Automatic Security Evaluation for Bit-oriented Block Ciphers in Related-key Model: Application to PRESENT-80, LBlock and Others*

Siwei Sun, Lei Hu, Peng Wang

State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences,
Beijing 100093, China

Abstract. Since AES and PRESENT are two international standard block ciphers representing the most elegant design strategies for byte-oriented and bit-oriented designs respectively, we regard AES and PRESENT the two most significant candidates to scrutinize with respect to related-key differential attack. In EUROCRYPT 2010 and CRYPTO 2013, the security of AES with respect to related-key differential attack has been completely analyzed by Alex Biryukov et al and Pierre-Alain Fouque et al with automatic related-key differential characteristic searching tools. In this paper, we propose two methods to describe the differential behaviour of an S-box with linear inequalities based on logical condition modelling and computational geometry. In one method, inequalities are generated according to some conditional differential properties of the S-box; in the other method, inequalities are extracted from the H-representation of the convex hull of all possible differential patterns of the S-box. For the second method, we develop a greedy algorithm for selecting a given number of inequalities from the convex hull. Using these inequalities combined with Mixed-Integer Linear Programming (MILP) technique, we successfully prove that 24 rounds of PRESENT-80 is enough to resist against standard related-key differential attack, which is the tightest security boud obtained so far for PRESENT-80 with respect to related-key differential attack. This proof is accomplished automatically on a workstation with 8 CPU cores in a time within 14 hours. In a similar way, we also prove that the probability of the best related-key differential characteristic of full LBlock is upper bounded by 2^{-56}, which is the first result concerning the security of full LBlock with respect to related-key differential attack. The methodology presented in this paper is generic, automatic and applicable to lightweight constructions with small block size, small S-boxes, and bit-oriented operations, including but not limited to PRESENT, EPCBC, LBlock, etc, which opens a new interesting direction of research for bit-oriented ciphers and for the application of MILP technique in cryptography.

* All source code for generating valid cutting-off inequalities and MILP instances will be made freely available online after publication of the paper.

Keywords: Related-key differential attack, Active S-box, Mixed-integer Linear Programming, Logical condition modelling, Convex hull

1 Introduction

Contrary to the single-key model, where methodologies for constructing block ciphers provably resistant to differential attack are readily available, the understanding of the security of block ciphers with regard to related-key differential attack is relatively limited. This situation can be seen from the fact that even internationally standardized block ciphers such as AES and PRESENT enjoy no security proof at all for related-key differential attack at the time of their publication. This limited understanding of the security concerning related-key differential attack has been greatly improved in recent years for AES-like byte- or word-oriented SPN block ciphers. Along this line of research, two representative papers [8, 17] were published in Eurocrypt 2010 and Crypto 2013. In the former paper [8], an efficient search tool for finding differential characteristics both in the state and in the key was presented, and the best differential characteristics were obtained for some byte-oriented block ciphers such as AES, byte-Camellia, and Khazad. In the latter paper [17], Pierre-Alain Fouque et al showed that the full-round AES-128 can not be proven secure against related-key differential attack unless the exact coefficients of the MDS matrix and the S-Box differential properties are taken into account. Moreover, a variant of Dijkstra's shortest path algorithm to efficiently find the most efficient related-key attacks on SPN ciphers was developed in [17].

For bit-oriented block ciphers such as PRESENT-80, Sareh Emami proved that no related-key differential characteristic exists with probability higher than 2^{-64} for PRESENT-80, and therefore PRESENT-80 is secure against basic related-key differential attack [16]. In [30], Sun et al obtained tigher security bounds for PRESENT-80 with respect to related-key differential attack using MILP technique.

Due to the fact that the PRESENT block cipher is an international standard for light-weight cryptography and is a representative bit-oriented design, we think it is important to scrutinize its security against related-key differential attack. In this paper, we investigate this problem by Mixed-Integer Linear Programming technique.

Mixed-Integer Linear Programming (MILP). The problem of Mixed Integer Linear Programming (MILP) is a class of optimization problems derived from Linear Programming in which the aim is to optimize an objective function under certain constraints. The field of MILP has received extensive study and achieved great success in both academic and industrial worlds. A Mixed Integer Linear Programming problem can be formally described as follows.

MILP: Find a vector $x \in \mathbb{Z}^k \times \mathbb{R}^{n-k} \subseteq \mathbb{R}^n$ with $Ax \leq b$, such that the linear function $c_1x_1 + c_2x_2 + \cdots + c_nx_n$ is minimized (or maximized), where $(c_1, \ldots, c_n) \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$.

Despite its intimate relationship with discrete optimization problems, such as the set covering problem, 0-1 knapsack problem, and travelling salesman problem, it is only in recent years that MILP has been explicitly applied in cryptographic research.

In [33], Michael Walter et al modelled the problem of finding a set of variables involved in a system of polynomial equations over $\mathbb{F}_2$, such that when assigned to fixed values, the number of known variables in the system can be maximized as an MILP problem. They have applied this idea in guess-and-determine algebraic attack on the EPCBC block cipher [38], and experimental results showed that this strategy resulted in a faster key recovery attack compared to random assignment. In [12], Julia Borghoff employed two methods, standard conversion [4] and adapted standard conversion [22], to convert the problem of solving a system of polynomial equations into an MILP problem. Martin Albrecht et al [1] treated the problem of recovering cryptographic key material from decayed DRAM as a Partial Weighted Max-Polynomial System Solving Problem which can be solved with MILP techniques. Bulygin et al studied the invariant coset attack on PRINTcipher by establishing a one-to-one correspondence between defining sets of the invariant projected subsets of PRINTcipher and all feasible solutions of a specific 0-1 integer programming problem [13]. Moreover, MILP was employed in error-tolerant side channel algebraic attacks [25].

In this paper, we are mainly concerned with the application of MILP method in evaluating the security of block ciphers against related-key differential cryptanalysis. Roughly speaking, differential attack [6] is a cryptanalysis technique used to discover non-random behaviour of a cipher by analyzing the input and output difference of a cipher. A practical approach to evaluate the security of a cipher against differential attack is to determine the lower bound of the number of active S-boxes throughout the cipher. This strategy has been employed in many designs [2, 7, 11, 10, 15]. MILP was applied in automatically determining the lower bounds of the numbers of active S-boxes for some word-oriented symmetric-key ciphers, and therefore used to prove their security against differential cryptanalysis [9, 23, 36] . Sun at el [30] extended this method by making it applicable to ciphers involving bit-oriented operations.

Our Contributions. We find that a main imperfection of [30] is as follows which prevents researchers from obtaining tighter security bounds for round-reduced variants of PRESENT-80.

The constraints presented in [30] is too coarse (and some of these constraints are redundant in some specific case) to accurately describe the differential properties of a specific cipher, since there are a large number of invalid differential patterns of the cipher satisfying all these constraints, which yields a feasible region of the MILP problem much larger than the set of all valid differential characteristics.

In this paper, we propose two methods to tighten the feasible region by cutting off some impossible differential patterns of a specific S-box with linear inequalities: one method is based on logical condition modelling, and the other

is a more general approach based on convex hull computation — a fundamental algorithmic problem in computational geometry. In the first method, typically less than 15 inequalities are generated according to some conditional differential properties of the S-box; while in the second method, several hundreds of inequalities are extracted from the H-representation of the convex hull of all possible differential patterns of the S-box.

However, the second approach produces too many inequalities so that adding all of them to an MILP problem will make the solving process impractical. Therefore, we develop a greedy algorithm for selecting a given number of linear inequalities from the convex hull.

By adding all or a part of the constraints generated by these methods, we automatically prove that the probability of the best related-key differential characteristic of 24-round PRESENT-80, a bit-oriented SPN block cipher, is upper bounded by 2^{-64}. Also, we apply the method to LBlock [37] — a bit-oriented Feistel block cipher, and prove that the probability of the best related-key differential characteristic for full-round LBlock is at most 2^{-56}. This is the first theoretic result concerning full LBlock's security against differential attack in the related-key model. Moreover, the methodology presented in this paper is generic, automatic, and applicable to other lightweight ciphers with bit-oriented operations such as EPCBC [38], LBlock [37], and MIBS [20].

Limitations of the Methodology. The methodology presented in this paper has some limitations which we would like to make clear, and trying to overcome these limitations is a topic deserving further investigation.

1. It's very hard to solve the MILP models generated in this paper, since in our models, we introduce a new variable for almost every input/output bit-level difference, which makes the sizes of the MILP instances reasonably large with respect to the number of variables and constraints. Also, there seems to be no method to estimate the computational complexity before actual computation. This is why we only apply our method to the light-weight block ciphers PRESENT-80 and LBlock whose block sizes and key sizes are 64-bit and 80-bit respectively. For the same reason, the method presented in this paper can only analyze a small number of rounds of a full cipher.

2. This methodoly is only suitable to evaluate the security of constructions with S-boxes, XOR operations and bit permutations, and can not be applied to block ciphers like SIMON and SPECK [3], which involve modulo addition ,and bitwise AND and no S-boxes at all.

Organization of the paper. In Section 2, we introduce Mouha et al's framework and its extension for counting the number of active S-boxes of PRESENT-like ciphers automatically with MILP technique. In Sections 3, 4 and 5 we introduce the concept of valid cutting-off inequalities for tightening the feasible region of an MILP problem, and explore how to generate and select valid cutting-off inequalities. We add these inequalities to the overall constraints of the MILP problems describing the differential behaviour of the block ciphers PRESENT-80 and LBlock in Section 6, which enables us to obtain tighter security bounds

for PRESENT-80 against related-key differential attack. Finally, in Section 7 we conclude the paper and propose some research directions for bit-oriented ciphers and the application of MILP technique in cryptography.

2 Mouha et al's Framework and Its Extension

In this section, we present Mouha et al's framework and its extension for counting the number of differentially active S-boxes for word-oriented and bit-oriented block ciphers respectively.

2.1 Mouha et al's Framework for Word-oriented Block Ciphers

Assume a cipher is composed of the following three word-oriented operations, where m is the word size:

- XOR, $\oplus : \mathbb{F}_2^\omega \times \mathbb{F}_2^\omega \to \mathbb{F}_2^\omega$
- Linear transformation $L : \mathbb{F}_{2^\omega}^m \to \mathbb{F}_{2^\omega}^m$ with branch number

$$\mathcal{B}_L = \min_{a \neq 0} \{\mathrm{WT}(a||L(a)) : a \in \mathbb{F}_{2^\omega}^m\},$$

  where $\mathrm{WT}(a||L(a))$ is the number of non-zero entries of the $2m$-dimensional vector $a||L(a)$ over the finite field $\mathbb{F}_{2^\omega}$
- S-box, $\mathcal{S} : \mathbb{F}_2^\omega \to \mathbb{F}_2^\omega$

Mouha et al's framework uses 0-1 variables, which are subjected to certain constraints imposed by the above operations, to denote the word level differences propagating through the cipher (1 for nonzero difference and 0 for otherwise).

Detailed MILP model building process for determining a lower bound of the number of active S-boxes is described as follows. Firstly, we should include the constraints imposed by the operations of the cipher.

Constraints Imposed by XOR Operations:

Suppose $a \oplus b = c$, where $a,\ b,\ c \in \mathbb{F}_2^\omega$ are the input and output differences of the XOR operation, the following constraints will make sure that when a, b, and c are not all zero, then there are at least two of them are nonzero:

$$\begin{cases} a + b + c \geq 2d_\oplus \\ d_\oplus \geq a \\ d_\oplus \geq b \\ d_\oplus \geq c \end{cases} \tag{1}$$

where $d_\oplus$ is a dummy variable taking values from $\{0, 1\}$.

Constraints Imposed by Linear Transformation:

Let x_{i_k} and $y_{j_k}, k \in \{0, 1, \ldots, m-1\}$, be 0-1 variables denoting the word-level input and output differences of the linear transformation L respectively. Since for nonzero input differences, there are totally at least $\mathcal{B}_L$ nonzero m-bit words in the input and output differences, we include the following constraints:

$$\begin{cases} \sum_{k=0}^{m-1} (x_{i_k} + y_{j_k}) \geq \mathcal{B}_L d_L \\ d_L \geq x_{i_k}, \quad k \in \{0, \ldots, m-1\} \\ d_L \geq y_{j_k}, \quad k \in \{0, \ldots, m-1\} \end{cases} \tag{2}$$

where d_L is a dummy variable taking values in $\{0, 1\}$ and $\mathcal{B}_L$ is the branch number of the linear transformation.

Then, we set up the **objective function** to be the sum of all variables representing the input words of the S-boxes. Obviously, this objective function corresponds to the number of active S-boxes, and can be minimized to determine its lower bound.

Following this approach, the minimum numbers of active S-boxes were obtained in [23] for the r-round ($r \leq 96$) Enocoro-128V2 and full AES ciphers under both single-key and related-key models. We refer the reader to [23] for more information.

2.2 Extension of Mouha's Framework for PRESENT-like Ciphers

For PRESENT-like ciphers, bit-level representations and additional constraints are needed [30]. For every input and output bit-level difference, a new 0-1 variable x_i is introduced obeying the following rule of variable assignment

$$x_i = \begin{cases} 1, \text{ for nonzero difference at this bit,} \\ 0, \text{ otherwise.} \end{cases}$$

For every S-box in the schematic diagram, including the encryption process and the key schedule algorithm, we introduce a new 0-1 variable A_j such that

$$A_i = \begin{cases} 1, \text{ if the input word of the Sbox is nonzero,} \\ 0, \text{ otherwise.} \end{cases}$$

At this point, it is nature to choose the objective function f, which will be minimized, as $\sum A_j$ for the goal of determining a lower bound of the number of active S-boxes.

For PRESENT-like ciphers, we need to include two sets of constraints. The first one is the set of constraints imposed by XOR operations, and the other is due to the S-box operation. After changing the representations to bit-level, the set of constraints imposed by XOR operations for PRESENT-like ciphers are the same as that presented in (1). The S-box operation is more tricky.

Constraints Describing the S-box Operation:

Suppose $(x_{i_0}, \ldots, x_{i_{\omega-1}})$ and $(y_{j_0}, \ldots, y_{j_{\omega-1}})$ are the input and output bit-level differences of an S-box marked by A_t. Firstly, to ensure that $A_t = 1$ holds if and only if $x_{i_0}, \ldots, x_{i_{\omega-1}}$ are not all zero, we require that:

$$\begin{cases} A_t - x_{i_k} \geq 0, & k \in \{0, \ldots, \omega - 1\} \\ x_{i_0} + x_{i_1} + \cdots + x_{i_{\omega-1}} - A_t \geq 0 \end{cases} \tag{3}$$

Also, nonzero input difference must result in nonzero output difference and vice versa:

$$\begin{cases} \omega y_{j_0} + \omega y_{j_1} + \cdots + \omega y_{j_{\omega-1}} - (x_{i_0} + x_{i_1} + \cdots + x_{i_{\omega-1}}) \geq 0 \\ \omega x_{i_0} + \omega x_{i_1} + \cdots + \omega x_{i_{\omega-1}} - (y_{j_0} + y_{j_1} + \cdots + y_{j_{\omega-1}}) \geq 0 \end{cases} \tag{4}$$

Finally, the Hamming weight of the 2ω-bit word $x_{i_0} \cdots x_{i_{\omega-1}} y_{j_0} \cdots y_{j_{\omega-1}}$ is lower bounded by the branch number $\mathcal{B}_\mathcal{S}$ of the S-box for nonzero input difference $x_{i_0} \cdots x_{i_{\omega-1}}$, where $d_\mathcal{S}$ is a dummy variable:

$$\begin{cases} \sum_{k=0}^{\omega-1} (x_{i_k} + x_{j_k}) \geq \mathcal{B}_\mathcal{S} d_\mathcal{S} \\ d_\mathcal{S} \geq x_{i_k}, & k \in \{0, \ldots, \omega - 1\} \\ d_\mathcal{S} \geq y_{j_k}, & k \in \{0, \ldots, \omega - 1\} \end{cases} \tag{5}$$

where the branch number of an S-box $\mathcal{S}$, $\mathcal{B}_\mathcal{S}$, is defined as

$$\mathcal{B}_\mathcal{S} = \min_{a \neq b} \{\mathrm{wt}((a \oplus b) || (\mathcal{S}(a) \oplus \mathcal{S}(b)) : a, b \in \mathbb{F}_2^\omega\}$$

and $\mathrm{wt}(\cdot)$ is the standard Hamming weight of a 2ω-bit word. We point out that constraint (5) is redundant for an invertible S-box with branch number $\mathcal{B}_\mathcal{S} = 2$, since in this particular case, all differential patterns not satisfying (5) violate (4).

0-1 Variables:

The MILP model proposed above is indeed a Pure Integer Programming Problem since all variables appearing are 0-1 variables. However, in practice we only need to explicitly restrict variables representing the differences of plaintexts, master keys and all dummy variables to be 0-1, while all other variables can be allowed to be any real numbers, which leads to a Mixed-integer Linear Programming problem. Following this approach, the MILP solving process may be accelerated as suggested in [12].

3 Tighten the Feasible Region with Valid Cutting-off Inequalities

The feasible region of an MILP problem is defined as the set of all variable assignments satisfying all constraints in the MILP problem. The modelling process presented in the previous sections indicates that every differential path corresponds to a solution in the feasible region of the MILP problem. However, a feasible solution of the MILP model is not guaranteed to be a valid differential

path, since our constraints are far from perfect to rule out all invalid differential patterns. For instance, assume x_i and y_i $(0 \leq i \leq 3)$ are the bit-level input and output differences of the PRESENT-80 S-box (see Table 1). According to Section 2.2, x_i, y_i are subjected to the constraints of (3), (4) and (5). Obviously, $(x_0 \cdots, x_3, y_0, \cdots, y_3) = (1,0,0,1,1,1,0,1)$ satisfies the above constraints, whereas $0x9 = 1001 \rightarrow 0xB = 1101$ (the left most bit is the least significant bit in our representation) is not a valid difference propagation pattern for the PRESENT S-box, which can be seen from the differential distribution table presented in Table 2. Hence, we are actually trying to minimize the number of active S-boxes over a larger region as illustrated in Fig. 1, and the optimum value obtained in this setting must be smaller than or equal to the actual minimum number of active S-boxes. Although the above fact will not invalidate the lower bound we obtained from our MILP model, this prevents the designers from obtaining tighter security bounds and therefore making better security and efficiency trade-offs.

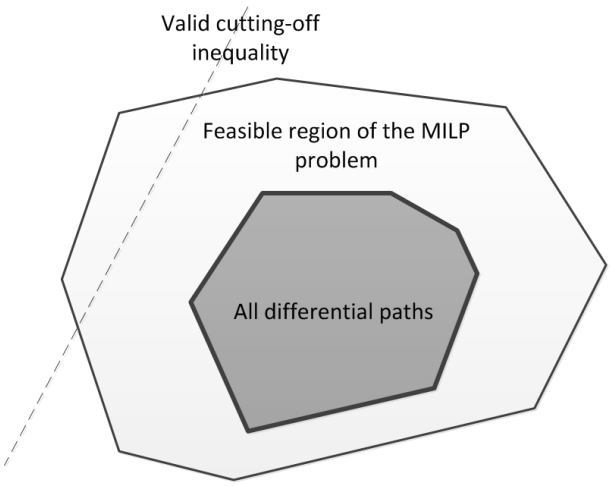

Fig. 1: The relationship between the set of all differential paths and the feasible region of the MILP problem, and the effect of cutting-off inequality

The situation would be even worse when modelling an invertible S-box with branch number $\mathcal{B}_\mathcal{S} = 2$, which is the minimal value of the branch number for an invertible S-box. In the case of invertible S-box with $\mathcal{B}_\mathcal{S} = 2$, the constraints of (3), (4) are enough, and (5) is redundant. In this situation, all differential patterns with nonzero input and output differences satisfy the constraints presented in the previous sections, which is obviously too coarse to describe a specific S-box. For instance, all 10 S-boxes of LBlock [37] are invertible and their branch numbers are all 2.

Therefore, we are motivated to look for linear inequalities which can cut off some part of the feasible region of the MILP model while leaving the region of valid differential paths intact as illustrated in Fig. 1. For the convenience of discussion, we give the following definition.

Definition 1. *A linear inequality satisfied by all possible valid differential patterns is called a valid cutting-off inequality if it is violated by at least one feasible solution corresponding to an impossible differential pattern in the feasible region of the original MILP problem.*

4 Methods for Generating Valid Cutting-off Inequalities

In this section, we present two methods for generating valid cutting-off inequalities by analyzing the differential behaviour of the underlying S-box.

Table 1: The S-box of PRESENT-80

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Table 2: The Differential Distribution Table of the PRESENT S-box

	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2_x	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3_x	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4_x	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5_x	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6_x	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7_x	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8_x	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9_x	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
A_x	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
B_x	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
C_x	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
D_x	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
E_x	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
F_x	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

4.1 Modelling Conditional Differential Behaviour with Linear Inequalities

In building integer programming models in practice, sometimes it is possible to model certain logical constraints as linear inequalities. For example, assume x is a continuous variable such that $0 \leq x \leq M$, and we know that δ is a 0-1 variable taking value 1 when $x > 0$, that is

$$x > 0 \quad \Rightarrow \quad \delta = 1.$$

It is easy to verify that the above logical condition can be achieved by imposing the following constraint

$$x - M\delta \leq 0.$$

In fact, there is a surprisingly large number of different types of logical conditions can be imposed in a similar way, and we refer the reader to [34, 35] for many other examples.

In this subsection, we take the advantage of this technique to describe the conditional differential behaviour of the PRESENT S-box, which is referred to as undisturbed bits in [31].

Theorem 1. *The S-box of PRESENT-80 has the following properties:*

(i) If the input difference of the S-box is $0x9 = 1001$*, then the least significant bit of the output difference must be* 0*;*

(ii) If the input difference of the S-box is $0x1 = 1000$ *or* $0x8 = 0001$*, then the least significant bit of the output difference must be* 1*;*

(iii) If the output difference of the S-box is $0x1 = 1000$ *or* $0x4 = 0010$*, then the least significant bit of the input difference must be* 1*; and*

(iv) If the output difference of the S-box is $0x5 = 1010$*, then the least significant bit of the input difference must be* 0*.*

Note that similar conditional differential behaviours of other ciphers were also used by other cryptanalysts in different context [18, 21, 13].

Theorem 2. *Let 0-1 variables* x_i *and* y_i $(0 \leq i \leq 3)$ *represent the input and output bit-level differences of the S-box respectively. Then the logical conditions in Theorem 1 can be described by the following linear inequalities:*

$$x_0 + x_3 - x_1 - x_2 + y_0 \leq 2 \tag{6}$$

$$\begin{cases} x_0 - x_1 - x_2 - x_3 - y_0 \leq 0 \\ x_3 - x_0 - x_1 - x_2 - y_0 \leq 0 \end{cases} \tag{7}$$

$$\begin{cases} y_0 - y_1 - y_2 - y_3 - x_0 \leq 0 \\ y_2 - y_0 - y_1 - y_3 - x_0 \leq 0 \end{cases} \tag{8}$$

$$y_0 + y_2 - y_1 - y_3 + x_0 \leq 2 \tag{9}$$

For example, the linear inequality (6) removes all differential patterns of the form $(x_0, x_1, x_2, x_3, y_0, y_1, y_2, y_3) = (1, 0, 0, 1, 1, *, *, *)$, where $(x_0, \ldots, x_3)$ and $(y_0, \ldots, y_3)$ are the input and output differences of the PRESENT S-box respectively. We call this group of constraints presented in (6), (7), (8), and (9) the constraints of conditional differential propagation (CDP constraints for short). The CDP constraints obtained from Theorem 1 and the differential patterns removed by these CDP constraints are given in Table 3.

Table 3: Impossible differential patterns removed by the CDP constraints generated according to the differential properties of the PRESENT S-box. Here, a 9-dimensional vector $(\lambda_0, \ldots, \lambda_3, \gamma_0, \ldots, \gamma_3, \theta)$ in the left column denotes a linear inequality $\lambda_0 x_0 + \cdots + \lambda_3 x_3 + \gamma_0 y_0 + \cdots + \gamma_3 y_3 + \theta \geq 0$, and an 8-dimensional vector in the right column denotes a difference propagation pattern, e.g., $(1, 0, 0, 1, 1, 1, 1, 0)$ denotes $0x9 = 1001 \rightarrow 0x7 = 1110$.

Constraints obtained by logical condition modelling	Impossible differential patterns removed
$(-1, 1, 1, -1, -1, 0, 0, 0, 2)$	(1, 0, 0, 1, 1, 1, 1, 0) (1, 0, 0, 1, 1, 0, 0, 1) (1, 0, 0, 1, 1, 0, 0, 0) (1, 0, 0, 1, 1, 1, 0, 0) (1, 0, 0, 1, 1, 1, 0, 1) (1, 0, 0, 1, 1, 0, 1, 0) (1, 0, 0, 1, 1, 0, 1, 1) (1, 0, 0, 1, 1, 1, 1, 1)
$(-1, 1, 1, 1, 1, 0, 0, 0, 0)$	(1, 0, 0, 0, 0, 1, 1, 1) (1, 0, 0, 0, 0, 1, 1, 0) (1, 0, 0, 0, 0, 0, 0, 1) (1, 0, 0, 0, 0, 1, 0, 0) (1, 0, 0, 0, 0, 1, 0, 1) (1, 0, 0, 0, 0, 0, 1, 0) (1, 0, 0, 0, 0, 0, 1, 1)
$(1, 1, 1, -1, 1, 0, 0, 0, 0)$	(0, 0, 0, 1, 0, 1, 1, 1) (0, 0, 0, 1, 0, 1, 1, 0) (0, 0, 0, 1, 0, 0, 0, 1) (0, 0, 0, 1, 0, 1, 0, 0) (0, 0, 0, 1, 0, 1, 0, 1) (0, 0, 0, 1, 0, 0, 1, 0) (0, 0, 0, 1, 0, 0, 1, 1)
$(-1, 0, 0, 0, -1, 1, -1, 1, 2)$	(1, 1, 1, 0, 1, 0, 1, 0) (1, 0, 1, 1, 1, 0, 1, 0) (1, 1, 0, 1, 1, 0, 1, 0) (1, 1, 1, 1, 1, 0, 1, 0) (1, 0, 1, 0, 1, 0, 1, 0) (1, 0, 0, 0, 1, 0, 1, 0) (1, 1, 0, 0, 1, 0, 1, 0)
$(1, 0, 0, 0, -1, 1, 1, 1, 0)$	(0, 0, 0, 1, 1, 0, 0, 0) (0, 1, 0, 0, 1, 0, 0, 0) (0, 1, 0, 1, 1, 0, 0, 0) (0, 0, 1, 0, 1, 0, 0, 0) (0, 0, 1, 1, 1, 0, 0, 0) (0, 1, 1, 1, 1, 0, 0, 0) (0, 1, 1, 0, 1, 0, 0, 0)
$(1, 0, 0, 0, 1, 1, -1, 1, 0)$	(0, 0, 1, 1, 0, 0, 1, 0) (0, 1, 0, 0, 0, 0, 1, 0) (0, 1, 1, 0, 0, 0, 1, 0) (0, 0, 1, 0, 0, 0, 1, 0) (0, 1, 1, 1, 0, 0, 1, 0) (0, 1, 0, 1, 0, 0, 1, 0)

However, there are cases where no such conditional differential property exists. For example, two out of the eight S-boxes of Serpent [5] exhibit no such property. Even when the S-box under consideration can be described with this logical condition modelling technique, the inequalities generated may be not enough to produce a satisfied result. The number of valid cutting-off inequalities can be obtained in this way is given in Table 4 for typical 4×4 S-boxes.

In the next subsection, a more general approach for generating valid cutting-off inequalities is proposed.

4.2 Convex Hull of All Possible Differentials for an S-box

The convex hull of a set Q of discrete points in $\mathbb{R}^n$ is the smallest convex set that contains Q. A convex hull in $\mathbb{R}^n$ can be described as the common solutions

Table 4: Number of valid cutting-off inequalities obtained using different methods. Notations: the "# CDP" columns record the number of constraints obtained using logical condition modelling approach, and the "#CH" columns record the number of constraints in the H-representation of the convex hull.

S-box	#CDP	#CH	S-box	#CDP	#CH
Klein	0	312	LBlock S6	12	205
Piccolo	12	202	LBlock S7	12	205
TWINE	0	324	LBlock S8	12	205
PRINCE	0	300	LBlock S9	12	205
MIBS	0	378	Serpent S0	6	327
PRESENT/LED	6	327	Serpent S1	6	327
LBlock S0	12	205	Serpent S2	6	325
LBlock S1	12	205	Serpent S3	0	368
LBlock S2	12	205	Serpent S4	3	321
LBlock S3	12	205	Serpent S5	3	321
LBlock S4	12	205	Serpent S6	3	327
LBlock S5	12	205	Serpent S7	6	368

of a set of finitely many linear equations and inequalities as follows:

$$\begin{cases} \lambda_{0,0}x_0 + \cdots + \lambda_{0,n-1}x_{n-1} + \lambda_{0,n} \geq 0 \\ \qquad \cdots \\ \gamma_{0,0}x_0 + \cdots + \gamma_{0,n-1}x_{n-1} + \gamma_{0,n} = 0 \\ \qquad \cdots \end{cases} \tag{10}$$

This is called the H-Representation of a convex hull. Computing the H-representation of the convex hull of a set of finitely many points is a fundamental algorithm in computation geometry with many applications [19, 28, 26]. Here, we treat a possible differential of the PRESENT S-box as a point in $\mathbb{R}^8$. For example, the difference propagation pattern $0x9 = 1001 \rightarrow 0x7 = 1110$ is identified with $(1, 0, 0, 1, 1, 1, 1, 0)$.

We now define the convex hull of a specific $\omega \times \omega$ S-box to be the set of all linear inequalities in the H-Representation of the convex hull $\mathcal{V}_{\mathcal{S}} \subseteq \mathbb{R}^{2\omega}$ of all possible differential patterns of the S-box. For instance, the Convex Hull of PRESENT S-box can be found in Appendix B. This result is obtained by using the inequality_generator() function in the sage.geometry.polyhedron class of the SAGE computer algebra system [29]. The convex hull of the PRESENT S-box contains 327 linear inequalities. Any one of these inequalities can be taken as a valid cutting-off inequality. The numbers of linear inequalities in the convex hulls of typical 4×4 S-boxes are given in Table 4.

5 Selecting Valid Cutting-off Inequalities from the Convex Hull: A Greedy Approach

It is well known that the number of equations and inequalities in the H-Representation of a convex hull computed from a set of discrete points in n dimensional space is exponential in n. For instance, the convex hull $\mathcal{V}_{\mathcal{S}} \subseteq \mathbb{R}^8$ of a 4×4 S-box typically involves several hundreds of linear inequalities. Adding all of them to an MILP problem for counting the number of active S-boxes will quickly make the MILP problem insolvable in practical time. Hence, it is necessary to select a small number, say n, of "best" inequalities from the convex hull. Here by "best" we mean that, among all possible selections of n inequalities, the selected ones maximize the number of removed impossible differentials. Obviously, this is a hard combinatorial optimization problem. Therefore, we design a greedy algorithm, listed in Algorithm 1, to approximate the optimum selection.

Algorithm 1: Selecting n inequalities from the convex hull $\mathcal{H}$ of an S-box

Input:
$\mathcal{H}$: the set of all inequalities in the H-representation of the convex hull of an S-box;
$\mathcal{X}$: the set of all possible differential patterns of an S-box;
n: a positive integer.
Output: $\mathcal{O}$: a set of n inequalities selected from $\mathcal{H}$

$l^* :=$ None;
$\mathcal{X}^* := \mathcal{X}$;
$\mathcal{H}^* := \mathcal{H}$;
$\mathcal{O} := \emptyset$;
for $i \in \{0, \ldots, n-1\}$ **do**
  $l^* :=$ The inequality in $\mathcal{H}^*$ which maximizes the number of removed impossible differential patterns from $\mathcal{X}^*$;
  $\mathcal{X}^* := \mathcal{X}^* - \{$removed impossible differential patterns by $l^*\}$;
  $\mathcal{H}^* := \mathcal{H}^* - \{l^*\}$;
  $\mathcal{O} := \mathcal{O} \cup \{l^*\}$;
end
return $\mathcal{O}$

The algorithm builds up a set of valid cutting-off inequalities by selecting at each step an inequality from the convex hull which maximizes the number of removed impossible differential patterns from the current feasible region.

We select 6 valid cutting-off inequalities from the convex hull of the PRESENT S-box using Algorithm 1. These inequalities and the impossible differential patterns removed are listed in Table 5. Compared with the 6 valid cutting-off inequalities obtained in Theorem 1 (see Table 3), they cut off $66 - 42 = 24$ more impossible differential patterns, which leads to a relatively tighter feasible region.

Table 5: Impossible differential patterns removed by the constraints selected from the convex hull of the PRESENT S-box

Constraints selected from the convex hull by the greedy algorithm	Impossible differential patterns removed
$(-2, 1, 1, 3, 1, -1, 1, 2, 0)$	(1, 0, 1, 0, 0, 1, 0, 0) (1, 0, 0, 0, 1, 1, 0, 0) (1, 0, 0, 0, 1, 0, 0, 0) (1, 0, 1, 0, 0, 1, 1, 0) (1, 0, 0, 0, 1, 1, 1, 0) (1, 1, 0, 0, 0, 1, 0, 0) (1, 1, 0, 0, 0, 1, 1, 0) (1, 0, 0, 0, 0, 1, 1, 0) (1, 0, 1, 0, 1, 1, 0, 0) (1, 0, 0, 0, 0, 0, 1, 0, 0) (1, 0, 0, 0, 0, 1, 0, 1) (1, 0, 0, 0, 0, 0, 1, 0) (1, 1, 0, 0, 1, 1, 0, 0) (1, 1, 1, 0, 0, 1, 0, 0)
$(1, -2, -3, -2, 1, -4, 3, -3, 10)$	(0, 1, 1, 0, 1, 1, 0, 1) (1, 1, 1, 0, 0, 1, 0, 1) (0, 1, 1, 1, 0, 1, 1, 1) (1, 0, 1, 1, 0, 1, 0, 1) (0, 1, 1, 0, 0, 1, 0, 1) (0, 1, 1, 1, 0, 1, 0, 0) (0, 1, 1, 1, 0, 1, 0, 1) (1, 1, 1, 1, 1, 1, 0, 1) (0, 0, 1, 1, 0, 1, 0, 1) (0, 1, 1, 1, 1, 1, 0, 1) (1, 1, 1, 1, 0, 1, 0, 1) (0, 1, 0, 1, 0, 1, 0, 1) (0, 0, 1, 1, 1, 1, 0, 1)
$(2, -2, 3, -4, -1, -4, -4, 1, 11)$	(0, 1, 0, 1, 0, 1, 1, 0) (1, 1, 0, 1, 0, 1, 1, 0) (0, 0, 0, 1, 1, 1, 1, 0) (0, 1, 0, 1, 0, 1, 1, 1) (0, 0, 0, 1, 1, 1, 1, 1) (0, 1, 0, 1, 1, 1, 1, 1) (0, 1, 0, 1, 1, 1, 1, 0) (0, 0, 0, 1, 0, 1, 1, 0) (1, 1, 0, 1, 1, 1, 1, 0) (0, 1, 1, 1, 1, 1, 1, 0) (1, 1, 0, 1, 1, 1, 1, 1)
$(-1, -2, -2, -1, -1, 2, -1, 0, 6)$	(1, 1, 1, 0, 1, 0, 1, 1) (1, 1, 1, 0, 1, 0, 1, 0) (1, 1, 1, 1, 1, 0, 0, 1) (1, 1, 1, 1, 1, 0, 0, 0) (0, 1, 1, 1, 1, 0, 1, 1) (1, 1, 1, 1, 1, 0, 1, 0) (0, 1, 1, 1, 1, 0, 1, 0) (1, 1, 1, 1, 0, 0, 1, 1) (1, 1, 1, 1, 1, 0, 1, 1) (1, 1, 1, 0, 0, 1, 0)
(-2, 1, -2, -1, 1, -1, -2, 0, 6)	(1, 1, 1, 1, 0, 1, 1, 0) (1, 1, 1, 1, 0, 1, 1, 1) (1, 0, 1, 1, 0, 0, 1, 0) (1, 0, 1, 0, 0, 1, 1, 1) (1, 0, 1, 1, 0, 0, 1, 1) (1, 0, 1, 1, 1, 1, 1, 0) (1, 0, 1, 1, 1, 1, 1, 1) (1, 0, 1, 1, 0, 1, 1, 1) (1, 0, 1, 1, 0, 1, 1, 0)
$(2, 1, 1, -3, 1, 2, 1, 2, 0)$	(0, 0, 0, 1, 1, 0, 0, 0) (0, 0, 1, 1, 0, 0, 1, 0) (0, 0, 0, 1, 0, 0, 0, 1) (0, 1, 0, 1, 1, 0, 0, 0) (0, 0, 0, 1, 0, 1, 0, 0) (0, 0, 0, 1, 0, 0, 1, 0) (0, 0, 1, 1, 1, 0, 0, 0) (0, 1, 0, 1, 0, 0, 1, 0) (0, 0, 0, 1, 1, 0, 1, 0)

6 Application to PRESENT-80 and LBlock

In this section, we apply our method to two block ciphers with different structures. One is the bit-oriented SPN block cipher PRESENT-80, and the other is the bit-oriented Feistel block cipher LBlock.

6.1 The 28-round Reduced PRESENT-80 is Secure Against Related-key Differential Attack

We have applied the method presented in previous sections to the block cipher PRESENT-80 to determine its security bound with respect to related-key differential attack. A Python module [32] is developed to generate the MILP instances in "lp" format [14]. In each of these MILP models, we include one more constraint to ensure that the difference of the initial key register is nonzero, since the case where the difference of the initial key register is zero can by analyzed in a single-key model. Then we employ the Gurobi 5.5 optimizer [24] to solve the MILP instances.

By default the computations are performed on a PC using 4 threads with Intel(R) Core(TM) Quad CPU (2.83GHz, 3.25GB RAM, Windows XP), and a star "*" is appended on a timing data to mark that the corresponding computation is taken on a workstation equipped with two Intel(R) Xeon(R) E5620 CPU(2.4GHz, 8GB RAM, totally 8 cores). Despite there are only 2 CPUs and totally 8 physical cores on the workstation, we fire up 16 threads in Gurobi5.5 to solve the corresponding MILP instances to exploit Intel's Hyper-Threading Technology, where for each physical core, the operating system simulates two virtual or logical cores, and shares the workload between them.

We have computed the number of active S-boxes for PRESENT-80 in the related-key model up to 14 rounds, and the results are summarized in Table 6 and Table 7, where the "#Constraints" columns record the number of constraints imposed and the "#Variables" columns show the number of 0-1 variables and continuous variables in the underlying MILP instance. For example, according to the first row of Table 6, there are 97 0-1 variables and 277 continuous variables in the MILP instance corresponding to 1-round PRESENT-80, and the Gurobi optimizer find the minimum number of active S-boxes is 0 in no more than 1 second.

Note that there are $(17 \times 6)r$ more constraints of the rth row(corresponding to r rounds) of Table 6 than that of Table 7. This is due to the fact that for every S-box there are 6 more constraints (see Theorem 1) in the MILP instance with CDP constraints included, and for every round there are 17 S-boxes: 16 in the encryption process (Appendix A, Fig. 2) and 1 in the key schedule algorithm (Appendix A, Fig. 3).

These results clearly demonstrate that the MILP models with CDP constraints lead to tighter security bounds. In particular, we have proved that there are at least 16 active S-boxes in the best related-key differential characteristic for any consecutive 12-rounds of PRESENT-80. Therefore, the probability of the best related-key differential characteristic of 24-round PRESENT-80 is

Table 6: MILP models with CDP constraints added

Rounds	#Variables	#Constraints	#Active S-boxes	Time (in seconds)
1	$97 + 277$	632	0	1
2	$130 + 474$	1262	0	1
3	$163 + 671$	1892	1	1
4	$196 + 868$	2522	2	1
5	$229 + 1065$	3152	3	5
6	$262 + 1262$	3782	5	16
7	$295 + 1459$	4412	7	107
8	$328 + 1656$	5042	9	254
9	$361 + 1853$	5672	10	522
10	$394 + 2050$	6302	13	4158
11	$427 + 2247$	6932	15	18124
12	$460 + 2444$	7562	16	50017
13	$493 + 2641$	8192	18	137160*
14	$526 + 2838$	8822	20	1316808*
15	$559 + 3035$	9452	–	$>$ 20days

Table 7: MILP models without CDP constraints

Rounds	#Variables	#Constraints	#Active S-boxes	Time (in seconds)
1	$97 + 277$	530	0	1
2	$130 + 474$	1058	0	1
3	$163 + 671$	1586	1	1
4	$196 + 868$	2114	2	1
5	$229 + 1065$	2642	3	3
6	$262 + 1262$	3170	4	10
7	$295 + 1459$	3698	6	26
8	$328 + 1656$	4226	8	111
9	$361 + 1853$	4754	9	171
10	$394 + 2050$	5282	12	1540
11	$427 + 2247$	5810	13	8136
12	$460 + 2444$	6338	15	18102
13	$493 + 2641$	6866	17	49537*
14	$526 + 2838$	7394	18	685372*
15	$559 + 3035$	7922	–	$>$ 20days

$(2^{-2})^{16} \times (2^{-2})^{16} = 2^{-64}$, leading to the result that the 24-round PRESENT-80 is resistant to basic related-key differential attack.

For round reduced variants of PRESENT-80 with round $r \geq 15$, we are unable to accomplish the computation within 20 days.

It is possible to get tighter security bounds by adding more constraints: experimental result shows that, by adding 6 more valid cutting-off inequalities listed in Table 5 to the MILP problems for each S-box appearing in the schematic representation of PRESENT-80, we are able to prove that the guaranteed number of active S-boxes in related-key model for 7-round PRESENT-80 is at least 8, which is the tightest bound obtained so far (see Table 6 and Table 7 for comparison).

6.2 Results on LBlock

Up to now, there is no concrete result concerning the security of full-round LBlock [37] against differential attack in the related-key model due to a lack of proper tools for bit-oriented designs.

Since the encryption process of LBlock is nibble-oriented, the security of LBlock against single-key differential attack can be evaluated by those word-oriented techniques. However, the "$\lll 29$" operations in the key schedule algorithm of LBlock destroy its overall nibble-oriented structure, and make those word-oriented approaches infeasible in evaluating the security of LBlock against related-key differential attack.

In this subsection, we apply the method proposed in this paper to LBlock, some results concerning its security against related-key differential attack are obtained. The valid cutting-off inequalities used to obtain these results are listed in Appendix C. Note that the type of constraints given in (5) are removed in our MILP models for LBlock according to the explanations presented in previous sections.

From Table 8, we can deduce that the probability of the best differential characteristic for full LBlock (totally $32 = 11+11+10$ rounds) is upper bounded by $(2^{-2})^{10} \times (2^{-2})^{10} \times (2^{-2})^{8} = 2^{-56}$, where 2^{-2} is the best differential probability for a single S-box of LBlock. To the best of our knowledge, this is the first result concerning the security of the full-round LBlock against related-key differential attack.

7 Conclusion and Directions for Future Work

In this paper, we bring new constraints into Mohua et al's framework to describe the differential properties of a specific S-box, and therefore obtain a more accurate mixed integer programming model for the differential behaviour of a block cipher. Following this methodology, we prove that the 24-round PRESENT-80 is secure against basic related-key differential attack. Moreover, our method is automatic, generic, and applicable to bit-oriented ciphers such as PRESENT, LBlock, and EPCBC.

Table 8: Results for related-key differential analysis on LBlock

Rounds	#Variables	#Constraints	#Active S-boxes	Time (in seconds)
1	218+104	660	0	1
2	292+208	1319	0	1
3	366+312	1978	0	1
4	440+416	2637	0	1
5	514+520	3296	1	2
6	588+624	3955	2	12
7	662+728	4614	3	38
8	736+832	5273	5	128
9	810+936	5932	6	386
10	884+1040	6591	8	19932
11	958+1144	7250	10	43793

At this point, several open problems emerged. Firstly, we have observed that the MILP instances derived from such cryptographic problems are very hard to solve compared with general MILP problems with the same scale with respect to the numbers of variables and constraints. Hence, it is interesting to develop specific methods to accelerate the solving process of such problems and therefore increase the number of rounds of the cipher under consideration can be dealt with. Secondly, the method presented in this paper is very general, is it possible to develop a compiler which can convert a standard description, say a description using hardware description language [27], of a cipher into an MILP instance to automate the entire security evaluation cycle with respect to (related-key) differential attack?

References

1. Albrecht, M., Cid, C.: Cold boot key recovery by solving polynomial systems with noise. In: Applied Cryptography and Network Security. pp. 57–72. Springer (2011)
2. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platformsdesign andanalysis. In: Selected Areas in Cryptography. pp. 39–56. Springer (2001)
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013), `http://eprint.iacr.org/2013/404`
4. Beigel, R.: The polynomial method in circuit complexity. In: Structure in Complexity Theory Conference, 1993., Proceedings of the Eighth Annual. pp. 82–95. IEEE (1993)
5. Biham, E., Anderson, R., Knudsen, L.: Serpent: A new block cipher proposal. In: Fast Software Encryption. pp. 222–238. Springer (1998)
6. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. Journal of CRYPTOLOGY 4(1), 3–72 (1991)
7. Bilgin, B., Bogdanov, A., Knežević, M., Mendel, F., Wang, Q.: Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In:

Cryptographic Hardware and Embedded Systems-CHES 2013, pp. 142–158. Springer (2013)
8. Biryukov, A., Nikolić, I.: Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, Camellia, Khazad and others. In: Advances in Cryptology–EUROCRYPT 2010, pp. 322–344. Springer (2010)
9. Bogdanov, A.: On unbalanced feistel networks with contracting mds diffusion. Designs, Codes and Cryptography 59(1-3), 35–58 (2011)
10. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Cryptographic Hardware and Embedded Systems-CHES 2007, pp. 450–466. Springer (2007)
11. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., et al.: Prince–a low-latency block cipher for pervasive computing applications. In: Advances in Cryptology–ASIACRYPT 2012, pp. 208–225. Springer (2012)
12. Borghoff, J., Knudsen, L.R., Stolpe, M.: Bivium as a mixed-integer linear programming problem. In: Cryptography and Coding, pp. 133–152. Springer (2009)
13. Bulygin, S., Walter, M.: Study of the invariant coset attack on printcipher: more weak keys with practical key recovery. Tech. rep., Cryptology eprint Archive, Report 2012/85 (2012)
14. CPLEX, I.I.: Ibm software group. User-Manual CPLEX 12 (2011)
15. Daemen, J., Rijmen, V., Proposal, A.: Rijndael. In: Proceedings from the First Advanced Encryption Standard Candidate Conference, National Institute of Standards and Technology (NIST) (1998)
16. Emami, S., Ling, S., Nikolic, I., Pieprzyk, J., Wang, H.: The resistance of PRESENT-80 against related-key differential attacks. Cryptology ePrint Archive, Report 2013/522 (2013), http://eprint.iacr.org/
17. Fouque, P.A., Jean, J., Peyrin, T.: Structural evaluation of AES and chosen-key distinguisher of 9-round aes-128. In: Canetti, R., Garay, J. (eds.) Advances in Cryptology CRYPTO 2013, Lecture Notes in Computer Science, vol. 8042, pp. 183–203. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40041-4_11
18. Fuhr, T.: Finding second preimages of short messages for hamsi-256. In: Advances in Cryptology-ASIACRYPT 2010, pp. 20–37. Springer (2010)
19. Goodman, J.E., O'Rourke, J.: Handbook of discrete and computational geometry. CRC press (2010)
20. Izadi, M., Sadeghiyan, B., Sadeghian, S.S., Khanooki, H.A.: Mibs: a new lightweight block cipher. In: Cryptology and Network Security, pp. 334–348. Springer (2009)
21. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of nlfsr-based cryptosystems. In: Advances in Cryptology-ASIACRYPT 2010, pp. 130–145. Springer (2010)
22. Lamberger, M., Nad, T., Rijmen, V.: Numerical solvers and cryptanalysis. Journal of mathematical cryptology 3(3), 249–263 (2009)
23. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Information Security and Cryptology. pp. 57–76. Springer (2012)
24. Optimization, G.: Gurobi optimizer reference manual. URL: http://www. gurobi. com (2013)

25. Oren, Y., Kirschbaum, M., Popp, T., Wool, A.: Algebraic side-channel analysis in the presence of errors. In: Cryptographic Hardware and Embedded Systems, CHES 2010, pp. 428–442. Springer (2010)
26. o'Rourke, J.: Computational geometry in C. Cambridge university press (1998)
27. Pedroni, V.A.: Circuit design with VHDL. The MIT Press (2004)
28. Preparata, F.P., Shamos, M.I.: Computational geometry: An introduction (monographs in computer science). Monographs in Computer Science (Springer-Verlag, New York, 1985), ISBN 3540961313 (1993)
29. Stein, W., et al.: Sage: Open source mathematical software (2008)
30. Sun, S., Hu, L., Song, L., Xie, Y., Wang, P.: Automatic security evaluation of block ciphers with s-bp structures against related-key differential attacks. Cryptology ePrint Archive, Report 2013/547 (2013), `http://eprint.iacr.org/`
31. Tezcan, C.: Improbable differential attack on present using undisturbed bits. In: International Conference on Applied and Computational Mathematics (2013)
32. Van Rossum, G., et al.: Python programming language. In: USENIX Annual Technical Conference (2007)
33. Walter, M., Bulygin, S., Buchmann, J.: Optimizing guessing strategies for algebraic cryptanalysis with applications to EPCBC. In: The 8th China International Conference on Information Security and Cryptology (Inscrypt 2012). Springer (2012)
34. Williams, H.P.: Logical problems and integer programming. Bulletin of the Institute of Mathematics and its Applications 13, 18–20 (1977)
35. Williams, H.P.: Model building in mathematical programming (1999)
36. Wu, S., Wang, M.: Security evaluation against differential cryptanalysis for block cipher structures. Tech. rep., Cryptology ePrint Archive, Report 2011/551 (2011)
37. Wu, W., Zhang, L.: LBlock: a lightweight block cipher. In: Applied Cryptography and Network Security. pp. 327–344. Springer (2011)
38. Yap, H., Khoo, K., Poschmann, A., Henricksen, M.: EPCBC-a block cipher suitable for electronic product code encryption. In: Cryptology and Network Security, pp. 76–97. Springer (2011)

A The PRESENT-80 Lightweight Block Cipher

PRESENT-80 is a 31-round SPN block cipher with 64-bit block size and 80-bit secret key. The substitution and diffusion layers of PRESENT-80 are constructed with 4×4 S-boxes and bit-wise permutation to make its hardware implementation suitable for extremely constrained devices.

The schematic description of PRESENT-80's encryption process and key schedule algorithm are given in Fig.2 and Fig.3. These two schematic descriptions are enough to understand the contents of the following sections, and for more information on PRESENT, we refer the reader to [10].

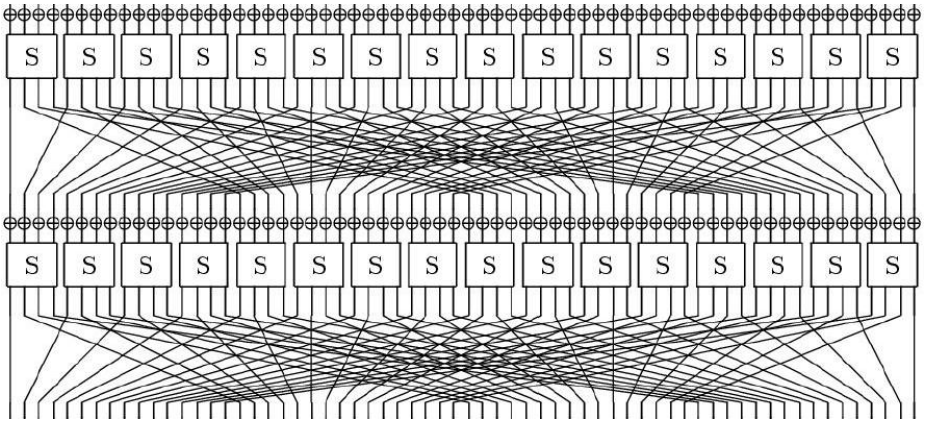

Fig. 2: Two consecutive rounds of PRESENT-80 encryption process

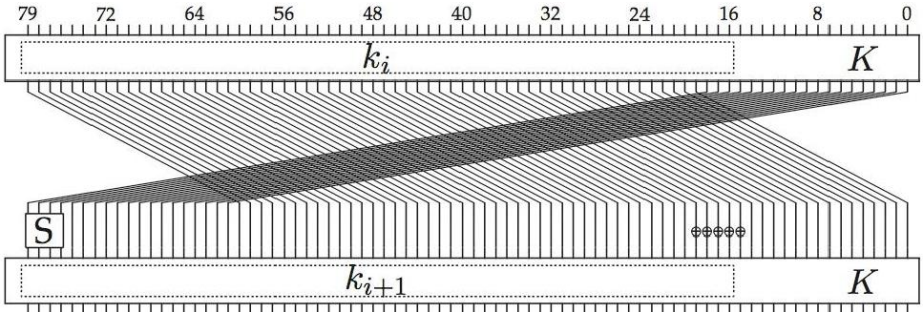

Fig. 3: The key schedule algorithm of PRESENT-80: for each round the most significant 64 bits of the 80-bit key register K are extracted as the subkey k_i

B The Convex Hull of the PRESENT S-box

```
( 0,  2, -2,  1, -1, -1, -2, -2,  6)( 0, -2,  2,  1, -2, -1, -1, -2,  6)( 0,  1, -1,  1,  0, -1, -1, -1,  3)(-1,  0,  1, -1, -1, -1,  0, -1,  4)
( 1,  0,  1,  1,  1,  1, -1,  0,  0)( 0,  1,  1,  1, -1,  0,  0,  1,  0)(-1, -1,  0,  1, -1,  0, -1, -1,  4)( 1,  0, -1,  2,  1,  2,  2, -1,  0)
( 1, -1, -1,  0,  0, -1,  1, -1,  3)(-1,  0, -1, -1,  1,  0, -1,  0,  3)( 0,  1,  1,  0, -1,  1, -1,  1,  1)( 1,  1,  1,  0,  0,  0, -1,  1,  0)
( 2,  1,  2,  2,  0,  1, -1, -1,  0)(-1,  0,  0,  1,  1,  1,  1,  0,  0)(-1,  0,  0,  1,  1,  0,  1,  1,  0)( 0,  1,  1, -1, -1,  1, -1,  0,  2)
( 0,  0, -1,  0,  0,  0,  0,  0,  1)( 1,  2,  0, -1, -2, -2, -2, -1,  6)( 0,  1, -1, -1, -1, -1,  0,  1,  3)( 0,  0, -1,  0,  1,  1,  1,  1,  0)
( 0,  1,  1, -2, -2, -1, -2, -2,  7)( 2,  2, -1,  2, -1,  3,  2, -1,  0)( 1,  0,  1, -1,  0,  1,  1,  1,  0)( 0,  1, -1,  2,  1,  2,  2, -1,  0)
(-2,  1,  1,  3,  1, -1,  1,  2,  0)( 1, -1, -1,  1,  1,  0,  0, -1,  2)( 0,  1,  1, -1,  1,  1,  1,  0,  0)(-2,  1,  1,  2,  1,  0,  1,  1,  0)
( 0, -1, -1, -1, -1,  1,  0, -1,  4)(-1,  1, -2, -1, -2, -2,  1,  2,  6)( 2, -2,  3, -4, -1, -4, -4,  1, 11)( 1, -1,  1,  2,  2,  2,  0, -1,  0)
( 0, -1,  1, -1,  0, -1, -1,  1,  3)( 0,  1, -1,  1,  0,  1,  1, -1,  1)( 1, -1,  2, -1,  0,  2,  2,  2,  0)(-1,  0, -1,  1, -1,  0, -1, -1,  4)
( 0, -1, -1,  0, -1,  1, -1,  1,  3)( 2,  1,  1,  0, -2, -1, -2, -1,  4)( 1,  0, -1,  1,  0,  1,  1, -1,  1)( 1,  2,  2,  1,  0,  0, -2,  1,  0)
( 0,  0,  0,  0,  0, -1,  0,  0,  1)( 1,  1,  1,  1,  1,  1,  1, -2,  0)( 1,  0,  1, -1,  1,  1,  0,  1,  0)( 0, -1,  1, -1, -1, -1, -1,  0,  4)
( 2,  1,  1, -2,  1,  1,  0,  2,  0)(-2,  1, -1, -1,  1, -1, -1, -1,  5)( 1, -1,  0,  2,  2,  2,  1, -1,  0)( 3,  2,  2, -1, -4, -2, -4, -1,  8)
( 0,  0,  0,  1,  1, -1,  1,  1,  0)( 0, -1, -1, -1,  1,  0, -1, -1,  4)( 2,  2,  1,  2, -1,  1,  0, -1,  0)( 1, -1, -1,  0,  1, -1,  0, -1,  3)
( 1, -1, -1,  0, -1,  0,  1, -1,  3)(-2,  0,  0,  1,  2,  1,  2,  1,  0)( 1, -1,  1,  2, -2, -2,  1, -2,  5)( 1, -1,  0,  0,  1,  1, -1, -1,  2)
( 3,  2,  3,  3, -1, -1,  0, -1,  0)( 1,  1, -1,  1, -1, -1,  2,  1, -1)( 1, -1, -1,  1,  2,  0,  2,  1,  0)( 1, -1, -1,  1,  0,  0,  1, -1,  2)
( 1,  1,  1,  1,  1,  1, -2,  1,  0)( 2,  1,  2, -1,  1,  0, -1,  2,  0)( 0, -1, -1, -1, -1,  0,  1, -1,  4)(-2,  1, -2, -1,  1, -1, -2,  0,  6)
( 0, -1,  2, -1, -2, -2, -2, -1,  7)(-1,  0, -1, -1,  1, -1,  0, -1,  4)( 3, -3, -2,  1, -1, -1,  3, -3,  7)( 1,  1, -1,  1,  0,  1,  1,  0,  0)
( 1,  1, -1, -1, -1, -1,  0,  0,  3)( 1, -2, -2,  0,  1, -2,  1, -1,  5)(-1, -1, -1, -1,  3,  4,  3,  4,  0)( 2,  2,  2,  0, -1,  0, -1,  1,  0)
( 2,  0,  2, -1,  2,  1, -1,  2,  0)( 1, -1, -2, -1, -2,  0,  2, -2,  6)( 1, -1,  0, -1,  1,  0, -1, -1,  3)( 1,  0, -1, -1,  0, -1,  1, -1,  3)
( 1,  1,  1,  1,  0,  0, -1,  0,  0)( 1,  0,  1,  1,  1,  1,  0, -1,  0)( 0,  0,  0, -1,  0,  0,  0,  0,  1)( 1, -2, -2, -1,  2, -3,  1, -2,  7)
( 1,  0,  1,  0, -1,  1,  1,  1,  0)(-1,  1,  1, -2, -1, -1, -1, -2,  6)( 1,  2, -1, -2, -2, -1,  1,  1,  4)( 2,  2, -1,  1, -1,  2,  2,  0,  0)
( 1,  1,  0,  1, -1,  1,  0, -1,  1)( 0,  0,  0, -1,  1,  1,  1,  1,  0)( 1,  1, -1,  0,  0,  1,  1,  1,  0)( 1, -2, -2, -1,  1, -1, -2,  1,  4)
( 1,  1, -1,  0, -1,  1,  1,  0,  1)(-3,  2, -2, -1,  1, -2, -2, -1,  8)( 0,  0,  1, -1, -1, -1, -1, -1,  4)( 0,  0,  0,  1,  0,  0,  0,  0,  0)
(-1,  0, -1, -1,  3,  3,  2,  3,  0)( 0,  1, -1, -1, -1, -1, -1,  0,  4)(-1,  0, -1, -1, -1, -1,  1,  1,  3)( 1, -1,  1,  0,  0,  1,  1,  1,  0)
( 1, -1,  1,  0,  0,  1,  1,  1,  0)( 1, -1,  1,  0,  1,  1, -1,  0,  1)(-4,  1,  1,  2,  3,  2,  3,  1,  0)( 2,  1,  1, -3,  1,  2,  1,  2,  0)
( 0,  0, -1,  1,  1,  1,  1,  0,  0)(-1,  0, -1, -1,  1,  1,  0,  1,  2)( 0,  0, -1,  1,  1,  0,  1,  1,  0)( 0,  1, -1,  1, -1,  0, -1, -1,  3)
( 1, -1,  0, -1,  1, -1,  0, -1,  3)(-1,  1,  0, -1,  0, -1, -1, -1,  4)( 0, -1,  0,  0,  0,  0,  0,  0,  1)( 0, -1, -1, -1, -1,  1, -1,  0,  4)
( 1,  1, -1,  2,  1, -2, -2, -2,  5)( 1,  1, -1, -1, -1,  0,  1,  0,  2)(-1,  1,  0, -1,  1, -1,  0, -1,  3)( 0, -1,  0,  0,  1,  1,  1,  1,  0)
( 0,  1,  1, -1, -1,  0, -1, -1,  3)( 0,  1, -1,  0, -1, -1, -1,  0,  4)( 2, -3, -1, -1,  2,  1, -3, -3,  8)( 0,  0,  0,  0,  0,  0,  0, -1,  1)
( 1,  2, -1, -2, -2, -2, -1,  0,  6)(-1,  1,  1,  2,  0, -1,  1,  2,  0)( 1,  1, -2,  1,  1,  1,  1,  1,  0)( 1, -1,  1,  1,  1,  1,  0,  0,  0)
( 0, -1, -1, -1,  1, -2,  1, -2,  5)( 1,  3, -2, -2,  3,  4,  1,  4,  0)(-1, -1, -1, -1,  0,  1,  0,  1,  3)(-2,  0,  2, -2, -1, -1,  1, -2,  6)
( 1, -2,  1,  0,  1,  2,  1,  2,  0)( 1, -1, -2, -2,  1, -3,  2, -2,  7)( 0, -2, -2,  3,  4,  1,  4,  1,  0)( 0,  0,  0, -1,  1, -1,  1, -1,  2)
(-1,  1,  1,  1,  0,  0,  0,  1,  0)(-1,  0,  1,  1,  0, -1,  1,  1,  1)( 0,  2,  2,  1, -1,  1, -1,  2,  0)( 1,  2, -1, -2, -2, -2,  0,  1,  5)
(-1,  1,  1, -1,  1, -1,  1, -2,  3)( 1,  1,  0,  1, -1,  1,  1,  0,  0)( 0,  2,  2,  0, -1, -1,  1,  2,  0)( 2,  1, -2, -2, -1, -1,  1, -1,  5)
( 4,  1,  3, -2,  3,  1, -2,  4,  0)( 1,  0,  0, -1, -1, -1, -1,  1,  3)( 0,  0, -2,  1,  2,  1,  2,  1,  0)( 0,  0,  0,  0,  0, -1,  0,  0,  1)
( 2,  2,  2,  1,  1, -3,  1,  1,  0)( 1,  1,  0, -2,  1,  2,  1,  2,  0)( 0, -1,  1,  1,  1,  1,  0, -1,  1)(-1,  1,  0,  1,  0, -1,  1,  1,  1)
( 1, -1,  1, -1,  1,  0, -1,  0,  2)( 1, -2, -1, -2,  2, -3,  1, -2,  7)( 1,  1,  0, -1, -1,  0,  1,  1,  1)( 1,  1,  1, -2,  1,  1,  1,  1,  0)
(-1, -1,  1,  0,  1, -1, -1,  1,  3)( 1,  1,  1, -1,  1,  0,  1,  0,  0)( 1,  1,  0, -1,  0,  1,  1,  1,  0)(-2, -1, -1,  2, -2,  0, -2, -1,  7)
( 1,  1,  0, -1, -1, -1, -1,  0,  3)( 1, -1, -1,  0,  1,  0,  1,  1,  1)( 0, -1, -1,  1, -1, -1,  1, -1,  5)( 3,  1,  1, -2, -2, -2, -2,  1,  5)
(-2, -1,  2, -1, -2, -2,  0, -1,  7)( 1,  1,  1,  0, -1,  0, -1,  0,  1)( 2,  0,  2,  1, -2, -1, -1, -2,  4)( 1,  0,  1,  0, -1, -1, -1, -1,  3)
(-2,  2,  1,  4,  1, -2,  2,  3,  0)(-1, -1,  1,  0, -1, -1,  0, -1,  4)( 1,  1,  0,  1,  0, -1, -1, -1,  2)( 1, -1, -1,  0,  1, -1,  1,  0,  2)
( 0, -1, -1,  1,  1,  0,  1,  0,  1)(-1, -1, -1, -1, -1,  1,  0,  0,  4)( 1,  0, -1, -1, -1,  0,  1, -1,  3)( 1, -2,  3, -2,  1,  4,  3,  4,  0)
( 1,  1,  0, -1,  1,  1,  0,  1,  0)( 1,  4, -1, -2, -4, -4, -3, -2, 12)( 0,  0,  0,  0,  0,  0,  0,  1,  0)( 0, -1,  1,  1, -1,  0, -1, -1,  3)
( 0,  1,  0,  0,  0,  0,  0,  0,  0)( 3, -1,  3, -1,  3,  2, -1,  2,  0)( 1,  1,  1,  1, -2,  1,  1,  1,  0)( 1,  0,  1,  1,  1, -1, -1,  0, -1,  2)
( 0, -1, -1, -1,  1, -1,  0, -1,  4)( 2, -2,  1, -2,  1, -1, -1, -1,  5)( 1, -1,  3, -1, -2, -3, -3, -2,  9)( 1,  0,  1, -1, -1, -1, -1,  0,  3)
( 0, -1, -1,  1, -1,  0, -1,  1,  3)(-1,  0,  1,  2,  1, -1,  1,  2,  0)( 1,  0,  1, -1,  0, -1, -1,  1,  2)( 1,  0,  2, -1, -2, -2, -2, -1,  6)
( 1,  0,  1, -2,  1,  2,  1,  2,  0)( 0, -1,  0,  1,  1,  1,  1,  0,  0)( 1, -1, -1,  0,  1,  0, -1, -1,  3)( 0, -1,  0,  1,  1,  0,  1,  1,  0)
(-1,  1, -1, -1,  0, -1, -1,  0,  4)(-1,  0,  1, -1,  0, -1,  1, -1,  3)( 2,  1,  1, -1, -2, -1, -2,  0,  4)( 2, -1,  2,  1,  2,  2, -1,  0,  0)
( 1,  2,  2,  1,  0, -2,  0,  1,  0)( 1,  0,  0,  0,  0,  0,  0,  0,  0)( 1,  0,  1,  1,  0,  1, -1, -1,  1)( 2, -1, -3, -1, -3,  1,  2, -3,  8)
( 1, -1,  2, -2,  0, -2, -2,  1,  5)(-1, -1,  0, -1,  2,  3,  3,  3,  0)( 0,  1,  1,  1,  1,  0, -1,  0,  1)( 0, -1,  1,  0, -1,  0,  1,  0,  0)
( 1,  0,  1, -1,  1,  0, -1,  1,  1)(-2,  1,  2,  4,  1, -2,  2,  3,  0)( 1, -1,  4, -2, -3, -4, -4, -2, 12)( 1,  1,  0,  0,  1,  1, -1,  1,  0)
(-1, -1, -1,  0, -1,  1, -1,  0,  4)(-1, -3,  2,  1, -3, -2, -1, -3, 10)( 3,  3,  2,  3,  0, -1, -1, -1,  0)( 1,  1,  0,  0, -1,  1,  1,  1,  0)
( 0,  0,  0,  0,  0,  1,  0,  0,  0)( 0, -1, -1,  2,  2,  1,  2,  0,  0)( 4,  3,  1, -2, -2,  1,  3,  4,  0)( 0,  2, -1, -1, -2, -2, -2, -1,  7)
( 0,  1,  1,  2,  1, -2,  1,  1,  0)( 1,  1,  1,  1,  0, -1,  0,  0,  0)( 0,  0, -1, -1,  2,  2,  0,  2,  1)( 0, -1, -1,  0, -1, -1,  0,  1,  3)
( 1,  1, -1,  2,  0,  2,  2, -1,  0)( 1,  3, -1, -1, -3, -3, -2, -2,  9)( 1, -1, -1,  2,  1,  1,  1, -1,  1)( 0, -1, -1, -1,  0, -1,  1, -1,  4)
( 2,  2,  1, -1, -1,  0,  1,  2,  0)( 1, -2,  2, -3, -1, -3, -3,  1,  9)( 3,  4,  4,  1, -2,  0, -2,  1,  0)( 0,  1,  1,  0,  1,  0,  1, -1,  0)
( 2,  1,  1,  0, -2,  1,  1,  2,  0)( 0, -2,  0,  1,  2,  1,  2,  1,  0)( 1, -2, -1, -1,  2, -2,  0, -2,  6)(-2,  2,  0, -2,  1, -1, -1, -2,  6)
( 0, -1, -1, -1,  0,  1, -1, -1,  4)(-2, -1, -1,  1, -2,  1, -2, -1,  7)(-1,  2, -3,  1, -1, -2, -3, -3, 10)( 1,  2, -1, -1,  2,  2,  0,  2,  0)
(-3, -2,  2, -1, -2, -2,  1, -1,  8)( 0, -1,  1,  0, -1, -1, -1, -1,  4)( 1, -1, -1,  3,  2,  2,  2, -1,  0)( 1, -1,  0,  1,  1,  1,  0, -1,  1)
( 2,  3, -2, -4, -4, -4, -1,  1, 11)( 2,  0,  1, -2, -1, -2, -2,  1,  5)(-1,  1, -1,  0,  1, -1, -1,  0,  3)( 2,  3,  3,  2,  1, -4,  1,  1,  0)
(-1,  1,  0, -1,  1,  0, -1, -1,  3)(-1,  1,  1,  3,  1, -2,  1,  2,  0)( 0,  0,  0,  0,  0,  0, -1,  0,  1)(-2,  2, -1, -1,  0, -2, -2, -1,  7)
( 1,  1,  1,  0,  0, -1,  0,  1,  0)( 1,  1,  0,  0, -1, -1, -1, -1,  3)( 1,  2, -2, -3, -3, -3, -1,  1,  9)( 1, -2,  1,  1,  1,  1,  1,  1,  0)
(-3,  1,  1,  1,  2,  2,  2,  1,  0)( 1,  2,  2,  0, -1,  1, -1,  1,  0)( 1, -2, -3, -2,  1, -4,  3, -3, 10)(-1,  0,  1, -1, -1,  0,  1, -1,  3)
( 0, -1,  1,  2,  2,  2,  1, -1,  0)( 0,  1,  1,  1,  0,  0, -1,  1,  0)( 3,  2,  2,  2, -1,  0, -1, -1,  0)(-1, -1, -1, -1,  0,  1, -1,  0,  4)
( 2,  2,  2, -1,  3, -1,  3, -1,  0)(-1,  1, -1,  0,  0, -1, -1, -1,  4)( 2,  1,  1,  1, -1,  0, -1, -1,  1)( 2,  2,  0,  1, -1, -1, -2, -2,  4)
( 0,  0,  0,  1,  1,  1,  1, -1,  0)(-1,  0,  0,  0,  0,  0,  0,  0,  1)( 1, -1,  1,  1,  1,  2, -1, -1,  1)( 0,  0,  0,  0,  0,  0,  1,  0,  0)
(-1, -2, -2, -1, -1,  2, -1,  0,  6)( 1,  0, -1,  0, -1,  1,  1, -1,  2)( 0, -1,  1, -1,  0,  1,  1,  1,  1)( 0, -1,  1, -1,  1,  2,  2,  2,  0)
( 4,  3,  3, -1, -1, -1, -1,  3,  0)(-1, -1, -1,  1, -1,  0, -1,  0,  4)( 1,  1, -1,  0,  1,  1,  0,  1,  0)( 1,  1,  1,  1,  0,  0,  0, -1,  0)
(-1, -1,  0, -1,  0,  1,  1,  1,  2)( 1,  1,  0, -1, -1, -1,  0,  1,  2)(-2,  1,  2,  4,  2, -2,  1,  3,  0)(-2, -2,  1, -1, -2, -1,  1,  0,  6)
( 0, -1,  1,  1, -1, -1,  0, -1,  3)( 1,  1,  1, -1,  0,  0,  0,  1,  0)( 1,  2,  2,  1, -2,  0,  0,  1,  0)(-1, -1,  1,  0, -1, -1,  1,  0,  3)
( 3,  3,  2, -1, -1, -1,  0,  3,  0)(-1,  0,  0,  0,  1,  1,  1,  1,  0)( 3,  3, -1, -1, -1,  2,  3,  2,  0)( 2, -1,  2,  0,  2,  2,  1,  1,  0)
(-1, -1,  0, -1,  0, -1,  1, -1,  4)(-2, -1,  1, -1, -1, -1,  1, -1,  5)( 1,  1, -2,  0,  1,  2,  1,  2,  0)( 1, -1, -1,  0, -1,  0, -1,  1,  3)
( 0,  1,  0, -1, -1, -1, -1, -1,  4)( 2,  1,  1,  0,  1,  1, -2,  2,  0)(-2,  2,  1,  4,  2, -2,  1,  3,  0)( 1, -2, -2, -1,  1, -2,  2,  1,  7)
( 1,  0,  1,  0,  1,  1, -1,  1,  0)( 1, -2, -1, -1,  2,  0, -2, -2,  6)(-1,  0,  1,  1,  1, -1,  0,  1,  1)( 2,  1,  0, -2, -2, -2, -1,  1,  5)
( 1, -1, -2, -1,  0, -2,  2, -2,  6)(-1,  1,  1, -1,  0,  0,  0, -1,  2)( 1,  1,  1,  1, -1,  0,  0,  0,  0)( 1, -1, -1, -1,  1,  1, -2,  1, -1,  4)
( 2,  1,  1, -2,  0,  1,  1,  2,  0)(-1, -1,  1, -1, -1, -1,  0,  0,  4)( 3,  2,  3, -1,  0, -1, -1,  3,  0)( 1, -1,  1,  0,  1,  1,  0,  1,  0)
( 1, -2, -2,  2,  1,  0,  1, -1,  3)( 1, -1,  0,  1, -1, -1,  1, -1,  3)(-1,  3,  3, -1,  2,  2,  2, -1,  0)(-1,  1,  0,  1,  1, -1,  0,  1,  1)
(-1, -2,  1, -1,  1, -2, -2,  2,  6)( 0,  1, -1,  1, -1,  1,  0, -1,  2)( 0,  1, -1, -1,  2,  2,  1,  2,  0)( 0, -1, -1,  0, -1, -1,  1, -1,  4)
( 0,  0,  0,  0,  1,  0,  0,  0,  0)( 1, -3, -2, -2,  3, -4,  1, -3, 10)( 2,  2, -1,  0, -1,  2,  2,  1,  0)( 1, -1,  2, -2, -1, -2, -2,  0,  6)
( 2, -1,  2,  2,  2,  3, -1, -1,  0)( 0, -2, -2, -2, -1,  2, -1, -1,  7)( 3, -2, -3,  1,  3, -1, -1, -3,  7)(-1,  1,  0,  2,  1, -1,  1,  2,  0)
( 0,  1, -1, -1,  1,  1,  0,  1,  1)( 1,  1,  0,  1,  0,  1,  1, -1,  0)( 1, -1,  1, -1,  0, -1, -1,  0,  3)( 1, -1, -1,  0,  2,  1,  2,  2,  0)
( 0,  0,  1,  0,  0,  0,  0,  0,  0)( 1,  0, -1,  1,  1, -1, -1, -1,  3)(-1,  1,  1,  2,  1, -1,  0,  2,  0)
```

C Valid Cutting-off Inequalities Used in Analyzing LBlock

S-box	Valid cutting-off inequalities
S0	(-1, 2, -2, -1, 0, 0, -2, -1, 5), (0, 1, 0, 0, 1, -1, 1, 0, 0), (-1, -1, 1, -3, 3, -1, -2, 2, 5), (3, -1, -1, -1, 0, 3, 2, 1, 0), (-1, 1, 2, 0, -1, -1, 2, -2, 3), (0, -1, 0, 1, -1, 0, -1, 1, 2), (0, -1, 0, 0, 1, 1, 1, 0, 0), (-1, -1, -1, 0, -1, -1, 0, -1, 5), (1, 2, -2, 1, 0, 0, 1, 2, 0), (1, 2, 3, -2, 1, 0, -1, 3, 0), (-1, 0, 0, 0, 1, 0, 1, 1, 0), (1, 1, -2, -2, 0, -1, -1, -2, 6), (-1, -1, 1, 0, -1, 1, -1, -1, 4), (1, 0, 1, 1, 0, -1, -1, -1, 2), (1, -1, 1, -1, 2, 2, 0, 1, 0), (-1, 2, 2, 1, 0, 0, 2, -1, 0), (0, -1, -1, 1, 1, 1, 0, -1, 2), (-1, 1, 0, 0, -1, 1, 1, -1, 2), (2, 3, 1, 1, 0, -3, 1, 1, 0), (1, -1, -1, 0, 1, -1, -1, 1, 3), (2, -1, -1, 0, -1, 1, 1, 1, 1), (3, 1, 2, 1, -3, -1, 1, 3, 0), (2, -1, -1, 1, -2, 1, 0, 1, 2), (1, -1, 1, -1, 0, 1, 0, -1, 2), (1, 1, 2, 2, 0, 1, 1, -2, 0), (-1, -1, -1, -2, 2, 1, 0, 1, 3)
S1	(2, 1, 1, 1, -3, 0, 2, 1, 0), (-1, 2, -2, -1, 0, 0, -1, -2, 5), (0, 1, 0, 0, 1, -1, 0, 1, 0), (-1, -1, 1, -3, 3, -1, 2, -2, 5), (3, -1, -1, -1, 0, 3, 1, 2, 0), (-1, 4, 5, 3, -1, -1, -2, 5, 0), (0, -1, 0, 1, -1, 0, 1, -1, 2), (0, -1, 0, 0, 1, 1, 0, 1, 0), (1, 2, -2, 1, 0, 0, 2, 1, 0), (-1, -1, -1, 0, -1, -1, -1, 0, 5), (0, 1, 2, -2, 1, 0, 2, -1, 1), (-1, 0, 0, 0, 1, 0, 1, 1, 0), (1, 1, -2, -2, 0, -1, -2, -1, 6), (2, 0, 1, 1, -2, -1, 1, -1, 2), (-1, -1, 1, 0, -1, 1, -1, -1, 4), (-1, 1, -1, 0, -1, 1, -1, 0, 3), (1, -1, 1, -1, 0, 1, -1, 0, 2), (-1, -1, -1, 0, 1, 1, -1, 0, 3), (0, -1, 1, 1, 1, -1, -1, -1, 3), (1, -1, -1, 1, 0, 1, -1, 0, 2), (2, 3, 1, 1, 0, -3, 1, 1, 0), (2, -1, 2, 3, -1, 2, 3, -1, 0), (-1, 1, 1, -1, 2, 0, 1, 1, 0), (-1, -1, 0, 0, -1, -1, -1, 1, 4), (1, -1, 0, 0, 0, 1, 0, 1, 0), (3, -1, -1, 0, -1, 2, 2, 2, 0), (1, 0, -1, 1, 0, -1, 1, -1, 2)
S2	(2, 1, 1, 1, 1, -3, 2, 0, 0), (-1, 2, -2, -1, -2, 0, -1, 0, 5), (0, 1, 0, 0, 1, 1, 0, -1, 0), (-1, -1, 1, -3, -2, 3, 2, -1, 5), (3, -1, -1, -1, 2, 0, 1, 3, 0), (-1, 4, 5, 3, 5, -1, -2, -1, 0), (0, -1, 0, 1, -1, -1, 1, 0, 2), (0, -1, 0, 0, 1, 1, 0, 1, 0), (-1, -1, -1, 0, 0, -1, -1, -1, 5), (1, 2, -2, 1, 1, 0, 2, 0, 0), (1, -2, 1, -2, 1, 3, 2, 4, 0), (1, 1, -2, -2, -1, 0, -2, -1, 6), (-1, 0, 0, 0, 1, 1, 1, 0, 0), (2, 0, 1, 1, -1, -2, 1, -1, 2), (0, -1, 1, 1, -1, 1, -1, -1, 3), (0, 1, 1, -1, -1, 0, 1, 0, 1), (-1, -1, -1, 0, 0, 1, -1, 1, 3), (0, -1, 1, -1, -1, -1, -1, 1, 4), (1, -1, -1, 1, 0, 0, -1, 1, 2), (-1, 1, 0, 0, 1, -1, -1, 1, 2), (3, 2, -1, 3, -1, 0, 3, -1, 0), (1, 2, 1, 1, 1, 0, 0, -2, 0), (-1, 2, 1, -2, 1, 3, 2, 0, 0), (-1, 1, 2, 0, 2, -1, -2, -1, 3), (3, 1, 2, 2, 1, -4, 2, 1, 0), (-1, -1, 1, 1, -1, -1, 0, 1, 3), (3, -1, -1, 0, 2, -1, 2, 2, 0)
S3	(2, 1, 1, 1, 0, 1, -3, 2, 0), (-1, 2, -2, -1, 0, -2, 0, -1, 5), (0, 1, 0, 0, -1, 1, 1, 0, 0), (-1, -1, 1, -3, -1, -2, 3, 2, 5), (3, -1, -1, -1, 3, 2, 0, 1, 0), (-1, 1, 2, 0, -1, 2, -1, -2, 3), (0, -1, 0, 1, 0, -1, -1, 1, 2), (0, -1, 0, 0, 1, 1, 1, 0, 0), (-1, -1, -1, 0, -1, 0, -1, -1, 5), (1, 2, -2, 1, 0, 1, 0, 2, 0), (1, 2, 2, -1, 0, -1, 0, 2, 0), (1, 1, -2, -2, -1, -1, 0, -2, 6), (-1, 0, 0, 0, 0, 1, 1, 1, 0), (1, 0, 1, 1, -1, -1, 0, -1, 2), (1, -1, -1, 1, 1, 0, 0, -1, 2), (1, -1, 1, -1, 2, 0, 2, 1, 0), (-1, 0, 1, 1, 1, 0, 0, 0), (-1, 0, -1, 0, 1, -1, 1, -1, 3), (-1, 1, -1, 0, 1, 0, -1, -1, 3), (0, -1, 1, -1, 1, -1, -1, -1, 4), (3, 2, -1, 3, -1, -1, 0, 3, 0), (1, 2, -1, 1, -1, 1, 0, 1, 0), (3, 1, 2, 0, -1, 1, -2, 2, 0), (-1, -1, 1, 1, 1, -1, -1, 0, 3), (1, 1, 2, 2, 1, 1, 0, -2, 0), (2, -1, -1, 0, 1, 1, -1, 1, 1), (0, -1, 1, 1, -1, -1, 1, -1, 3)
S4	(2, 1, 1, 1, 1, -3, 0, 2, 0), (-1, 2, -2, -1, -2, 0, 0, -1, 5), (0, 1, 0, 0, 1, 1, -1, 0, 0), (-1, -1, 1, -3, -2, 3, -1, 2, 5), (3, -1, -1, -1, 2, 0, 3, 1, 0), (-1, 4, 5, 3, 5, -1, -1, -2, 0), (0, -1, 0, 1, -1, -1, 0, 1, 2), (0, -1, 0, 0, 1, 1, 1, 0, 0), (-1, -1, -1, 0, 0, -1, -1, -1, 5), (1, 2, -2, 1, 1, 0, 0, 2, 0), (1, -1, 1, -1, -1, 2, 1, 1, 1), (1, 1, -2, -2, -1, 0, -1, -2, 6), (-1, 0, 0, 0, 0, 1, 1, 0, 1, 0), (-1, -1, -1, 0, 0, 1, 1, -1, 3), (1, 0, 1, 1, -1, 0, -1, -1, 2), (1, -1, -1, 1, 0, 0, 1, -1, 2), (-1, 1, 0, 0, 1, -1, 1, -1, 2), (2, 2, 3, -1, -1, 0, -1, 3, 0), (-1, -1, 1, 0, -1, -1, 1, -1, 4), (3, -1, -1, 0, 2, -1, 2, 2, 0), (-1, 1, 1, -1, 1, 2, 0, 1, 0), (1, -1, 0, 0, 1, 0, 1, 0, 0), (1, -1, 1, -1, 0, 0, 1, -1, 2), (-1, -1, 1, 0, -1, 1, -1, -1, 4), (2, 3, 1, 1, 1, 0, -3, 1, 0), (1, 0, -1, 1, -1, 0, -1, 1, 2), (0, 1, 2, -2, -1, 1, 0, 2, 1)
S5	(2, 1, 1, 1, -3, 1, 0, 2, 0), (-1, 2, -2, -1, 0, -2, 0, -1, 5), (0, 1, 0, 0, 1, 1, -1, 0, 0), (-1, -1, 1, -3, 3, -2, -1, 2, 5), (3, -1, -1, -1, 0, 2, 3, 1, 0), (-1, 1, 2, 0, -1, 2, -1, -2, 3), (0, -1, 0, 1, -1, -1, 0, 1, 2), (0, -1, 0, 0, 1, 1, 1, 0, 0), (-1, -1, -1, 0, -1, 0, -1, -1, 5), (1, 2, -2, 1, 0, 1, 0, 2, 0), (1, -1, 1, -1, 2, -1, 1, 1, 1), (-1, 0, 0, 0, 1, 1, 0, 1, 0), (1, 1, -2, -2, 0, -1, -1, -2, 6), (0, 1, 1, -1, 0, -1, 0, 1, 1), (-1, 0, 1, 0, 1, 1, 1, 0, 0), (1, 0, 1, 1, 0, -1, -1, -1, 2), (1, -1, -1, 1, 0, 0, 1, -1, 2), (-1, 1, -1, 0, -1, 0, 1, -1, 3), (0, -1, 1, -1, -1, -1, 1, -1, 4), (-1, -1, -1, 0, 1, 0, 1, -1, 3), (2, -1, -1, 0, -1, 1, 1, 1, 1), (1, -1, -1, 0, 1, -1, -1, 1, 3), (1, 1, 0, 0, 0, 1, -1, 0, 0), (5, 2, 4, 1, -4, 1, -2, 4, 0), (1, 1, 2, 2, 0, 1, 1, -2, 0), (-1, -1, 1, 1, -1, -1, 1, 0, 3), (0, -1, 1, 1, 1, -1, -1, -1, 3)
S6	(2, 1, 1, 1, -3, 0, 2, 1, 0), (-1, 2, -2, -1, 0, 0, -1, -2, 5), (0, 1, 0, 0, 1, -1, 0, 1, 0), (-1, -1, 1, -3, 3, -1, 2, -2, 5), (3, -1, -1, -1, 0, 3, 1, 2, 0), (-1, 4, 5, 3, -1, -1, -2, 5, 0), (0, -1, 0, 1, -1, 0, 1, -1, 2), (0, -1, 0, 0, 1, 1, 0, 1, 0), (1, 2, -2, 1, 0, 0, 2, 1, 0), (-1, -1, -1, 0, -1, -1, -1, 0, 5), (0, 1, 2, -2, 1, 0, 2, -1, 1), (-1, 0, 0, 0, 1, 0, 1, 1, 0), (1, 1, -2, -2, 0, -1, -2, -1, 6), (2, 0, 1, 1, -2, -1, 1, -1, 2), (-1, -1, 1, 0, -1, 1, -1, -1, 4), (-1, 1, -1, 0, -1, 1, -1, 0, 3), (1, -1, 1, -1, 0, 1, -1, 0, 2), (-1, -1, -1, 0, 1, 1, -1, 0, 3), (0, -1, 1, 1, 1, -1, -1, -1, 3), (1, -1, -1, 1, 0, 1, -1, 0, 2), (2, 3, 1, 1, 0, -3, 1, 1, 0), (2, -1, 2, 3, -1, 2, 3, -1, 0), (-1, 1, 1, -1, 2, 0, 1, 1, 0), (-1, -1, 0, 0, -1, -1, -1, 1, 4), (1, -1, 0, 0, 0, 1, 0, 1, 0), (3, -1, -1, 0, -1, 2, 2, 2, 0), (1, 0, -1, 1, 0, -1, 1, -1, 2)
S7	(2, 1, 1, 1, -3, 0, 2, 1, 0), (-1, 2, -2, -1, 0, 0, -1, -2, 5), (0, 1, 0, 0, 1, -1, 0, 1, 0), (-1, -1, 1, -3, 3, -1, 2, -2, 5), (3, -1, -1, -1, 0, 3, 1, 2, 0), (-1, 4, 5, 3, -1, -1, -2, 5, 0), (0, -1, 0, 1, -1, 0, 1, -1, 2), (0, -1, 0, 0, 1, 1, 0, 1, 0), (1, 2, -2, 1, 0, 0, 2, 1, 0), (-1, -1, -1, 0, -1, -1, -1, 0, 5), (0, 1, 2, -2, 1, 0, 2, -1, 1), (-1, 0, 0, 0, 1, 0, 1, 1, 0), (1, 1, -2, -2, 0, -1, -2, -1, 6), (2, 0, 1, 1, -2, -1, 1, -1, 2), (-1, -1, 1, 0, -1, 1, -1, -1, 4), (-1, 1, -1, 0, -1, 1, -1, 0, 3), (1, -1, 1, -1, 0, 1, -1, 0, 2), (-1, -1, -1, 0, 1, 1, -1, 0, 3), (0, -1, 1, 1, 1, -1, -1, -1, 3), (1, -1, -1, 1, 0, 1, -1, 0, 2), (2, 3, 1, 1, 0, -3, 1, 1, 0), (2, -1, 2, 3, -1, 2, 3, -1, 0), (-1, 1, 1, -1, 2, 0, 1, 1, 0), (-1, -1, 0, 0, -1, -1, -1, 1, 4), (1, -1, 0, 0, 0, 1, 0, 1, 0), (3, -1, -1, 0, -1, 2, 2, 2, 0), (1, 0, -1, 1, 0, -1, 1, -1, 2)
S8	(2, 1, 1, 1, 0, -3, 2, 1, 0), (-1, 2, -2, -1, 0, 0, -1, -2, 5), (0, 1, 0, 0, -1, 1, 0, 1, 0), (-1, -1, 1, -3, -1, 3, 2, -2, 5), (3, -1, -1, -1, 3, 0, 1, 2, 0), (-1, 4, 5, 3, -1, -1, -2, 5, 0), (0, -1, 0, 1, 0, -1, 1, -1, 2), (0, -1, 0, 0, 1, 1, 0, 1, 0), (1, 2, -2, 1, 0, 0, 2, 1, 0), (-1, -1, -1, 0, -1, -1, -1, 0, 5), (1, 1, 2, -1, -1, 0, 2, -1, 1), (1, 1, -2, -2, -1, 0, -2, -1, 6), (-1, 0, 0, 0, 0, 1, 1, 1, 0), (3, 2, 3, 3, -1, 0, -1, -1, 0), (1, -1, -1, 1, 1, 0, -1, 0, 2), (1, -1, 1, -1, 1, 2, 1, -1, 1), (-1, -1, -1, 0, 1, 1, -1, 0, 3), (-1, 1, 0, 0, 1, -1, -1, 1, 2), (0, -1, 1, -1, 1, -1, -1, -1, 4), (-1, 2, 1, -2, 0, 3, 2, 1, 0), (3, 2, -1, 3, -1, 0, 3, -1, 0), (-1, -1, 0, 0, -1, -1, -1, 1, 4), (-1, -1, 1, 0, -1, 1, -1, -1, 4), (2, 3, 1, 1, -3, 0, 1, 1, 0), (3, 1, 2, 2, 1, -4, 2, 1, 0), (-1, -1, 1, 1, 1, -1, 0, -1, 3), (-1, 1, 0, -1, 0, 0, 1, -1, 2)
S9	(2, 1, 1, 1, 1, 2, 0, -3, 0), (-1, 2, -2, -1, -2, -1, 0, 0, 5), (0, 1, 0, 0, 1, 0, -1, 1, 0), (-1, -1, 1, -3, -2, 2, -1, 3, 5), (3, -1, -1, -1, 2, 1, 3, 0, 0), (-1, 4, 5, 3, 5, -2, -1, -1, 0), (0, -1, 0, 1, -1, 1, 0, -1, 2), (0, -1, 0, 0, 1, 0, 1, 1, 0), (-1, -1, -1, 0, 0, -1, -1, -1, 5), (1, 2, -2, 1, 1, 2, 0, 0, 0), (1, -1, 1, -1, -1, 1, 1, 2, 1), (1, 1, -2, -2, -1, -2, -1, 0, 6), (-1, 0, 0, 0, 1, 1, 0, 1, 0), (-1, -1, -1, 0, 0, -1, 1, 1, 3), (-1, 1, 0, 0, 1, -1, 1, -1, 2), (1, -1, -1, 0, -1, 1, 0, 2), (-1, 1, 0, -1, -1, 1, 0, 0, 2), (6, 2, 3, 3, -1, 3, -1, -4, 0), (0, -1, 1, 1, -1, -1, -1, 1, 3), (0, -1, 1, -1, -1, -1, 1, -1, 4), (1, -1, 0, 0, 1, 0, 1, 0, 0), (2, 3, 1, 1, 1, 1, -3, 0, 0), (1, 0, -1, 1, -1, 1, -1, 0, 2), (3, -1, -1, -1, 2, 1, 2, -1, 1), (-1, 0, 1, -1, 1, 1, 1, 2, 0), (-1, 1, 2, 0, 2, -2, -1, -1, 3), (-1, -1, 1, 1, -1, 0, 1, -1, 3)