5], y_2[2], y_2[15]) \xrightarrow{L} (x_3[8], x_3[9], x_3[10], x_3[11])$

For each branch we create a list of transitions to check for and add to it those two ones. Then as long as the list is non-empty, we pick the transition with the least possible values for the input or output, remove it from the list and check whether it restricts the possible values of involved variables. For instance, here $y_2[8, 5, 2, 15]$ is fixed so we begin by the transition $(y_2[8], y_2[5], y_2[2], y_2[15]) \xrightarrow{L} (x_3[8], x_3[9], x_3[10], x_3[11])$. We look into the propagation table T_L and for each value from $T_L[y_2[8], y_2[5], y_2[2], y_2[15]]$ contained in $Vx_3[8] \times Vx_3[9] \times Vx_3[10] \times Vx_3[11]$ (where $Vx_3[i]$ is the set of possible values for $x_3[i]$), we save the possible values for each nibble independently. Finally, we update the sets $Vx_3[i]$ and for each of them which changed we add the two corresponding transitions to the list. If a set $Vx_3[i]$ becomes empty then there is no solution and the branch is cut. Tables 3 and 4 described the two branches, modified sets of values are colored in blue.

Table 3: Internal state at Step $s+1$ - first branch

	y_1	x_2	y_2	x_3	y_3	x_4	y_4	x_5
0	4	F	F	7	F	$1, 6, C$	0	0
1	F	4	F	B	$2, 4$	B	0	0
2	F	F	F	D	$1, 2, C$	7	0	6
3	6	F	F	1	8	A, E	8	0
4	F	F	F	F	$2, 6, 9$	7	C	0
5	F	F	2	F	$1, 2, 6, C$	$2, 6, 9$	0	C
6	F	F	3	F	$1, 2, 6$	E	4	0
7	F	F	1	F	$1, 2, 6, C$	0	0	0
8	F	F	F	F	$1, 2, 6, C$	0	0	0
9	F	F	F	E	$1, 6, C$	0	0	0
10	F	F	F	7	F	$1, 2, 6, C$	6	0
11	F	F	F	B	$2, 4$	$3, 7, 9, A, E$	0	0
12	F	6	F	F	6	E	2	E
13	F	F	1	7	2	0	0	0
14	F	F	F	F	F	$3, 7, E$	0	0
15	F	F	F	B	0	0	0	0

Table 4: Internal state at Step $s+1$ - second branch

	y_1	x_2	y_2	x_3	y_3	x_4	y_4	x_5
0	4	F	F	7	F	$1, 4, 6, C$	0	0
1	F	4	F	B	$0, 2, 4$	B	0	0
2	F	F	F	D	$1, 2, 4, 6, C$	7	0	6
3	6	F	F	1	$0, 2$	$6, A, C, E$	8	0
4	F	F	F	F	$2, 3, 5, 6, 9, D, F$	$3, 5, 7, E$	C	0
5	F	F	2	F	$1, 2, 6, C$	$2, 3, 5, 6, 9, B$	0	C
6	F	F	3	F	$1, 2, 6, 9, C$	$3, A, C, E$	4	0
7	F	F	1	F	$1, 2, 4, 6, C$	0	0	0
8	F	F	F	F	$1, 2, 4, 6, 9, C$	0	0	0
9	F	F	F	$3, 6, A, E$	$1, 3, 6, C$	0	0	0
10	F	F	F	B, F	F	$1, 2, 6, C$	6	0
11	F	F	F	$3, 7, B, E$	$2, 4, 6$	$1, 2, 3, 4, 5, 6, 7, 9, A, C, E$	0	0
12	F	6	F	F	$1, 3, 6, C$	$1, 3, 6, C, E$	2	E
13	F	F	1	7	2	0	0	0
14	F	F	F	F	F	$1, 3, 5, 6, 7, C, E$	0	0
15	F	F	F	B	0	0	0	0