Automatic Search of Truncated Impossible Differentials for Word-Oriented Block Ciphers (Full Version)[⋆]

Shengbao Wu[1,2], Mingsheng Wang[3]

1. Institute of Software, Chinese Academy of Sciences, Beijing 100190, PO Box 8718, China
2. Graduate School of Chinese Academy of Sciences, Beijing 100190, China
3. State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

wushengbao@is.iscas.ac.cn
mingsheng_wang@yahoo.com.cn

Abstract. Impossible differential cryptanalysis is a powerful technique to recover the secret key of block ciphers by exploiting the fact that in block ciphers specific input and output differences are not compatible. This paper introduces a novel tool to search truncated impossible differentials for word-oriented block ciphers with bijective Sboxes. Our tool generalizes the earlier $\mathcal{U}$-method and the UID-method. It allows to reduce the gap between the best impossible differentials found by these methods and the best known differentials found by ad hoc methods that rely on cryptanalytic insights. The time and space complexities of our tool in judging an r-round truncated impossible differential are about $O(c \cdot l^4 \cdot r^4)$ and $O(c' \cdot l^2 \cdot r^2)$ respectively, where l is the number of words in the plaintext and c, c' are constants depending on the machine and the block cipher. In order to demonstrate the strength of our tool, we show that it does not only allow to automatically rediscover the longest truncated impossible differentials of many word-oriented block ciphers, but also finds new results. It independently rediscovers all 72 known truncated impossible differentials on 9-round CLEFIA. In addition, finds new truncated impossible differentials for AES, ARIA, Camellia without FL and FL^{-1} layers, E2, LBlock, MIBS and Piccolo. Although our tool does not improve the lengths of impossible differentials for existing block ciphers, it helps to close the gap between the best known results of previous tools and those of manual cryptanalysis.

Key words: word-oriented block ciphers, truncated impossible differentials, difference propagation system, $\mathcal{U}$-method, UID-method

1 Introduction

Impossible differential cryptanalysis is one of the most popular cryptanalytic tools for block ciphers. It was firstly proposed by Knudsen to analyze DEAL [14] in 1998 and then extended by Biham *et al.* to attack IDEA [5] and Skipjack [4]. Unlike traditional differential cryptanalysis [7], which uses differential characteristics with high probabilities to recover the right key, impossible differential cryptanalysis is a sieving method which exploits differentials with probability zero to retrieve the right key by filtering out all wrong keys. Until now, impossible differential cryptanalysis has shown its superiority over differential cryptanalysis in many block ciphers such as IDEA, Skipjack, CLEFIA [23] and AES [9].

Impossible differential cryptanalysis mainly consists of two steps. Firstly, an attacker tries to find impossible differentials, that is, differentials that never occur. Then, after gaining a list of plaintext-ciphertext pairs, the attacker guesses some subkey material involved in the outer rounds of the impossible differentials, and then partially encrypts/decrypts each plaintext-ciphertext pair to check whether the corresponding internal differences are identical to the input and output differences of the impossible differentials. Once that happens, the guessed subkey will be discarded. The right key will be recovered if we discard all wrong keys.

Several factors influence the success of impossible differential cryptanalysis, including the length of impossible differentials, specific input/output difference patterns and the strength of one-round encryption/decryption. Among them, the most important factor is the length of an impossible differential. The

[⋆] This is the full version of a paper accepted for publication at Indocrypt 2012.

longer the impossible differential is, the better the attack will be. Another important factor is the input/output difference pattern when two impossible differentials have the same length, because the new impossible differentials may well result in improved attacks [27, 17, 10]. If we find more impossible differentials, we can perform a successful attack or improve the time/data complexities of known attacks with higher possibilities.

In Indocrypt 2003, Kim *et al.* [13] proposed the $\mathcal{U}$-method to find impossible differentials for various block cipher structures with bijective round functions. The $\mathcal{U}$-method is based on the miss-in-the-middle approach (see 1-(a) of Fig. 1): it first constructs two differentials with probability one from the encryption and decryption direction and subsequently demonstrates some contradictions by combining them. In the $\mathcal{U}$-method, the propagation of differences in a block cipher (structure) is translated into simple matrix operations, and some inconsistent conditions are used to detect impossible differentials. Luo *et al.* [18] developed the idea of the $\mathcal{U}$-method and proposed a more general method —the UID-method. The UID-method removed some limitations in the $\mathcal{U}$-method and harnesses more inconsistent conditions to evaluate impossible differentials. So far, the $\mathcal{U}$-method and the UID-method have been employed as tool by some block cipher designers to evaluate the security of their designs against impossible differential cryptanalysis, for instance, LBlock [26] and Piccolo [22].

However, the $\mathcal{U}$-method and the UID-method only focus on finding impossible differentials with the miss-in-the-middle approach, which limits their power. An example is illustrated in 1-(b) of Fig. 1. In this case, we cannot detect a contradiction in the match point of the two probability-one differentials, but instead recover some useful information, which will feed back to the internal rounds to produce a contradiction. The $\mathcal{U}$-method and the UID-method fail to detect this kind of impossible differentials because they do not fully use the information in the match point.

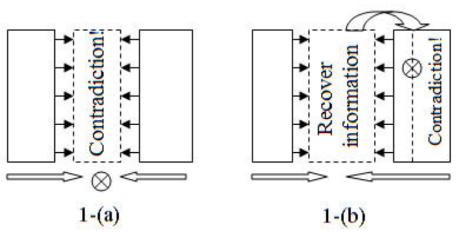


Fig. 1. Basic idea of the miss-in-the-middle approach (1-(a)) and impossible differentials with information feedback (1-(b))

Indeed, the longest impossible differentials of many block ciphers known so far are not found by the $\mathcal{U}$-method and the UID-method, but constructed by ad hoc approaches and the experience of cryptanalysts, such as 8-round MIBS [3], 6-round E2 [25], 8-round Camellia without the FL and FL^{-1} layers [27] [25] and all 72 9-round impossible differentials of CLEFIA listed in [24]. Almost all impossible differentials of these block ciphers fall under the model of 1-(b) of Fig. 1, which implies that they are beyond the abilities of the $\mathcal{U}$-method and the UID-method. Of course, ad hoc approaches also have some disadvantages. For example, it is very computationally intensive to find even one impossible differential, and the success of finding an impossible differential is highly dependent to the experience of a cryptanalyst. Thus, they are not efficient and systematic in practice. Especially in designing a block cipher, one has to modify his/her design and frequently re-evaluate its security against impossible differential cryptanalysis.

Our Contributions. In this work, we propose a new tool for automatically finding impossible differentials of word-oriented block ciphers with bijective S-boxes. The goals of our tool are to reduce the gap between previous automatic tools (i.e., the $\mathcal{U}$-method and the UID-method) and ad hoc approaches, and to provide an automated method to find impossible differentials with a reduced complexity. The development of attack algorithms to exploit these impossible differentials is outside the scope of this paper; we leave it for further work.

Unlike the miss-in-the-middle approach, which splits a block cipher into two parts factitiously, we treat it as an entirety. The inputs of our tool are some constraints of the plaintext difference and the ciphertext difference, and a system of equations that describes the propagation behavior when differences pass through the inner primitives of a block cipher. Then, our tool predicts information about unknown variables from the known ones iteratively, with probability one in each step. Finally, it outputs a flag indicating whether a truncated differential is impossible under several filtering conditions. The time and space complexities of our tool in judging an r-round truncated impossible differential are about $O(c \cdot l^4 \cdot r^4)$ and $O(c' \cdot l^2 \cdot r^2)$ respectively, where l is the number of words in the plaintext and c, c' are constants depending on the machine and the block cipher.

Although our tool does not improve the lengths of impossible differentials for existing block ciphers, it helps in reducing the gap between previous automatic tools and ad hoc approaches. Experimental results also indicate that our tool is efficient and systematic. It not only rediscovers the longest truncated impossible differentials of many word-oriented block ciphers known so far, but also finds new results. It independently rediscovers all 72 known truncated impossible differentials on 9-round CLEFIA [24]. In addition, besides the best results known so far, our tool finds new truncated impossible differentials for many other word-oriented block ciphers, such as the AES, Camellia [1] without FL and FL^{-1} layers, MIBS [11], LBlock [26], ARIA [15], E2 [12] and Piccolo [22]. The number of new truncated impossible differentials obtained by our tool is summarized in Table 1.

Table 1. Summary of new truncated impossible differentials (ID) obtained by our tool. Camellia* is a variant of Camellia without FL and FL^{-1} layers.

Block Cipher	Word unit	Previous results			In this paper		
		Round	No. of IDs	Method	Round	No. of IDs	New IDs
AES	byte	4 ([6, 20, 21, 2, 19])	269,554	ad hoc	4	3,608,100	3,338,546
ARIA	byte	4 ([27, 17, 10, 16])	156	ad hoc	4	94,416	94,260
Camellia*	byte	8 ([27],[25])	3	ad hoc	8	4	1
E2	byte	6 ([25])	1	ad hoc	6	56	55
MIBS	nibble	8 ([3])	2	ad hoc	8	8	6
LBlock	nibble	14 ([26])	64	$\mathcal{U}$-method	14	80	16
Piccolo	nibble	7 ([22])	1	$\mathcal{U}$-method	7	450	449

An interesting observation is that the $\mathcal{U}$-method and the UID-method are specific cases of our tool, and our tool is more powerful than them. An example is given to indicate that our tool can find longer impossible differentials than the $\mathcal{U}$-method and the UID-method. Thus, we expect that our tool is useful in evaluating the security of block ciphers against impossible differential cryptanalysis, especially when one tries to design a word-oriented block cipher with bijective Sboxes.

For the convenience of verifying our results, the source code for finding truncated impossible differentials of SPN ciphers and Feistel ciphers with SPN round functions are listed in Appendix B.

Outline of This Paper. In Sect. 2, we discuss how to build difference propagation systems, which describe the propagation behavior when differences pass through the inner primitives of block ciphers. In Sect. 3, we discuss our idea to find new impossible differentials. Then, a tool for automatically searching truncated impossible differentials is proposed in Sect. 4. Experimental results are also provided in this section. Finally, we compare our tool with the $\mathcal{U}$-method and the UID-method in Sect. 5 and conclude this paper in Sect. 6.

2 Difference Propagation System

Throughout this paper, we consider the exclusive-or difference, and we assume that: (1) $\mathcal{E}$ is an r-round word-oriented block cipher with block length $l \cdot s$ bits (where s is the bit length of a word), that is, the plaintext and the ciphertext of $\mathcal{E}$ are vectors in $\mathbb{F}_{2^s}^l$, and all inner operations of $\mathcal{E}$ consist only of calculations over $\mathbb{F}_{2^s}$; (2) bijective Sboxes over $\mathbb{F}_{2^s}$ are the only nonlinear primitives of $\mathcal{E}$; (3) all subkeys are exclusive-ored to the internal state. Thus, we do not consider the subkey addition operation since it does not influence the propagation of differences. Although some block ciphers do not satisfy all the above conditions, e.g., IDEA, we believe that similar ideas can also be applied, with some modifications.

In this section, we discuss how to build a system of equations that describes the propagation behavior when differences pass through the inner primitives of a word-oriented block cipher. This system will be called *difference propagation system* in the subsequent discussions.

2.1 Difference Propagation of Basic Primitives

Before studying block ciphers, we first investigate the difference propagation of four basic primitives which are often employed as parts of a word-oriented block cipher, namely the branching operation, the XOR-operation, the bijective Sbox layer and the linear permutation layer. These primitives are illustrated in (a), (b) and (c) of Fig. 2.

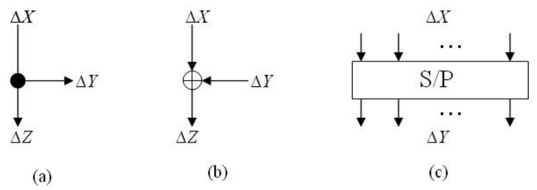


Fig. 2. Basic primitives of a block cipher: (a) the branching operation, (b) the XOR-operation and (c) the bijective Sbox/linear permutation layer

Suppose $\Delta X = (\Delta x_i)_{1\le i\le n}$, $\Delta Y = (\Delta y_i)_{1\le i\le n}$ and $\Delta Z = (\Delta z_i)_{1\le i\le n}$ are row vectors in $\mathbb{F}_{2^s}^n$, the difference propagation of basic primitives can be described as follows.

Lemma 1. *(The branching operation.) For a branching operation (see (a) of Fig. 3), we have $\Delta X = \Delta Y = \Delta Z$. This equation can be written as $2n$ linear equations $\Delta x_i \oplus \Delta y_i = 0$ and $\Delta x_i \oplus \Delta z_i = 0$.*

Lemma 2. *(The XOR-operation.) For an XOR-operation (see (b) of Fig. 3), we have $\Delta X \oplus \Delta Y = \Delta Z$. This equation can be written as n linear equations $\Delta x_i \oplus \Delta y_i \oplus \Delta z_i = 0$.*

Lemma 3. *(The linear permutation layer.) A linear permutation (see (c) of Fig. 3) has matrix representation $P = (p_{i,j})_{1\le i,j\le n}$ over $\mathbb{F}_{2^s}$, that is, $\Delta Y^T = P \cdot \Delta X^T$, where ΔX^T is the transposed vector of ΔX. This equation can be written as n linear equations $\Delta y_i \oplus \bigoplus_{j=1}^n p_{i,j} \cdot \Delta x_j = 0$.*

Lemma 4. *(The Sbox layer.) For an Sbox layer consisting of n bijective Sboxes $S_i : \mathbb{F}_{2^s} \to \mathbb{F}_{2^s}$ (see (c) of Fig. 3), we build n formal equations $\overline{S}_i(\Delta x_i, \Delta y_i) = 0$.*

Remark 1. Notice that $\overline{S}_i(\cdot,\cdot)$ is inherently a nonlinear map if we try to write its concrete expression, since S_i is a nonlinear bijective Sbox. Each pair $(\Delta x_i, \Delta y_i)$ with $Pr(\Delta x_i \to \Delta y_i) \neq 0$ in the Difference Distributed Table [7] of S_i is a solution of $\overline{S}_i(\Delta x_i, \Delta y_i) = 0$, which means that an input difference Δx_i may propagate to the output difference Δy_i. However, in Lemma 4, we build a formal equation for an Sbox without considering its concrete expression because the only property used in our tool is that it is bijective.

2.2 Build Difference Propagation Systems

The two most important classes of block ciphers are SPN ciphers and Feistel ciphers with SPN round functions (see (a) and (b) of Fig. 3). In this section, we choose them as examples to display how to build difference propagation systems.

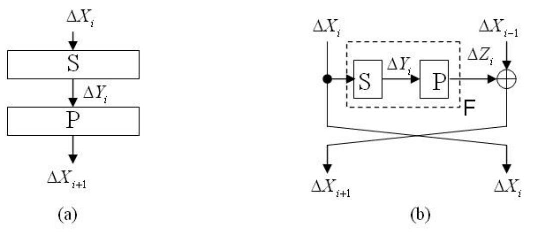


Fig. 3. The three most important classes of block ciphers: (a) SPN ciphers and (b) Feistel ciphers with SPN round function

SPN Cipher. One round of an SPN cipher typically has three layers (see (a) of Fig. 3): the SubkeyAddition layer, the Sbox layer and the linear permutation layer. As mentioned above, we omit the SubkeyAddition layer since it does not influence the propagation of differences. Additionally, we can omit the last linear permutation layer since it does not influence the length of an impossible differential.

We denote by $\Delta X_i = (\Delta X_{i,j})_{1\le j\le l}$ and $\Delta Y_i = (\Delta Y_{i,j})_{1\le j\le l}$ the differences before and after the Sbox layer of round i, respectively. Then, for an r-round SPN cipher, we build a difference propagation system as follows:

$$\begin{cases} \overline{S}_j(\Delta X_{i,j}, \Delta Y_{i,j}) = 0 \;\text{ for } 1\le i\le r \text{ and } 1\le j\le l \;, \\ \Delta X_{i+1}^T \oplus P\cdot \Delta Y_i^T = 0 \;\text{ for } 1\le i\le r-1 \;, \end{cases} \tag{1}$$

where P is the matrix of linear permutation layer. This system contains $2\cdot l\cdot r$ unknown variables, $l\cdot r$ formal (and nonlinear) equations from Sbox layers and $l\cdot(r-1)$ linear equations (using Lemma 3). ΔX_1 is the plaintext difference and ΔY_r is the ciphertext difference.

Feistel Cipher with SPN Round Functions. For an r-round Feistel cipher (see (b) of Fig. 3), we denote by $\Delta X_{i-1} = (\Delta X_{i-1,j})_{1\le j\le \frac{l}{2}}$ and $\Delta X_i = (\Delta X_{i,j})_{1\le j\le \frac{l}{2}}$ the differences of the right branch and the left branch of round i, respectively. Note that we assume that l is even. Thus, $(\Delta X_1, \Delta X_0)$ is the plaintext difference and $(\Delta X_{r+1}, \Delta X_r)$ is the ciphertext difference.

The SPN round function typically has three layers: the SubkeyAddition layer, the Sbox layer and the linear permutation layer. We introduce $\Delta Y_i = (\Delta Y_{i,j})_{1\le j\le \frac{l}{2}}$ to denote the difference after the Sbox layer of round i and $\Delta Z_i = (\Delta Z_{i,j})_{1\le j\le \frac{l}{2}}$ to represent the output difference of F function in round i. Then, we build a difference propagation system as follows:

$$\begin{cases} \overline{S}_j(\Delta X_{i,j}, \Delta Y_{i,j}) = 0 \;\text{ for } 1\le i\le r \text{ and } 1\le j\le \frac{l}{2} \;, \\ \Delta Z_i^T \oplus P\cdot \Delta Y_i^T = 0 \;\text{ for } 1\le i\le r \;, \\ \Delta X_{i-1}\oplus \Delta X_{i+1}\oplus \Delta Z_i = 0 \;\text{ for } 1\le i\le r \;, \end{cases} \tag{2}$$

where P is the matrix of the linear permutation layer in F. This system contains $\frac{l}{2}\cdot(3r+2)$ unknown variables, $\frac{l}{2}\cdot r$ formal (and nonlinear) equations and $l\cdot r$ linear equations (using Lemma 2 and Lemma 3).

If F is an SPS-type round function (e.g., the block cipher E2), that is, F consists of two consecutive SPN round functions, a similar system can be built by introducing new variables to represent the input difference and the output difference of the second Sbox layer.

Other Block Ciphers. Similar systems can be built for other block ciphers by introducing proper internal variables and combining Lemma 1 to Lemma 4 appropriately. For the convenience of building formal equations from Sbox layers and running our tool, different variables should be introduced before and after an Sbox layer to represent its input difference and output difference, respectively.

Remark 2. From Lemma 1 to Lemma 4, we know that a basic primitive costs $3l$ variables and provides $2l$ equations at most. Thus, for one round block cipher consisting of m basic primitives, we can build a difference propagation system with $3l \cdot m$ variables and $2l \cdot m$ equations in the worst case. In general, m is a small constant. For example, m is 2 in SPN ciphers (see (a) of Fig. 3) and m is 4 in Feistel ciphers with SPN round functions (see (b) of Fig. 3). Hence, for an r-round block cipher, we can build a difference propagation system with $O(c_1 \cdot r \cdot l)$ variables and $O(c_2 \cdot r \cdot l)$ equations, where c_1 and c_2 are constants depending on specific block ciphers.

3 Finding Impossible Differentials

In this section, we first introduce the basic idea of finding impossible differentials. Then, we discuss how to predict information from a given difference propagation system and how to detect contradictions. Finally, we briefly review the $\mathcal{U}$-method and the UID-method.

3.1 Basic Idea

For a given difference propagation system, each solution satisfying it is an r-round differential characteristic, without considering its probability. On the contrary, an r-round impossible differential is obtained if one demonstrates that there is no solution satisfying the system under given plaintext difference and ciphertext difference.

The idea of finding impossible differentials in this paper is simple: given some information of the plaintext difference and the ciphertext difference, we may predict the information of new variables according to the difference propagation system, yielding a new set of "known" variables. Then, new information may again be predicted from these "known" variables. This process will continue until we find a contradiction or we can no longer obtain any new information. Notice that every prediction we made is deterministic (i.e., with probability one), which implies that the system does not have any solution if a contradiction is found. In other words, we obtain an impossible differential if a contradiction is detected in the process of predicting information, under given plaintext difference and ciphertext difference.

3.2 Predict Information and Detect Contradictions

We can divide a difference propagation system into two subsystems — $\mathcal{L}$ and $\mathcal{NL}$. $\mathcal{L}$ includes all linear equations while $\mathcal{NL}$ contains all formal (and nonlinear) equations from bijective Sboxes. Then, the information can be obtained with probability one in the following two ways:

(i) Predict information from the linear system $\mathcal{L}$. If system $\mathcal{L}$ has solutions, then they can be solved by the *Gauss-Jordan Elimination* algorithm, which gets solutions by firstly reducing the augmented matrix of $\mathcal{L}$ to row echelon form using elementary row operations and then back-substituting until the entire solution is found. The reduced augmented matrix after the back-substituting step represents a linear system that is equivalent to the original. Then, we have the following lemma.

Lemma 5. *Suppose $\mathcal{L}$ has solutions and the reduced augmented matrix of $\mathcal{L}$ is obtained, then*

1) *If an affine equation with only one variable, that is, $\Delta X \oplus c = 0$ (c is a constant), is found in the reduced system of $\mathcal{L}$, we have $\Delta X = 0$ if $c = 0$ and $\Delta X \neq 0$ if $c \neq 0$.*
2) *If a linear equation with two variables, that is, $\Delta X \oplus \Delta Y = 0$, is found in the reduced system of $\mathcal{L}$, we have $\Delta X \neq 0$ if and only if $\Delta Y \neq 0$.*

Remark 3. After solving a system of linear equations $\mathcal{L}$ with the *Gauss-Jordan Elimination* algorithm, each variable that has a unique solution will be recovered, that is, all equations of the form $\Delta X \oplus c = 0$ (c is a constant) exist in the reduced augmented matrix of $\mathcal{L}$. Hence, Lemma 5 does not miss any information that falls under case 1).

(ii) Predict information from the nonlinear system $\mathcal{NL}$. We have

Lemma 6. *Suppose S is a bijective Sbox, ΔX is the input difference and ΔY is the output difference. Then, ΔX is zero (respectively, nonzero) if and only if ΔY is zero (respectively, nonzero).*

According to the basic idea of finding impossible differentials, the strategy of predicting information is clear now: predict information from system $\mathcal{L}$ and $\mathcal{NL}$ alternately until a contradiction is found or we can no longer obtain any new information. An impossible differential is detected by the following proposition:

Proposition 1. *We denote by ΔP and ΔC the plaintext difference and the ciphertext difference, respectively. Then, $\Delta P \rightarrow \Delta C$ is impossible if one of the following two situations happens:*

- **I.** *The linear system $\mathcal{L}$ does not have any solution. That is, the rank of its coefficient matrix is not equal to the rank of its augmented matrix.*
- **II.** *There exists a variable with both zero and nonzero values.*

Two tiny examples of the second case are given below.

Example 1. Suppose the equations $\Delta Y \oplus \Delta Z = 0$ and $S(\Delta X, \Delta Y) = 0$ are included in a difference propagation system, and we know that $\Delta X = 0$ and $\Delta Z \neq 0$ from the previous information. Then, in the next prediction, we know that $\Delta Y \neq 0$ from Lemma 5 while $\Delta Y = 0$ from Lemma 6, which is a contradiction.

Example 2. Suppose we have known that $[\Delta a_1, \Delta a_2, \ldots, \Delta a_n]$ is a nonzero vector. Then, a contradiction is found if we deduce that $\Delta a_i = 0$ (for $1 \leq i \leq n$) in the subsequent information predictions.

3.3 Related Work — the $\mathcal{U}$-method and the UID-method

In this section, we briefly review the $\mathcal{U}$-method and the UID-method. The specification of these tools can be found in [13] and [18].

Both the $\mathcal{U}$-method and the UID-method mainly have three steps in finding an impossible differential. First, both tools construct a characteristic matrix which describes the propagation of differences in one round encryption/decryption. For example, for one round of Feistel structure (see (b) of Fig. 3), we have $\Delta X_{i+1} = F(\Delta X_i) \oplus \Delta X_{i-1}$ and $\Delta X_i = \Delta X_i$, that is, $(\Delta X_{i+1}, \Delta X_i)^T = \mathcal{E} \cdot (\Delta X_i, \Delta X_{i-1})^T$, where $\mathcal{E} = \left(\begin{smallmatrix} F & 1 \\ 1 & 0 \end{smallmatrix}\right)$ is the characteristic matrix of one round encryption. The characteristic matrix $\mathcal{D}$ of one round decryption can be defined similarly. Secondly, the $\mathcal{U}$-method and the UID-method defined some operations to calculate the multiplications between a characteristic matrix and a vector, because the output difference after r_1-round encryptions (resp., r_2-round decryptions) can be described as $\Delta U = \mathcal{E}^{r_1} \cdot \Delta P^T$ (resp., $\Delta V = \mathcal{D}^{r_2} \cdot \Delta C^T$). Finally, suppose $\Delta U = (\Delta u_j)_{1 \leq j \leq l}$ and $\Delta V = (\Delta v_j)_{1 \leq j \leq l}$ are two vectors which should be combined in the miss-in-the-middle approach, the following filtering conditions are used to detect contradictions.

Definition 1. *(Definition 1 of [18]) Vectors ΔU and ΔV are inconsistent if there exists a subset $I \subseteq \{1, 2, \ldots, l\}$ such that $\oplus_{i \in I}(\Delta u_i \oplus \Delta v_i) \neq 0$, where Δu_i (respectively, Δv_i and $f = \oplus_{i \in I}(\Delta u_i \oplus \Delta v_i)$) is a linear XOR combination of the four types of differences: zero difference, nonzero fixed difference, nonzero unspecified difference and unknown difference. Especially, the $\mathcal{U}$-method always considers subsets that have exactly one index.*

There are two main differences between the $\mathcal{U}$-method and the UID-method. First, the UID-method relaxes the 1-Property (i.e., the number of 1 entries in each column of the characteristic matrix is zero or one) required in the $\mathcal{U}$-method. Secondly, the UID-method exploits a more general filtering condition to detect contradictions than the $\mathcal{U}$-method, which has shown in Definition 1. Thus, the UID-method is more general than the $\mathcal{U}$-method.

4 Algorithm to Find Truncated Impossible Differentials

In this section, we first sketch our algorithm in finding impossible differentials. Then, we discuss some details of our algorithm. Finally, experimental results and some discussions of our tool are introduced.

4.1 Sketch of Our Algorithm

After building a difference propagation system as (1) or (2), we first implement it in a computer. Then, we choose a set of promising $(\Delta P, \Delta C)$ pairs. Finally, for each of these pairs, our algorithm judges whether it is an impossible differential automatically, that is, our algorithm predicts information from the linear system $\mathcal{L}$ and the nonlinear system $\mathcal{NL}$ alternately until a contradiction is found or we can no longer obtain any new information. The outline of our algorithm is shown in Algorithm 1. $flag$ indicates whether $\Delta P \to \Delta C$ is impossible, and $index$ controls the termination of predicting information.

```
1  Implement the difference propagation system, i.e., $\mathcal{L}$ and $\mathcal{NL}$, on a computer;
2  for every pair of $(\Delta P, \Delta C)$ we choose do
3      $flag$:=false; $index$:=true;
4      while $index$ do
5          if System $\mathcal{L}$ does not have any solution then
6              $flag$:=true; $index$:=false;
           else
7              Predict information from the reduced augmented matrix of $\mathcal{L}$;
8              Predict information from the nonlinear system $\mathcal{NL}$;
9              if do not find new information then
10                 $index$:=false;
               else
11                 if find a variable with both zero and nonzero values then
12                     $flag$:=true; $index$:=false;
13     return $flag$;
```

Algorithm 1: The outline of our algorithm

In the subsequent sections, we will discuss some details of Algorithm 1, including how we implement a difference propagation system on a computer, the choices of the plaintext difference and the ciphertext difference, and a specific algorithm for automatically judging a truncated impossible differential. Λ_0 and Λ_1 are sets for storing variables with zero difference and nonzero differences in a difference propagation system.

4.2 Implementation of a Difference Propagation System

Since a difference propagation system is divided into two parts — systems $\mathcal{L}$ and $\mathcal{NL}$ in our tool, we discuss how to implement them on a computer, respectively.

Implement system $\mathcal{L}$ with a matrix. System $\mathcal{L}$ can be written formally as $A \cdot x = b$, where A, b and $B = [A|b]$ are called the *coefficient matrix*, *constant matrix* and *augmented matrix* of this system respectively, x is the set of *all* variables (in order) involved in the whole difference propagation system.

For a further step, the system $\mathcal{L}$ can be represented as a column vector form:

$$A_1 \cdot x_1 \oplus A_2 \cdot x_2 \oplus \cdots \oplus A_n \cdot x_n = b \ . \tag{3}$$

If we change the order of variables in (3), the coefficient matrix A will be changed accordingly. For example, the column positions of A_1 and A_2 in A will be exchanged if we exchange the order of x_1 and x_2 in x. Thus, matrix A (and B) is determined once the order of variables in x is fixed. In computer manipulations, we do not deal with the system in terms of equations but instead make use of the augmented matrix B.

In the following example, we show that matrix A can be easily constructed if we choose a proper order of variables.

Example 3. We consider Feistel ciphers with SPN round functions, that is, we need to generate the coefficient matrix of linear equations in (2). Firstly, we may simplify the second and third equations of (2) as

$$I_1 \cdot \Delta X_{i-1}^T \oplus I_2 \cdot \Delta X_{i+1}^T \oplus P \cdot \Delta Y_i^T = 0 \text{ for } 1 \le i \le r \ , \tag{4}$$

where $I_1 = I_2 = I$ and I is the identity matrix. Notice that variables ΔZ_is are eliminated in the simplified system. For a further step, the coefficient matrices of ΔX_{i-1}^T and ΔX_{i+1}^T may be substituted by other matrices according to the specification of the block ciphers.
Next, we fix the order of all variables in the simplified system as

$$x = [\Delta X_0, \Delta X_1, \Delta X_2, \Delta X_3, \ldots, \Delta X_{r-1}, \Delta X_r, \Delta X_{r+1}, \Delta Y_1, \Delta Y_2, \ldots, \Delta Y_r] \ . \tag{5}$$

Finally, (4) can be represented as

$$A \cdot x^T = 0 \ ,$$

where

$$A = \begin{pmatrix} I_1 & 0 & I_2 & 0 & \cdots & 0 & 0 & 0 & P & 0 & \cdots & 0 \\ 0 & I_1 & 0 & I_2 & \cdots & 0 & 0 & 0 & 0 & P & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & I_1 & 0 & I_2 & 0 & 0 & \cdots & P \end{pmatrix}$$

is a block matrix with r rows and $2r+2$ columns. Now, A is a highly structured matrix. We can easily generate it in a computer if we obtain the coefficient matrices of ΔX_{i-1}, ΔX_{i+1} and ΔY_i in (4).

For iterative block ciphers with other structures, similar techniques can be used to construct highly structured coefficient matrices.

A technical detail in solving a linear system is to dispose a zero variable. Without loss of generality, suppose $x_1 = 0$ in (3). Then, to avoid reduplicate judges of case 1) in Lemma 5, we can eliminate the term $A_1 \cdot x_1$ from (3) to obtain a system with one less variable, or keep the variable x_1 in the system while setting A_1 to a zero vector. Both methods keep the solutions of the other variables unchanged and need additional space to store the value of x_1. In our tool, we choose the latter manner for implementing system $\mathcal{NL}$ conveniently, and variables with zero value will be stored in set Λ_0. Notice that for the last method, B is a matrix with a fixed number of columns.

Implement system $\mathcal{NL}$ with a table. Once the order of variables in x is given, the i-th variable in x corresponds to the i-th column of A (and B). Then, we can store equations of system $\mathcal{NL}$ with a simple table using the column indexes of B. First, we initialize an empty table $\mathcal{T}$, then for each equation in $\mathcal{NL}$, we add an element $\{v_1, v_2\}$ to $\mathcal{T}$, where v_1, v_2 are two integers (i.e., the column indexes of B) indicating the two variables involved in the formal equation of an Sbox. For example, if the order of variables in a Feistel cipher is fixed as that in (5), then table $\mathcal{T}$ for formal equations in (2) is

$$\mathcal{T} = [\{\frac{l}{2} + \frac{l}{2} \cdot (i-1) + j, \frac{l}{2} \cdot (r+2) + \frac{l}{2} \cdot (i-1) + j\} : 1 \leq i \leq r, 1 \leq j \leq \frac{l}{2}] \ .$$

4.3 The Choices of the Plaintext Difference and the Ciphertext Difference

To judge an impossible differential, we have to first initialize some information, i.e., impose some constraints on the plaintext difference ΔP and the ciphertext difference ΔC. To avoid trivial results, an obvious constraint is that $\Delta P \neq 0$ and $\Delta C \neq 0$. However, besides this constraint, there are still $(2^{ls} - 1) \cdot (2^{ls} - 1)$ choices for all combinations of ΔP and ΔC. It is impossible to enumerate all of them on a personal computer, even for a 32-bit block cipher.

According to the property of word-oriented block ciphers, a natural way is to consider truncated differences. That is, for each word of $\Delta P = (\Delta P_i)_{1 \leq i \leq l}$ and $\Delta C = (\Delta C_i)_{1 \leq i \leq l}$, we assign an indicator to indicate the choice of its difference, representing by 0 a word without difference and by 1 a word with a difference. In such representation, the indicator vectors of ΔP and ΔC are row vectors in $\mathbb{F}_2^l$, and the number of all possible combinations of ΔP and ΔC is reduced to $(2^l - 1) \cdot (2^l - 1)$, which makes it feasible to enumerate all or a big part of them.

Notice that the value of ΔP_i (or ΔC_i) is zero if its indicator is 0, which implies that, in a difference propagation system, the variable ΔP_i (or ΔC_i) can be evaluated as zero. However, if the indicator of ΔP_i (or ΔC_i) is 1, we cannot evaluate any specific value for ΔP_i (or ΔC_i). In this case, we just leave ΔP_i (or ΔC_i) as an undetermined variable in a difference propagation system while storing its indicator information. In our tool, we will add the variable ΔP_i (or ΔC_i) to set Λ_1.

4.4 Algorithm for Judging a Truncated Impossible Differential

Let $p = [p_1, p_2, \ldots, p_l]$ (respectively, $c = [c_1, c_2, \ldots, c_l]$) be a vector, indicating the variable positions of ΔP (respectively, ΔC) in x. Fox example, $p = [\frac{l}{2}+1, \frac{l}{2}+2, \ldots, l, 1, 2, \ldots, \frac{l}{2}]$ if the order of variables in an r-round Feistel cipher is fixed as given in (5), since $[\Delta X_1, \Delta X_0]$ is the plaintext difference. $MulCol$(B,i,j) is a function that multiplies the j-th column of matrix B with element i, and $ColSubMatrix(B, i, j)$ is the submatrix of B with columns from i to j.

Our algorithm takes the matrix B, table $\mathcal{T}$, vector p, vector c, indicator vectors of ΔP and ΔC as inputs, and then predicts information as described in Algorithm 1 (i.e., from step 3 to step 12) automatically. Finally, it outputs a $flag$ indicating whether $\Delta P \to \Delta C$ is impossible, under the filter conditions listed in Proposition 1. The specific description of our algorithm is shown in Algorithm 2, and some sketches are listed as follows.

- In step 1, our tool initializes the sets Λ_0 and Λ_1, which are used for storing variables with zero difference and nonzero differences. We also introduce a new matrix B' to protect the matrix B against revision, because B can be re-used for different indicator vectors of ΔP and ΔC (we also discuss it below in the remarks of Algorithm 2).
- From step 2 to step 6, our tool scans the indicator vectors of ΔP and ΔC, and stores their information in Λ_0 or Λ_1. As mentioned in Sect. 4.2, if the value of a variable is zero, then the corresponding column of B' will be set to a zero vector.
- From step 7 to step 17, our tool predicts information from system $\mathcal{L}$ and $\mathcal{NL}$ alternately. This process is terminated if $index$ is false, which implies that a contradiction is found by using Proposition 1 or we can no longer obtain any new information. Especially,
  - in step 10 and step 11, our tool detects whether there is a type **I** contradiction of Proposition 1;
  - in step 12, it preforms *Gauss-Jordan Elimination* algorithm to obtain the reduced augmented matrix of the system $\mathcal{L}$;
  - in step 13, it predicts information from the reduced augmented matrix of the system $\mathcal{L}$ and the system $\mathcal{NL}$ by calling the subprogram **Predict_Info**;
  - in step 15 and step 16, it detects whether there is a type **II** contradiction of Proposition 1.
- From step 18 to step 36, our tool predicts information from the reduced augmented matrix of the system $\mathcal{L}$ and the system $\mathcal{NL}$. Especially,
  - in step 19, our tool defines a *temp* to represent whether it finally obtain some new information in the information prediction;
  - from step 20 to step 27, our tool predicts information from the reduced augmented matrix of the system $\mathcal{L}$, by scanning all linear equations in the system $\mathcal{L}$ and recovering information using Lemma 5;
  - from step 28 to step 33, our tool predicts information from the system $\mathcal{NL}$ using Lemma 6;
  - in step 34 and step 35, some new information is obtained if the number of variables in the set Λ_0 or Λ_1 is increased. If that happens, the returned *temp* will be true.

For an r-round word-oriented block cipher, we may obtain all r-round truncated impossible differentials by enumerating all possible nonzero indicator vectors of ΔP and ΔC. To obtain truncated impossible differentials with different lengths, it only needs to try different round numbers.

Some remarks on our algorithm are given below.

1. Since the encryption process of a block cipher is deterministic, for a fixed round number r, we only need to build the difference propagation system once. So, matrix B, table $\mathcal{T}$, vector p and vector c can be reused for different indicator vectors of ΔP and ΔC. Thus, we introduce a new matrix B' in Algorithm 2 to protect the matrix B against revision.
2. Besides indicator vectors of ΔP and ΔC, some linear constraints between nonzero variables in ΔP and ΔC can also be added while selecting the initial constraints. Our tool still works in this case by translating all linear constraints to row matrices firstly and then adding these rows to matrix B'.
3. Our tool only uses the bijective property of an Sbox to predict information, without solving nonlinear systems. The most time-consuming steps are calculating the rank of a matrix and solving a linear system.
4. Filter condition $\Lambda_0 \cap \Lambda_1 \neq \emptyset$ used in Algorithm 2 is a special form of case **II** listed in Proposition 1. In some block ciphers, e.g., AES, we may use the filtering condition of Example 2 to find contradictions.

```
Input: Matrix $B$, table $\mathcal{T}$, vector $p$, vector $c$ and indicator vectors of $\Delta P$ and $\Delta C$.
Output: A $flag$ indicates whether $\Delta P \to \Delta C$ is impossible.
1  $B' := B$; $\Lambda_0 := \emptyset$; $\Lambda_1 := \emptyset$; $n :=$NumberOfColumns($B'$);
   // Initialize the truncated information of $\Delta P$ and $\Delta C$ ...
   // ... using their indicator vectors SdeltaP and SdeltaC.
2  for $i := 1$ to $l$ do
3      $\Lambda_1 := \Lambda_1 \cup \{p_i\}$ if SdeltaP=1;
4      $\Lambda_0 := \Lambda_0 \cup \{p_i\}$ and $B' :=$MulCol($B', 0, p_i$) if SdeltaP=0;
5      $\Lambda_1 := \Lambda_1 \cup \{c_i\}$ if SdeltaC=1;
6      $\Lambda_0 := \Lambda_0 \cup \{c_i\}$ and $B' :=$MulCol($B', 0, c_i$) if SdeltaC=0;
   // Predict information and find contradictions.
7  $flag :=$ false; $index :=$true;
8  while $index$ do
9      $A' :=$ColSubMatrix($B', 1, n-1$);
10     if $rank(A') \neq rank(B')$ then
11         $flag :=$ true; $index :=$false;
       else
           // Gauss-Jordan Elimination.
12         $B' :=$Reduced-row-echelon-form-of($B'$);
13         $< B', \Lambda_0, \Lambda_1, temp >:=$ Predict_Info($B', \mathcal{T}, \Lambda_0, \Lambda_1$);
14         $index := temp$;
15         if $\Lambda_0 \cap \Lambda_1 \neq \emptyset$ then
16             $flag :=$ true; $index :=$false;
17 return $flag$.

18 Predict_Info($B', \mathcal{T}, \Lambda_0, \Lambda_1$);
19 $n_0 := |\Lambda_0|$; $n_1 := |\Lambda_1|$; $temp :=$false; $n :=$NumberOfColumns($B'$);
   // Predict information from the reduced row echelon form of $B'$ using Lemma 5.
20 for $i := 1$ to NumberOfRows($B'$) do
21     $S := \emptyset$;
22     for $j := 1$ to $n-1$ do
23         $S := S \cup \{j\}$ if $B'[i,j] \neq 0$;
       // A linear equation with form $\Delta X = 0$.
24     if $|S| = 1$ and $B'[i,n] = 0$ then
25         $\Lambda_0 := \Lambda_0 \cup S$; $B' :=$MulCol($B', 0, j$) for $j \in S$ ;
       // An equation with form $\Delta X \oplus c = 0$ $(c \neq 0)$ or $\Delta X \oplus \Delta Y = 0$.
26     if $(|S| = 1$ and $B'[i,n] \neq 0)$ or $(|S| = 2$, $B'[i,n] = 0$ and $S \cap \Lambda_1 \neq \emptyset)$ then
27         $\Lambda_1 := \Lambda_1 \cup S$;
   // Scan table $\mathcal{T}$ and use Lemma 6 to predict information.
28 for $j := 1$ to NumberOfElements($\mathcal{T}$) do
29     if $\mathcal{T}[j] \cap \Lambda_0 \neq \emptyset$ and $\mathcal{T}[j] \setminus \Lambda_0 \neq \emptyset$ then
30         $B' :=$MulCol($B', 0, e$) for $e \in \mathcal{T}[j] \setminus \Lambda_0$;
31         $\Lambda_0 := \Lambda_0 \cup \mathcal{T}[j]$;
32     if $\mathcal{T}[j] \cap \Lambda_1 \neq \emptyset$ then
33         $\Lambda_1 := \Lambda_1 \cup \mathcal{T}[j]$;
34 if $|\Lambda_0| > n_0$ or $|\Lambda_1| > n_1$ then
35     $temp :=$true;
36 return $< B', \Lambda_0, \Lambda_1, temp >$.
```

Algorithm 2: Automatically evaluation of a truncated impossible differential

Algorithm Complexity. Suppose B' is a matrix with M rows and N columns. The time consumption for computing $rank(A')$, $rank(B')$ and *Gauss-Jordan Elimination* is about $O(M^2 \cdot N)$, and the time consumption for the subprogram **Predict_Info** is about $O(M \cdot N)$. Algorithm 2 terminates if *index* is false. Otherwise, at least one of sets Λ_0 and Λ_1 is updated after each while loop. According to the pigeonhole principle, there must be a type **II** contradiction mentioned in Proposition 1 when $|\Lambda_0| + |\Lambda_1| > N$. Therefore, while loop runs $N + 1$ times at most. In summary, the time complexity of judging a truncated impossible differential does not exceed $O(M^2 \cdot N^2)$. From Remark 2, we know M is about $O(c_1 \cdot l \cdot r)$ and N is about $O(c_2 \cdot l \cdot r)$. Thus, the time complexity of Algorithm 2 is about $O(c \cdot l^4 \cdot r^4)$, where c is a constant depending on the machine and the block cipher. The space complexity of Algorithm 2 is dominated by storing the matrices B, B' and A'. Thus, the space complexity is about $O(M \cdot N)$, that is, $O(c' \cdot l^2 \cdot r^2)$, where c' is also a constant depending on the machine and the block cipher.

4.5 Experimental Results

We apply our tool to find truncated impossible differentials for various byte-(or nibble-)oriented block ciphers, such as the AES, CLEFIA, E2, Camellia without FL and FL^{-1} layers, ARIA, LBlock, MIBS and Piccolo. All of these block ciphers have $l = 16$. We may classify them into three groups according to their underlying structures: SPN ciphers (AES and ARIA), Feistel ciphers (Camellia without FL and FL^{-1} layers, LBlock, MIBS and E2), and generalized Feistel ciphers (CLEFIA and Piccolo). The results of this section are obtained on a 2.66 GHz processor with MAGMA package [8]. For the convenience of verifying our results, the source code for searching the truncated impossible differentials of some block ciphers are listed in Appendix B.

SPN Ciphers. For the AES, our tool finds 3,608,100 4-round truncated impossible differentials in hours using the filter conditions of Example 2. All impossible differentials of AES follow an inherent rule. We can describe them explicitly (see Appendix A). The classes of impossible differentials discussed in [6, 20, 21, 2, 19] are included in our result set. For the ARIA, besides the results shown in [27, 17, 10], our tool finds 94,260 new 4-round truncated impossible differentials in about two weeks.

Feistel Ciphers. Since there is always a five round impossible differential $(0, \Delta X_0) \not\to_5 (\Delta X_6, 0)$ for the Feistel structure with bijective round function when $\Delta X_0 = \Delta X_6$ [13], we focus on finding impossible differentials $(0, \Delta X_0) \not\to_r (\Delta X_{r+1}, 0)$ with $r \geq 6$. The search space is reduced to $(2^{\frac{l}{2}} - 1) \cdot (2^{\frac{l}{2}} - 1)$. Results for Camellia without FL and FL^{-1} layers, MIBS, LBlock and E2 can be obtained in a few hours. Notice that E2 is a Feistel cipher with SPS-type round function.

Suppose I is a set, e_I denotes a vector whose i-th (for all $i \in I$) component is a nonzero difference while other components are zero difference. Then, the results obtained by our tool are illustrated in Table 2, where (e_I, e_J) is the shortened form of $(0, \Delta X_1) \not\to_r (\Delta X_{r+1}, 0)$. For example, (e_3, e_5) in the results of MIBS means that

$$(00000000, 00\alpha00000) \not\to_8 (0000\beta000, 00000000) \ , \tag{6}$$

where α and β are nonzero 4-bit differences.

Generalized Feistel Ciphers. CLEFIA is a 4-branch generalized Feistel cipher with SPN round functions, and Piccolo has a variant of 4-branch generalized Feistel structure with an SPS-type round function. The specification of these block ciphers can be found in [23] and [22].

The plaintext differences and the ciphertext differences of CLEFIA and Piccolo can be divided into four branches with four words in each branch. In this paper, we focus on finding truncated impossible differentials of the form $\Delta P = \Delta C = (0000, ????, 0000, ????) \in \mathbb{F}_2^{16} \setminus \{0\}$, where "?" can be zero difference or a nonzero difference. The number of choices is reduced to $(2^8 - 1) \cdot (2^8 - 1)$ now. Our computer enumerates them in a few hours. For the CLEFIA, our tool independently rediscovers all 72 known truncated impossible differentials listed in [24] but does not find new results. In [22], the designers of Piccolo claimed that they find a 7-round impossible differential using modified $\mathcal{U}$-method. However, the

Table 2. Truncated impossible differentials $(0, \Delta X_1) \nrightarrow_r (\Delta X_{r+1}, 0)$ for some Feistel ciphers. Previous results are marked with bold type. Camellia* is a variant of Camellia without FL and FL^{-1} layers.

Ciphers	r	No.	Truncated Impossible Differentials
Camellia*	8	4	$(\mathbf{e_1}, \mathbf{e_1})$, $(\mathbf{e_2}, \mathbf{e_2})$, $(\mathbf{e_3}, \mathbf{e_3})$[27], (e_4, e_4).
MIBS	8	6	$(\mathbf{e_3}, \mathbf{e_8})$, $(\mathbf{e_7}, \mathbf{e_5})$ [3], (e_3, e_5), (e_5, e_3), (e_5, e_7) , (e_8, e_3).
LBlock	14	80	$(\mathbf{e_i}, \mathbf{e_j})$ [26], for $1 \le i, j \le 8$. $(e_1, e_{\{4,6\}})$, $(e_2, e_{\{2,8\}})$, $(e_3, e_{\{5,7\}})$, $(e_4, e_{\{5,7\}})$, $(e_5, e_{\{2,8\}})$, $(e_6, e_{\{4,6\}})$, $(e_7, e_{\{1,3\}})$, $(e_8, e_{\{1,3\}})$, $(e_{\{1,2\}}, e_1)$, $(e_{\{3,8\}}, e_2)$, $(e_{\{5,6\}}, e_3)$, $(e_{\{3,8\}}, e_4)$, $(e_{\{5,6\}}, e_5)$, $(e_{\{4,7\}}, e_6)$, $(e_{\{1,2\}}, e_7)$, $(e_{\{4,7\}}, e_8)$.
E2	6	56	(e_1, e_1), $(\mathbf{e_1}, \mathbf{e_3})$[25], (e_1, e_5), (e_1, e_6), (e_1, e_8), (e_2, e_5), (e_2, e_6), (e_2, e_7), (e_2, e_8), (e_3, e_1), (e_3, e_3), (e_3, e_6), (e_3, e_8), (e_4, e_5), (e_4, e_6), (e_4, e_8), (e_5, e_1), (e_5, e_2), (e_5, e_3), (e_5, e_4), (e_5, e_5), (e_5, e_6), (e_5, e_8), (e_6, e_1), (e_6, e_2), (e_6, e_3), (e_6, e_4), (e_6, e_5), (e_6, e_6), (e_6, e_7), (e_7, e_2), (e_7, e_6), (e_7, e_7), (e_7, e_8), (e_8, e_1), (e_8, e_2), (e_8, e_3), (e_8, e_4), (e_8, e_5), (e_8, e_7), $(e_5, e_{\{2,7\}})$, $(e_5, e_{\{4,7\}})$, $(e_6, e_{\{1,8\}})$, $(e_6, e_{\{3,8\}})$, $(e_7, e_{\{2,5\}})$, $(e_7, e_{\{5,8\}})$, $(e_8, e_{\{1,6\}})$, $(e_8, e_{\{3,6\}})$, $(e_{\{2,7\}}, e_5)$, $(e_{\{4,7\}}, e_5)$, $(e_{\{1,8\}}, e_6)$, $(e_{\{3,8\}}, e_6)$, $(e_{\{2,5\}}, e_7)$, $(e_{\{5,8\}}, e_7)$, $(e_{\{1,6\}}, e_8)$, $(e_{\{3,6\}}, e_8)$.

specification is not shown. With the application of our tool, we find 450 truncated impossible differentials for 7-round Piccolo. They are distributed in two classes

$$(0000, 00\alpha_1\alpha_2, 0000, \alpha_3\alpha_4 00) \nrightarrow_7 (0000, \beta_1\beta_2 00, 0000, 00\beta_3\beta_4) \ , \tag{7}$$

$$(0000, \alpha_1\alpha_2 00, 0000, 00\alpha_3\alpha_4) \nrightarrow_7 (0000, 00\beta_1\beta_2, 0000, \beta_3\beta_4 00) \ , \tag{8}$$

where each α_i (respectively, β_i) can be a zero difference or a nonzero difference, and at least one of α_i (respectively, β_i) is nonzero.

4.6 Discussions

Despite the fact that impossible differential cryptanalysis has been proposed more than one decade, many problems are still unsolved. Until now, we even do not know how to prove whether an r-round impossible differential is one of the longest impossible differentials of a block cipher, since this implies that one can prove that there is not any $(r+1)$-round impossible differential. In other words, to prove this, one needs to show that there is an $(r+1)$-round differential characteristic satisfying given ΔP and ΔC, whatever the choices of ΔP and ΔC ($\Delta P \neq 0$ and $\Delta C \neq 0$). So far providing such a proof is beyond the state of the art.

For our tool, the only thing we can confirm is that impossible differentials obtained by our tool must be correct if one implements a difference propagation system and our algorithm on a computer correctly, because the conditions of judging an impossible differential are sufficient conditions. The ability and efficiency of our tool have been experimentally verified for many block ciphers.

However, our tool still has some limitations, which are also unsolved by the $\mathcal{U}$-method and the UID-method. First, the choice of truncated difference may result in missing some impossible differentials. For example, $\Delta P \to \Delta C$ should be an impossible differential if ΔP and ΔC are evaluated to some specific values, but our tool may miss it if we only know the indicator information of ΔP and ΔC. Secondly, our tool is not able to exploit any properties of the Sboxes beyond the fact that they are bijective. Thus, we may also miss some impossible differentials if we need some specific properties of an Sbox to detect these impossible differentials. Finally, our tool may fail if a block cipher is not word-oriented or uses an Sbox that is not bijective.

5 Comparison of Our Tool with the $\mathcal{U}$-method and the UID-method

In this section, we investigate the relationship of our tool with the $\mathcal{U}$-method and the UID-method.

On the one hand, we have

Proposition 2. *The $\mathcal{U}$-method and the UID-method are specific cases of our tool.*

Proof. First, the characteristic matrices defined by the $\mathcal{U}$-method and the UID-method are specific forms of our difference propagation systems.

Secondly, the operations used in these tools (see Table 2 and 3 in [13], Table 1 and Definition 3 in [18]) are included in Lemma 5, Lemma 6 and the difference propagation system. For example, in Table 3 of [13], the operation $1 \cdot 1_F = 1$, i.e., the result of a nonzero difference propagating through a nonlinear bijective function (Sbox) is also a nonzero difference, is described in Lemma 6.

Finally, we show that a contradiction that is detected by the $\mathcal{U}$-method or the UID-method can also be found by our tool. From Definition 1, to detect an impossible differential, we have to show that the final value of $f = \oplus_{i \in I}(\Delta u_i \oplus \Delta v_i)$ is a nonzero difference. However, the value of f is unpredictable if its final expression contains a term with an unknown difference or two terms with nonzero unspecified differences. And f is useless for us if it is a zero difference, or it contains a term with a nonzero fixed difference and a term with a nonzero unspecified difference. Thus, a useful f only consists of a single term with a nonzero fixed difference or a nonzero unspecified difference. In our tool, we can build l linear equations, i.e., $\Delta u_i \oplus \Delta v_i = 0$ for $1 \le i \le l$, in a difference propagation system by introducing proper internal variables. These linear equations are included in the subsystem $\mathcal{L}$, then we have

(1) $f = \Delta c$, where Δc is a nonzero fixed difference. That is, a nonzero fixed difference is in the linear space spanned by $\{\Delta u_i \oplus \Delta v_i : 1 \le i \le l\}$. In this case, we conclude that the rank of the coefficient matrix of $\mathcal{L}$ is not equal to the rank of the augmented matrix of $\mathcal{L}$. Thus, an impossible differential is detected by case **I** of Proposition 1.
(2) $f = \Delta a$, where Δa is a nonzero unspecified difference. In this case, we have known that there is a variable with nonzero unspecified difference, i.e., $\Delta a \neq 0$, from previous information. Meanwhile, we obtain $\Delta a = 0$ if linear system $\mathcal{L}$ is solved by the *Gauss-Jordan Elimination* algorithm. Thus, an impossible differential is detected by case **II** of Proposition 1.

In summary, the $\mathcal{U}$-method and the UID-method are specific cases of our tool. □

On the other hand, our tool is more powerful than the $\mathcal{U}$-method and the UID-method. An example, i.e., the 8-round impossible differential (e_3, e_5) of MIBS, is given to illustrate that our tool finds longer impossible differentials. The $\mathcal{U}$-method and the UID-method fail to detect this impossible differential because they do not fully use the information hiding in the match point of the two probability-one differentials (see Fig. 1).

Example 4. MIBS is a nibble-oriented block cipher following the Feistel structure. It operates on 64-bit blocks, uses keys of 64 or 80 bits, and iterates 32 rounds for both key sizes. Therefore, the plaintext/ciphertext can be represented with a vector with 16 nibbles, i.e., $l = 16$.

The difference propagation system of r-round MIBS is

$$\begin{cases} \overline{S}(\Delta X_{i,j}, \Delta Y_{i,j}) = 0 \text{ for } 1 \le i \le r \text{ and } 1 \le j \le 8 \ , \\ I_1 \cdot \Delta X_{i-1}^T \oplus I_2 \cdot \Delta X_{i+1}^T \oplus P \cdot \Delta Y_i^T = 0 \text{ for } 1 \le i \le r \ , \end{cases} \tag{9}$$

where I_1 and I_2 are the identity matrix, and P is the linear permutation layer in round functions. Here,

$$P = \begin{pmatrix} 1&1&0&1&1&0&1&1 \\ 0&1&1&1&1&1&1&0 \\ 1&1&1&0&1&1&0&1 \\ 0&1&1&1&0&0&1&1 \\ 1&0&1&1&1&0&0&1 \\ 1&1&0&1&1&1&0&0 \\ 1&1&1&0&0&1&1&0 \\ 1&0&1&1&0&1&1&1 \end{pmatrix} \text{ and its inverse } P^{-1} = \begin{pmatrix} 0&1&0&1&0&1&1&1 \\ 1&0&0&1&1&0&1&1 \\ 1&0&1&1&1&1&0&0 \\ 0&1&1&0&1&1&1&0 \\ 0&1&1&1&1&0&1&1 \\ 1&1&0&1&1&1&0&1 \\ 1&0&1&0&1&1&1&1 \\ 1&1&1&1&0&1&1&0 \end{pmatrix} .$$

Now, for given $(\Delta X_1 \| \Delta X_0) = (0,0,0,0,0,0,0,0 \| 0,0,a,0,0,0,0,0)$ $(a \neq 0)$ and $(\Delta X_9 \| \Delta X_8) = (0,0,0,0,g,0,0,0 \| 0,0,0,0,0,0,0,0)$ $(g \neq 0)$, we can deduce the internal state of MIBS round by round (see Fig. 4). After 4-round deductions in the forward direction, we obtain that the output difference of round 4 is

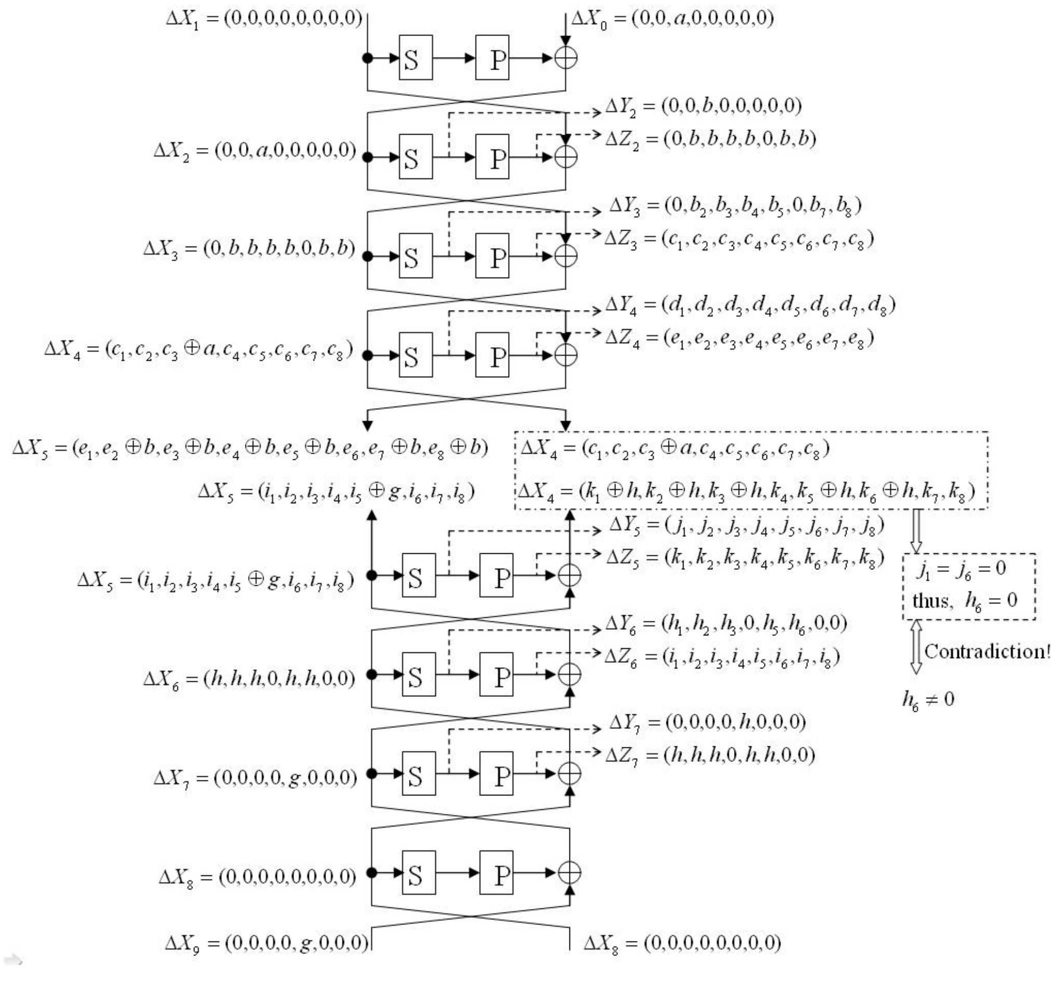
$$(\Delta X_5 \| \Delta X_4) = (e_1, e_2 \oplus b, e_3 \oplus b, e_4 \oplus b, e_5 \oplus b, e_6, e_7 \oplus b, e_8 \oplus b \| c_1, c_2, c_3 \oplus a, c_4, c_5, c_6, c_7, c_8) \ , \quad (10)$$

where a, b and b_t ($t \in \{2, 3, 4, 5, 7, 8\}$) are nonzero differences while c_t, d_t and e_t ($1 \le t \le 8$) are unspecific differences. While in the backward direction, after 4-round deductions, we obtain that the input difference of round 5 is

$$(\Delta X_5 \| \Delta X_4) = (i_1, i_2, i_3, i_4, i_5 \oplus g, i_6, i_7, i_8 \| k_1 \oplus h, k_2 \oplus h, k_3 \oplus h, k_4, k_5 \oplus h, k_6 \oplus h, k_7, k_8) \ , \quad (11)$$

where g, h and h_t ($t \in \{1, 2, 3, 5, 6\}$) are nonzero differences while i_i, j_t and k_t ($1 \le t \le 8$) are unknown differences.

Now, if we combine (10) and (11) together and try to find a contradiction by the filtering conditions of the $\mathcal{U}$-method and the UID-method, we get nothing because e_t and k_t ($1 \le t \le 8$) are unknown differences. Thus, the $\mathcal{U}$-method and the UID-method can not detect this impossible differential.



$\Delta X_1 = (0,0,0,0,0,0,0,0)$
$\Delta X_0 = (0,0,a,0,0,0,0,0)$
S
P
$\Delta Y_2 = (0,0,b,0,0,0,0,0)$
$\Delta Z_2 = (0,b,b,b,b,0,b,b)$
$\Delta X_2 = (0,0,a,0,0,0,0,0)$
$\Delta Y_3 = (0,b_2,b_3,b_4,b_5,0,b_7,b_8)$
$\Delta Z_3 = (c_1,c_2,c_3,c_4,c_5,c_6,c_7,c_8)$
$\Delta X_3 = (0,b,b,b,b,0,b,b)$
$\Delta Y_4 = (d_1,d_2,d_3,d_4,d_5,d_6,d_7,d_8)$
$\Delta Z_4 = (e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8)$
$\Delta X_4 = (c_1,c_2,c_3 \oplus a,c_4,c_5,c_6,c_7,c_8)$
$\Delta X_5 = (e_1,e_2 \oplus b,e_3 \oplus b,e_4 \oplus b,e_5 \oplus b,e_6,e_7 \oplus b,e_8 \oplus b)$
$\Delta X_4 = (c_1,c_2,c_3 \oplus a,c_4,c_5,c_6,c_7,c_8)$
$\Delta X_5 = (i_1,i_2,i_3,i_4,i_5 \oplus g,i_6,i_7,i_8)$
$\Delta X_4 = (k_1 \oplus h,k_2 \oplus h,k_3 \oplus h,k_4,k_5 \oplus h,k_6 \oplus h,k_7,k_8)$
$\Delta Y_5 = (j_1,j_2,j_3,j_4,j_5,j_6,j_7,j_8)$
$\Delta Z_5 = (k_1,k_2,k_3,k_4,k_5,k_6,k_7,k_8)$
$\Delta X_5 = (i_1,i_2,i_3,i_4,i_5 \oplus g,i_6,i_7,i_8)$
$j_1 = j_6 = 0$
thus, $h_6 = 0$
Contradiction!
$h_6 \neq 0$
$\Delta Y_6 = (h_1,h_2,h_3,0,h_5,h_6,0,0)$
$\Delta Z_6 = (i_1,i_2,i_3,i_4,i_5,i_6,i_7,i_8)$
$\Delta X_6 = (h,h,h,0,h,h,0,0)$
$\Delta Y_7 = (0,0,0,0,h,0,0,0)$
$\Delta Z_7 = (h,h,h,0,h,h,0,0)$
$\Delta X_7 = (0,0,0,0,g,0,0,0)$
$\Delta X_8 = (0,0,0,0,0,0,0,0)$
$\Delta X_9 = (0,0,0,0,g,0,0,0)$
$\Delta X_8 = (0,0,0,0,0,0,0,0)$

Fig. 4. A truncated impossible differential of 8-round MIBS

However, our tool retrieves some important information by solving a system of linear equations deduced from (10) and (11). Notice that from (10), we know that $\Delta X_4^T = \Delta Z_3^T \oplus \Delta X_2^T = P \cdot \Delta Y_3^T \oplus \Delta X_2^T$, and we also have $\Delta X_4^T = \Delta Z_5^T \oplus \Delta X_6^T = P \cdot \Delta Y_5^T \oplus P \cdot \Delta Y_7^T$ from (11). Thus, we get

$$P \cdot \Delta Y_3^T \oplus \Delta X_2^T \oplus P \cdot \Delta Y_5^T \oplus P \cdot \Delta Y_7^T = 0 \ . \quad (12)$$

It is equivalent to solve the following linear system

$$\Delta Y_3^T \oplus P^{-1} \cdot \Delta X_2^T \oplus \Delta Y_5^T \oplus \Delta Y_7^T = 0 \ . \tag{13}$$

Since $\Delta Y_3^T = (0, b_2, b_3, b_4, b_5, 0, b_7, b_8)^T$, $P^{-1} \cdot \Delta X_2^T = (0, 0, a, a, a, 0, a, a)^T$ and $\Delta Y_7^T = (0, 0, 0, 0, h, 0, 0, 0)^T$, we deduce that the first nibble and the six nibble of ΔY_5 are zero, that is, $j_1 = j_6 = 0$. From Lemma 6, we know $i_1 = i_6 = 0$. For a further step, since $\Delta Z_6^T = P \cdot \Delta Y_6^T$, we have

$$\begin{aligned} i_1 &= h_1 \oplus h_2 \oplus h_5 \ , \\ i_6 &= h_1 \oplus h_2 \oplus h_5 \oplus h_6 \ . \end{aligned}$$

Now, $i_1 = i_6 = 0$ implies that $h_6 = 0$, which contradicts with $h_6 \neq 0$.

We observe that the $\mathcal{U}$-method and UID-method fail to find any of the 6 impossible differentials of 8-round MIBS listed in Table 2. Similarly, all 4 impossible differentials of 8-round Camellia without FL and FL^{-1} layers are also beyond the abilities of the $\mathcal{U}$-method and UID-method.

6 Conclusions

This paper presents an automated tool for finding truncated impossible differentials of word-oriented block ciphers with bijective S-boxes. The $\mathcal{U}$-method and the UID-method are specific cases of our tool. Although our tool does not improve the lengths of impossible differentials for existing block ciphers, it reduces the gap between previous automated tools (i.e., the $\mathcal{U}$-method and the UID-method) and ad hoc approaches. With the application of our tool, we not only rediscover the longest truncated impossible differentials of many byte-(and nibble-)oriented block ciphers known so far, but also find new results. Although it is not clear whether new results found by our tool are useful to improve known attacks or not, they bring more choices in designing attack algorithms. Hence it may be possible to improve the known attacks.

To obtain a better tool in the future, one may find some more general filtering conditions than those given in Proposition 1 or manage to exploit any properties of the Sboxes beyond the fact that they are bijective.

We except that the tool proposed in this article, especially the idea of building and solving a difference propagation system, is not only helpful for evaluating the security of block ciphers against impossible differential cryptanalysis, but also useful in other attacks.

Acknowledgements. We are grateful to the anonymous reviewers for their valuable comments on this paper. This work was partially supported by the 973 Program (Grant No. 2013CB834203) and the National Natural Science Foundation of China (Grant No. 60970134 and Grant No. 11171323).

References

1. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita,T.: Camellia: a 128-bit block cipher suitable for multiple platforms — design and analysis. In: Stinson, D.R., Tavares, S.E. (eds.) SAC 2000. LNCS, vol. 2012, pp. 39–56. Springer, Heidelberg (2001)
2. Bahrak, B., Aref, M.R.: Impossible differential attack on seven-round AES-128. IET Information Security, vol. 2, pp. 28–32 (2008)
3. Bay, A., Nakahara Jr, J., Vaudenay, S.: Cryptanalysis of Reduced-Round MIBS Block Cipher. In: Heng, S.-H., Wright, R.N., Goi, B.-M. (eds.) CANS 2010. LNCS, vol. 6467, pp. 1–19. Springer, Heidelberg (2010)
4. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
5. Biham, E., Biryukov, A., Shamir, A.: Miss in the middle attacks on IDEA, Khufu and Khafre. In: Knudsen, L. (ed.) FSE'99. LNCS, vol. 1636, pp. 124–138. Springer, Heidelberg (1999)
6. Biham, E., Keller, N.: Cryptanalysis of Reduced Variants of Rijndael. In: The Third AES Candidate Conference (2000)
7. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology, vol. 4 (1), pp. 3–72 (1991)

8. Bosma, W., Cannon, J., Playoust, C.: The MAGMA Algebra System I: The User Language. In Journal of Symbolic Computation, vol. 24 (3-4), pp. 235–265 (1997)
9. Daemen, J., Rijmen, V.: The Design of Rijndael: AES – The Advanced Encryption Standard. Springer-Verlag (2002)
10. Du, C., Chen, J.: Impossible Differential Cryptanalysis of ARIA Reduced to 7 rounds. In: Heng, S.-H., Wright, R.N., Goi, B.-M. (eds.) CANS 2010. LNCS, vol. 6467, pp. 20–30. Springer, Heidelberg (2010)
11. Izadi, M.I., Sadeghiyan, B., Sadeghian, S.S., Khanooki, H.A.: MIBS: a newlightweight Block Cipher. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 334–348. Springer, Heidelberg (2009)
12. Kanda, M., Moriai, S., Aoki, K., Ueda, H., Takashima, Y., Ohta, K., Matsumoto, T.: E2 — A new 128-bit block cipher. In: IEICE Transactions Fundamentals – SpecialSection on Cryptography and Information Security, vol. E83 – A (1), pp. 48–59 (2000)
13. Kim, J., Hong, S., Sung, J., Lee, C., Lee, S.: Impossible differential cryptanalysis for block cipher structures. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 82–96. Springer, Heidelberg (2003)
14. Knudsen, L.R.: DEAL — A 128-bit block cipher. Technical Report 151, Department of Informatrics, University of Bergen, Bergen, Norway (1998)
15. Kwon, D., Kim, J., Park, S., et al.: New Block Cipher: ARIA. In: Lim, J.-I., Lee, D.-H. (eds.) ICISC 2003. LNCS, vol. 2971, pp. 432–445. Springer, Heidelberg (2004)
16. Li, R., Sun, B., Li, C.: Impossible differential cryptanalysis of SPN ciphers. In: IET Information Security, vol. 5 (2), pp. 111–120 (2011)
17. Li, S., Song, C.: Improved Impossible Differential Cryptanalysis of ARIA. In: ISA 2008, pp. 129–132. IEEE Computer Society, Los Alamitos (April 2008)
18. Luo, Y., Wu, Z., Lai, X., and Gong, G.: A Unified Method for Finding Impossible Differentials of Block Cipher Structures. Cryptology ePrint Archive: Report 2009/627. Available at: http://eprint.iacr.org/2009/627.
19. Mala, H., Dakhilalian, M., Rijmen, V., Modarres, M.-H.,: Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In: Gong, G., Gupta, K.C. (eds.): INDOCRYPT 2010. LNCS 6498, pp. 282–291, Springer, Heidelberg (2010)
20. Phan, R.C.-W., Siddiqi, M.U.: Generalised Impossible Differentials of Advanced Encryption Standard. Electronics Letters, vol. 37(14), pp. 896 – 898 (2001)
21. Phan, R.C.-W.: Classes of Impossible Differentials of Advanced Encryption Standard. Electronics Letters, vol. 38(11), pp. 508–510 (2002)
22. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 342–357. Springer, Heidelberg (2011)
23. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher CLEFIA (Extended abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007)
24. Tsunoo, Y., Tsujihara, E., Shigeri, M., Suzaki, T., Kawabata, T.: Cryptanalysis of CLEFIA Using Multiple Impossible Differentials. In: International Symposium on Information Theory and its Applications, ISITA 2008, pp. 1–6 (2008)
25. Wei, Y., Li, P., Sun, B., Li, C.: Impossible Differential Cryptanalysis on Feistel Ciphers with SP and SPS Round Functions. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 105–122. Springer, Heidelberg (2010)
26. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)
27. Wu, W., Zhang, W., Deng, D.: Impossible Diffrential Cryptanalysis of ARIA and Camellia. In: Journal of Computer Science and Technology, vol. 22(3), pp. 449–456 (2007)

A All Impossible Differentials of 4-Round AES

In this section, we describe all 4-round truncated impossible differentials of AES obtained by our tool. $\Delta X_i = (\Delta X_{i,j})_{1 \le j \le 16}$ (respectively, ΔY_i) can be represented by a 4×4 matrix over $\mathbb{F}_{2^8}$. In such a matrix, every byte in row m and column n is numbered as byte $m + 4 \cdot (n-1)$, that is, byte one is in the top-left corner, the first column is made of bytes 1-4, while the last column is made of bytes 13-16, with byte 16 in the bottom-right corner.

For given nonzero plaintext difference ΔX_1 and ciphertext difference ΔY_4, we denote by $I = \{j : \Delta X_{1,j} \neq 0 \text{ for } 1 \le j \le 16\}$ and $J = \{j : \Delta Y_{4,j} \neq 0 \text{ for } 1 \le j \le 16\}$. Suppose $a_1 = \{1, 6, 11, 16\}$, $a_2 = \{2, 7, 12, 13\}$, $a_3 = \{3, 8, 9, 14\}$, $a_4 = \{4, 5, 10, 15\}$ and $b_i = \{4 \cdot i - 3, 4 \cdot i - 2, 4 \cdot i - 1, 4 \cdot i\}$ for

$1 \leq i \leq 4$. We denote by $H(I)$ the number of true events that $I \cap a_i \neq \emptyset$ for $1 \leq i \leq 4$. For example, $H(I) = 2$ if $I = \{1,2\}$, because $I \cap a_1 \neq \emptyset$ and $I \cap a_2 \neq \emptyset$ but $I \cap a_3 = \emptyset$ and $I \cap a_4 = \emptyset$. Similarly, we denote by $H(J)$ the number of true events that $I \cap b_i \neq \emptyset$ for $1 \leq i \leq 4$. Then, we can search all 4-round truncated impossible differentials as follows.

1. We set truncated values for only a part of elements in ΔX_1 and ΔY_4 while letting other elements undetermined. Notice that ΔX_1 and ΔY_4 are nonzero vectors.
2. Use Algorithm 2 to predict the information of undetermined elements in ΔX_1 and ΔY_4.
3. $\Delta X_1 \rightarrow \Delta Y_4$ is impossible if we detect that ΔX_1 or ΔY_4 is a zero vector (see Example 2).

For example, in step 1, we initialize $\Delta X_1 = (?0000?0000?0000?) \in \mathbb{F}_2^{16} \setminus \{0\}$ and $\Delta Y_4 = (????000000000000) \in \mathbb{F}_2^{16} \setminus \{0\}$, that is, only 24 elements of ΔX_1 and ΔY_4 are fixed to zero while other elements are undetermined. Then, using Algorithm 2, we find ΔX_1 and ΔY_4 are zero vectors. Thus, we conclude that $\Delta X_1 \rightarrow \Delta Y_4$ is impossible, which implies that we obtain $15 \times 15 = 225$ truncated impossible differentials simultaneously. Other sets of impossible differentials can be obtained by changing the choice in step 1. Finally, we have

Proposition 3. *All 3,608,100 4-round truncated impossible differentials of AES we found are summarized as $H(I) + H(J) \leq 4$.*

Since $H(I) = 1$ provides $\binom{4}{1} \times (2^4 - 1) = 60$ choices, $H(I) = 2$ provides $\binom{4}{2} \times (2^4 - 1)^2 = 1350$ choices, and $H(I) = 3$ provides $\binom{4}{3} \times (2^4 - 1)^3 = 13500$ choices, the number of $H(I) + H(J) \leq 4$ is

$$60 \times (60 + 1350 + 13500) + 1350 \times (60 + 1350) + 13500 \times 60 = 3,608,100\ .$$

The impossible differentials discussed in [6, 20, 21, 2, 19] are subsets of our generalized results shown in Proposition 3. They include 269,554 impossible differentials in total.

What should be stressed here is that the method used in this section is highly relevant to the structure of AES. Thus, it is difficult to extend it to other block ciphers mentioned in Sect. 4.5.

B Source Code for Searching Truncated Impossible Differentials of LBlock

In this section, we list our source code for searching truncated impossible differentials of some typical block ciphers. First, the source code of Algorithm 2 is given. It costs about 100 lines in MAGMA package. Then, we list the code for searching truncated impossible differentials of ARIA and LBlock, which are specific examples of SPN ciphers and Feistel ciphers with SPN round functions, respectively.

B.1 Source Code of Algorithm 2

The specification of our source code for Algorithm 2 is given as follows.

```
//=============Sub-algorithms==========
//Algorithm 2 in this paper.
//A is the coefficient matrix of the linear system L.
//B:=[A|b] is the augmented matrix of the linear system L;

Predict_info:=function(B,T,Phi0,Phi1);
  m:=NumberOfRows(B);
  n:=NumberOfColumns(B);
  n0:=#Phi0;
  n1:=#Phi1;
  temp:=false;

//Predict information from the reduced row echelon form of B using Lemma 5.
  for i:=1 to m do;
    U:={};
    for j:=1 to n-1 do;
      if B[i,j] ne 0 then;
        U:=U join {j};
      end if;
    end for;

    //A linear equation with form X=0.
```

```
      if #U eq 1 and B[i,n] eq 0 then;
        Phi0:=Phi0 join U;
        for d in U do;
          MultiplyColumn(~B,0,d);
        end for;
     //A linear equation with form X+c=0 or X+Y=0
      elif #U eq 1 and B[i,n] ne 0 then;
        Phi1:=Phi1 join U;
      elif #U eq 2 and B[i,n] eq 0 and U meet Phi1 ne {} then;
        Phi1:=Phi1 join U;
      end if;
  end for;

 //Scan Table T and use Lemma 6 to predict information.
  numc:=NumberOfColumns(T);
  for k:=1 to numc do;
    if T[1,k] in Phi0 then;
        Phi0:=Phi0 join {T[2,k]};
        MultiplyColumn(~B,0,T[2,k]);
     elif T[1,k] in Phi1 then;
        Phi1:=Phi1 join {T[2,k]};
    end if;
  end for;

  if #Phi0 gt n0 or #Phi1 gt n1 then;
      temp:=true;
  end if;

 return <B,Phi0,Phi1,temp>;
end function;


Judge_IDs:=function(B,T,p,c,SdeltaP,SdeltaC);

  Phi0:={IntegerRing()|};
  Phi1:={IntegerRing()|};
  n:=NumberOfColumns(B);

//Initialize the truncated information of deltaP and deltaC using SdeltaP and SdeltaC.
    for i:=1 to NumberOfColumns(SdeltaP) do;
       if SdeltaP[1,i] eq 0 then;
          Phi0:=Phi0 join {p[i]};
          MultiplyColumn(~B,0,p[i]);
       else
          Phi1:=Phi1 join {p[i]};
       end if;

       if SdeltaC[1,i] eq 0 then;
          Phi0:=Phi0 join {c[i]};
          MultiplyColumn(~B,0,c[i]);
       else
          Phi1:=Phi1 join {c[i]};
       end if;
    end for;

//Predict information and detect contradictions.
  flag:=false;
  index:=true;
  while index do;
    A:=ColumnSubmatrixRange(B,1,n-1);
    b:=ColumnSubmatrixRange(B,n,n);
    if Rank(A) lt Rank(B) then;
       flag:=true; index:=false;
    else
       //Gauss-Jordan elimination;
       B:=EchelonForm(B);

       //Result:=<B,Phi0,Phi1,T2,temp>;
       Result:=Predict_info(B,T,Phi0,Phi1);
       B:=Result[1];
       Phi0:=Result[2];
       Phi1:=Result[3];

       if Phi0 meet Phi1 ne {} then;
         flag:=true;
         index:=false;
       else
         index:=Result[4];
       end if;
    end if;
  end while;
  return flag;
end function;
```

B.2 Source Code for ARIA

For SPN ciphers, we fix the order of all variables in (1) as

$$[\Delta X_1, \Delta Y_1, \Delta X_2, \Delta Y_2, \Delta X_3, \dots, \Delta Y_{r-1}, \Delta X_r, \Delta Y_r], \tag{14}$$

then, the coefficient matrix A of system $\mathcal{L}$ is

$$A = \begin{pmatrix} 0 & P & I & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & P & I & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & P & I & 0 \end{pmatrix}$$

and table $\mathcal{T}$ for formal equations in (1) is

$$\mathcal{T} = [\{2l(i-1)+j, 2l(i-1)+l+j\} : 1 \le i \le r, 1 \le j \le l]\ .$$

Now, we list our source code for finding truncated impossible differentials of SPN ciphers. ARIA is chosen as an example. For other SPN ciphers, what we need to specify in this code is the number of words l in the plaintext, the number of rounds r we consider, the finite field involved in the linear permutation layer and the coefficient matrices P of (1). Of course, the set of SdeltaP and SdeltaC may be changed according to other purposes. For example, if we are interested in SdeltaP and SdeltaC with exactly one nonzero word, then p1 and p2 in the following code will be chosen from set $\{2^i | 0 \le i \le l-1\}$.

```

//======Next is the Main Algorithm of ARIA======

l:=16;
//Define the coefficient matrices I and P.
//Note: different ciphers may have different matrices.
F:=GF(2);

//Build an identity matrix with l rows.
I:=ScalarMatrix(F, l, 1);

//Matrix P of ARIA.
P_ARIA:=Matrix(F,l,l,
[[0,0,0,1,1,0,1,0,1,1,0,0,0,1,1,0],
  [0,0,1,0,0,1,0,1,1,1,0,0,1,0,0,1],
  [0,1,0,0,1,0,1,0,0,0,1,1,1,0,0,1],
  [1,0,0,0,0,1,0,1,0,0,1,1,0,1,1,0],
  [1,0,1,0,0,1,0,0,1,0,0,1,0,0,1,1],
  [0,1,0,1,1,0,0,0,0,1,1,0,0,0,1,1],
  [1,0,1,0,0,0,0,1,0,1,1,0,1,1,0,0],
  [0,1,0,1,0,0,1,0,1,0,0,1,1,1,0,0],
  [1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1],
  [1,1,0,0,0,1,1,0,0,0,0,1,1,0,1,0],
  [0,0,1,1,0,1,1,0,1,0,0,0,0,1,0,1],
  [0,0,1,1,1,0,0,1,0,1,0,0,1,0,1,0],
  [0,1,1,0,0,0,1,1,0,1,0,1,1,0,0,0],
  [1,0,0,1,0,0,1,1,1,0,1,0,0,1,0,0],
  [1,0,0,1,1,1,0,0,0,1,0,1,0,0,1,0],
  [0,1,1,0,1,1,0,0,1,0,1,0,0,0,0,1]
]);


//To find IDs with different lengths, we can set the round number r to other integers.
r:=4;
//Note: Variable order we choose is [X_{1},Y_{1},X_{2},Y_{2},X_{3},\dots,Y_{r-1},X_{r},Y_{r},1],

//Generate the augmented matrix B, it contains l*(r-1) linear equations...
//...I \Delta X_{i+1}\oplus P \Delta Y_{i}=0;

 B:=ZeroMatrix(F,l*(r-1),l*(2*r)+1);
 for i:=1 to r-1 do;
```

```
   InsertBlock(~B,I,l*(i-1)+1,l*(2*i)+1);
   InsertBlock(~B,P_ARIA,l*(i-1)+1,l*(2*i-1)+1);
 end for;


// Construct table T;
// An element {v_1,v_2} in T is represented by two columns: [v_1,v_2] and [v_2,v_1];

  T:=ZeroMatrix(IntegerRing(),2,2*l*r);
  for i:=1 to r do;
    for j:=1 to l do;
      T[1,2*(l*(i-1)+j)-1]:=2*l*(i-1)+j;
      T[2,2*(l*(i-1)+j)-1]:=2*l*(i-1)+l+j;

      T[1,2*(l*(i-1)+j)]:=2*l*(i-1)+l+j;
      T[2,2*(l*(i-1)+j)]:=2*l*(i-1)+j;
    end for;
  end for;


//vector p and vector c;
p:=[IntegerRing()|1..l];
c:=[IntegerRing()|l*(2*r-1)+1..l*2*r];


for p1:=1 to 2^l-1 do;
 //SdeltaP is the sign vector of the plaintext difference: X_1;
  SdeltaP:=ZeroMatrix(F,1,l);
  L1:=IntegerToSequence(p1,2);
  for i:=1 to #L1 do;
    SdeltaP[1,i]:=L1[i];
  end for;

  for p2:=1 to 2^l-1 do;
     //SdeltaC is the sign vector of the ciphertext difference: Y_{r};
      SdeltaC:=ZeroMatrix(F,1,l);
      L2:=IntegerToSequence(p2,2);
      for i:=1 to #L2 do;
       SdeltaC[1,i]:=L2[i];
      end for;

    //Now, all inputs of the Algorithm 2 in this paper are ready.
    //Use algorithm 2 to judge whether given SdeltaP and SdeltaC is impossible;
    //Output the final results. They will be stored in the file "ARIA results.txt".
    TempB:=B;
    if Judge_IDs(TempB,T,p,c,SdeltaP,SdeltaC) then;
      Write("ARIA_results.txt",<r,SdeltaP,SdeltaC>);
    end if;

  end for;
end for;
```

B.3 Source Code for LBlock

In this section, we list our source code for finding truncated impossible differentials of Feistel ciphers with SPN round functions. Truncated impossible differentials we consider have the form $(0, \Delta X_0) \not\rightarrow (\Delta X_{r+1}, 0)$. What we need to specify in this code is the number of words l in the plaintext, the number of rounds r we consider, the finite field involved in the linear permutation layer and the coefficient matrices I_1, I_2 and P of (4).

We choose LBlock as an example to illustrate our source code. For MIBS, matrices I_1, I_2 and P are mentioned in Example 4.

```

//=======Next is the Main Algorithm of LBlock=======

l:=16;
//Define the coefficient matrices I_1, I_2 and P.
//Note: different ciphers may have different matrices.

F:=GF(2);
t:=Ceiling(l/2);

I_1:=Matrix(F,t,t,
[[0,0,1,0,0,0,0,0],
 [0,0,0,1,0,0,0,0],
 [0,0,0,0,1,0,0,0],
 [0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,1,0],
 [0,0,0,0,0,0,0,1],
 [1,0,0,0,0,0,0,0],
 [0,1,0,0,0,0,0,0]
]);

I_2:=ScalarMatrix(F, t, 1);

P_LBlock:=Matrix(F,t,t,
[[0,1,0,0,0,0,0,0],
 [0,0,0,1,0,0,0,0],
 [1,0,0,0,0,0,0,0],
 [0,0,1,0,0,0,0,0],
 [0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1],
 [0,0,0,0,1,0,0,0],
 [0,0,0,0,0,0,1,0]
]);

//To find IDs with different lengths, we can set the round number r to other integers.
r:=14;

//Note: Variable order we choose is [X_0,X_1,X_2,...,X_r,X_{r+1},Y_1,Y_2,..Y_r,1].

//Generate the augmented matrix B, it contains t*r linear equations...
//...I_1 \Delta X_{i-1} \oplus  I_2 \Delta X_{i+1} \oplus P \Delta Y_{i}=0.;
 B:=ZeroMatrix(F,t*r,t*(2*r+2)+1);
 for i:=1 to r do;
   InsertBlock(~B,I_1,t*(i-1)+1,t*(i-1)+1);
   InsertBlock(~B,I_2,t*(i-1)+1,t*(i+1)+1);
   InsertBlock(~B,P_LBlock,t*(i-1)+1,t*(r+2)+t*(i-1)+1);
 end for;

// Construct table T;
// An element {v_1,v_2} in T is represented by two columns: [v_1,v_2] and [v_2,v_1];
  T:=ZeroMatrix(IntegerRing(),2,l*r);
  for i:=1 to r do;
    for j:=1 to t do;
      T[1,2*(t*(i-1)+j)-1]:=t+t*(i-1)+j;
      T[2,2*(t*(i-1)+j)-1]:=t*(r+2)+t*(i-1)+j;
      T[1,2*(t*(i-1)+j)]:=t*(r+2)+t*(i-1)+j;
      T[2,2*(t*(i-1)+j)]:=t+t*(i-1)+j;
    end for;
  end for;

//vector p and vector c;
p:=[IntegerRing()|t+1..l] cat [IntegerRing()|1..t];
c:=[IntegerRing()|(t*r+t+1)..(t*r+l)] cat [IntegerRing()|(t*r+1)..(t*r+t)];


//Search truncated impossible differentials with form (0,X0)->(X_{r+1},0);
for p1:=1 to 2^(t)-1 do;
 //SdeltaP is the sign vector of the plaintext difference: (0,X_0);
  SdeltaP:=ZeroMatrix(F,1,l);
  L1:=IntegerToSequence(p1,2);
  for i:=1 to #L1 do;
    SdeltaP[1,t+i]:=L1[i];
  end for;

  for p2:=1 to 2^(t)-1 do;
     //SdeltaC is the sign vector of the ciphertext difference: (X_{r+1},0);
      SdeltaC:=ZeroMatrix(F,1,l);
      L2:=IntegerToSequence(p2,2);

      for i:=1 to #L2 do;
       SdeltaC[1,i]:=L2[i];
      end for;

    //Now, all inputs of the Algorithm 2 in this paper are ready.
    //Use algorithm 1 to judge whether given SdeltaP and SdeltaC is impossible;
    //Output the final results. They will be stored in the file ``LBlock results.txt''.
    TempB:=B;
    if Judge_IDs(TempB,T,p,c,SdeltaP,SdeltaC) then;
      Write("LBlock_results.txt",<r,SdeltaP,SdeltaC>);
    end if;

  end for;
end for;
```