Do NOT Misuse the Markov Cipher Assumption

Automatic Search for Differential and Impossible Differential Characteristics in ARX Ciphers

Zheng Xu[1,2], Yongqiang Li[1,2]✉, Lin Jiao[3], Mingsheng Wang[1,2], and Willi Meier[4]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
yongq.lee@gmail.com, {xuzheng,wangmingsheng}@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

[3] State Key Laboratory of Cryptology, Beijing, China
jiaolin_jl@126.com

[4] University of Applied Sciences and Arts Northwestern Switzerland (FHNW), Windisch, Switzerland
willimeier48@gmail.com

Abstract. Firstly, we improve the evaluation theory of differential propagation for modular additions and XORs, respectively. By introducing the concept of *additive sums* and using signed differences, we can add more information of value propagation to XOR differential propagation to calculate the probabilities of differential characteristics more precisely. Based on our theory, we propose the first modeling method to describe the general ARX differential propagation, which is not based on the Markov cipher assumption. Secondly, we propose an automatic search tool for differential characteristics with more precise probabilities in ARX ciphers. We find that some differential characteristics that used to be valid become impossible, and some probabilities that used to be underestimated increase. In applications, for CHAM-64/128 (one of the underlying block ciphers in COMET, one of 32 second-round candidates in NIST's lightweight cryptography standardization process), we find that there is no valid 39-round differential characteristic with a probability of 2^{-63} computed using previous methods, and we correct the probabilities to 2^{-64} and 2^{-64} instead of 2^{-65} and 2^{-65} computed using previous methods for two 39-round differential characteristics starting from the 1-st round, respectively; however, if we search for differential characteristics starting from the 5-th round, the two differential characteristics are invalid, which means that the round constants can affect the security of ARX ciphers against differential cryptanalysis; for Alzette with $c =$ `0xb7e15162` (one of the S-boxes in SPARKLE, one of 10 finalists in NIST's lightweight cryptography standardization process), we correct the probabilities to 0 and 2^{-22} instead of 2^{-23} and 2^{-23} computed using previous methods for two 4-round differential characteristics, respectively; for XTEA, we correct the probabilities to 0 and 2^{-49} instead of 2^{-58} and 2^{-56} computed using previous methods for two 10-round differential characteristics, respectively. Moreover, for Alzette with $c =$ `0xb7e15162`, XTEA, the `quarterround` function of Salsa20, and the round function of Chaskey, we find some invalid DCs that Leurent's ARX Toolkit cannot detect. Thirdly, we propose a SAT-based automatic search tool for impossible differential characteristics in ARX ciphers. We find some distinguishers ignored by previous methods. In applications, for CHAM-64/128, we find five 20-round and nineteen 19-round impossible differential characteristics starting from the 3-rd round for the first time. However, if we search for impossible differential characteristics starting from the 1-st round, we cannot find any 20-round impossible differential characteristic, which means that the round constants can affect the security of ARX ciphers against impossible differential cryptanalysis. Moreover, we find more impossible differential characteristics for 18-round, 16-round, 14-round, and 12-round CHAM-64/128, respectively. According to our results, the differential (resp. impossible differential) attack constructed by the previous methods of placing a DC (resp. an ID) anywhere in a block cipher may be invalid.

Keywords: Differential cryptanalysis · Differential probability · Impossible differential · ARX · SAT solver.

1 Introduction

The three components: modular addition, rotation, and XOR, constitute the basic operations in ARX cryptographic primitives [10]. By using these three operations, good diffusion and confusion can be achieved, as well as a cheap and fast implementation in both hardware and software. There are many noteworthy ARX algorithms, such as block ciphers SPECK [6], TEA [35], XTEA [28], and CHAM [18], MAC algorithm Chaskey [25], stream cipher Salsa20 [9], and hash functions MD4 [29], MD5 [30], SHA-1 [1], and BLAKE [4].

The security of ARX ciphers is evaluated by analyzing their robustness against various attacks. Some of the most successful attacks applied to ARX algorithms are differential cryptanalysis and impossible differential cryptanalysis. Since ARX ciphers use modular addition as a source of non-linearity, these attacks essentially exploit the non-randomness of differential propagation of modular addition.

Differential properties of modular additions have been studied for several decades. In 2001, Lipmaa and Moriai [22] proposed a fundamental method to determine whether a differential over the n-bit modular addition with two variable inputs is invalid and compute the differential probability with complexity $O(\log_2 n)$. Their method is widely used to evaluate the security of ARX ciphers against differential cryptanalysis. In 2010, Mouha et al. [27] introduced the concept of S-functions and used it to evaluate the probability of the modular addition with an arbitrary number of inputs. It is impressive to study the differential properties of modular additions by using S-functions. At ASIACRYPT'20, Azimi et al. [5] present the first bit-vector differential model for the n-bit modular addition by a constant input. The differential model in [5] is an elegant model that can determine whether a differential over the constant addition has non-zero probability and compute the binary logarithm of the differential probability.

In order to evaluate the security of block ciphers against differential cryptanalysis, Lai et al. [19] introduced Markov ciphers in 1991. Following the Markov cipher assumption, the probability of a differential characteristic (DC) can be computed by multiplying the probability of differential propagation of each round. Then, the Markov cipher assumption is used in practically all differential attacks and impossible differential attacks on block ciphers.

In the past 20 years, using automatic tools to search for DCs and impossible differentials (IDs) has become a new trend. The automatic tools are mainly of three types at present: Matsui's algorithm [24] by using the branch and bound search algorithm [11,12,16,23], mixed integer linear programming (MILP) models by converting the cryptographic properties into inequalities characterization problems [13,36], and using SAT/SMT solvers by characterizing the properties of components in ARX ciphers as a set of satisfiability problems [3,15,17,26,31].

All of these methods of searching for DCs and IDs are based on the Markov cipher assumption. Under the Markov cipher assumption, the probability of a DC in an ARX cipher is the product of the differential probability of each modular addition in each round because modular addition is the only nonlinear operation for an ARX cipher. However, some common ARX-based components do not follow the Markov cipher assumption since the operations contained may not keep independent and uniformly random without key injection, such as two consecutive modular additions, two parallel modular additions, and the XOR of a branch and a round constant. Therefore, searching for DCs and IDs under the Markov cipher assumption may lead to incorrect probabilities of DCs computed by simple segmented differential probability multiplications, ignoring some distinguishers, or deriving other improper cryptanalysis results. Thus, it is important to propose a method to better filter invalid DCs and obtain tight bounds for the probabilities of DCs.

To better filter invalid DCs and obtain tight bounds for the probabilities of DCs, cryptanalysts focus on the relations between differential bits and partially solve the above problem. In the groundbreaking works of Wang et al. [32–34] at EUROCRYPT'05 and CRYPTO'05, they used signed differences to find collisions in MD4, MD5, and full SHA-1. In 2012, Leurent [20] introduced the multi-bit constraints for consecutive bits of an XOR difference and proposed a notable automatic tool, ARX Toolkit, to search for valid DCs. However, using Leurent's ARX Toolkit, cryptanalysts can only get precise probabilities for the DCs with sparse differences since they cannot capture complete relations that involve a larger number of bits (i.e. inconsecutive bits). In 2013, Mouha et al. [26] used Lipmaa et al.'s conditions [22] and signed differences to capture complete relations between active bits. Furthermore, they presented an example that not all relations between bits can be captured by using Leurent's ARX Toolkit. The observation in [26] is rather subtle. However, Mouha et al. did not give a general calculation method for the probabilities of DCs. Besides, their method was a manual method, which means that the method can only be used to evaluate the security of ARX ciphers against differential cryptanalysis for a few rounds. Another shortcoming of only using Lipmaa et al.'s conditions and signed differences is that one cannot capture the relations between non-active bits, which leads to incorrect probability calculations and failure to filter invalid DCs.

Therefore, to better evaluate the security of ARX ciphers against differential cryptanalysis and impossible differential cryptanalysis, it is important to use Lipmaa et al.'s conditions, the complete relations between active bits, and the relations between non-active bits to build an automatic search model for DCs and IDs in ARX ciphers.

Our Contributions. In this paper, we revisit the differential properties of ARX ciphers. For ARX ciphers, we propose the first automatic method to search for DCs and IDs in ARX ciphers using Lipmaa et al.'s conditions, the complete relations between active bits, and the relations between non-active bits. The comparison of our method with previous methods is listed in Table 1. Our contributions are mainly three-fold.

Table 1. The comparison of our method with previous methods.

Method	Type	**active** bits	**non-active** bits	**consecutive** bits	**inconsecutive** bits
Wang et al.'s method and Mouha et al's method	manual	✓	✗	✓	✓
Leurent's method	automatic	✓	✓	✓	✗
Other automaic methods based on the Markov cipher assumption	automatic	✗	✗	✗	✗
Our method	automatic	✓	✓	✓	✓

More precise ARX differential propagation model. We improve the evaluation theory of differential propagation for modular additions and XORs, respectively. By introducing the concept of *additive sums* and using signed differences, we can add more information of value propagation to XOR differential propagation to calculate the probabilities of DCs more precisely. Based on our theory, we propose the first modeling method to describe the general ARX differential propagation using Lipmaa et al.'s conditions, signed differences, and additive sums. The method is not based on the Markov cipher assumption. For active bits and non-active bits, our method can handle the constraints between consecutive bits and the constraints between inconsecutive bits. Moreover, using our theory, we can evaluate the validity of a differential over a constant addition. Using our method, more precise ARX differential propagation can be modeled.

Automatic search algorithm for DCs with more precise probabilities in ARX ciphers. According to our new modeling method, we propose an automatic algorithm to search for DCs with more precise probabilities in ARX ciphers. We demonstrate the proposed algorithm on block ciphers CHAM-64/128 [18] and XTEA [28], a 64-bit ARX-box Alzette [7], the `quarterround` function of stream cipher Salsa20 [9], and the round function of MAC algorithm Chaskey [25]. Then, we find that some DCs that used to be valid become impossible, and some probabilities that used to be underestimated increase using this more precise tool. CHAM-64/128 is an ultra-lightweight block cipher that has remarkable efficiency on resource-constrained devices and very small hardware footprint [18]. COMET [14], one of 32 second-round candidates in NIST's lightweight cryptography standardization process [2], uses CHAM-64/128 as one of the underlying block ciphers.

For CHAM-64/128, we search for DCs starting from the 1-st round:

(i) We find that there is no valid 39-round DC with a probability of 2^{-63} computed following the Markov cipher assumption, which means that the 39-round optimal DC found by Roh et al. [31] is invalid.

(ii) We find eight valid 39-round DCs with a probability of 2^{-64} computed following the Markov cipher assumption (including the one found by Huang et al. [16]). Moreover, we confirm that the refined probabilities of the eight valid 39-round DCs are indeed 2^{-64}.

(iii) We find eighty-four valid 39-round DCs with a probability of 2^{-65} computed following the Markov cipher assumption. Then, we find that the refined probabilities of two of them are both 2^{-64} and the refined probabilities of the remaining eighty-two DCs are all 2^{-65}. However, if we search for DCs **starting from the 5-th round**, the two DCs with a refined probability of 2^{-64} are invalid, which means that the choice of the round constants can affect the security of ARX ciphers against differential cryptanalysis.

At CRYPTO'20, Beierle et al. presented a 64-bit ARX-box Alzette [7], which can be evaluated in constant time using only 12 instructions on modern CPUs. SPARKLE [8], one of 10 finalists in NIST's lightweight cryptography standardization process, provides the first application of the Alzette S-box.

For Alzette with $c =$ `0xb7e15162`,

(i) we correct the probabilities to 0 and 2^{-22} instead of 2^{-23} and 2^{-23} computed using previous methods for two 4-round DCs, respectively; then, we experimentally verify the probabilities;

(ii) we find a 4-round invalid DC that Leurent's ARX Toolkit cannot detect; then, we experimentally verify the probability.

XTEA is a lightweight block cipher proposed by Needham et al. [28]. Because XTEA is easy to implement, it is used in the encryption of some network protocols and databases (e.g. KCP Protocol[5] and H2 Database[6]).

For XTEA,

(i) we find the 18-round related-key DC found by Azimi et al. [5] is valid, and the probability is correct;

(ii) we correct the probabilities to 0 and 2^{-49} instead of 2^{-58} and 2^{-56} computed using previous methods for two 10-round DCs, respectively;

(iii) we find a 9-round invalid DC that Leurent's ARX Toolkit cannot detect.

[5] https://github.com/skywind3000/kcp

[6] http://www.h2database.com/html/main.html

Salsa20 [9] is a stream cipher designed by Bernstein in 2005 as a candidate for the eSTREAM competition. The 12-round variant of Salsa20, Salsa20/12, was accepted into the final eSTREAM software portfolio.

For the `quarterround` function of Salsa20, we find an invalid DC that Leurent's ARX Toolkit cannot detect.

Chaskey [25] is a lightweight MAC algorithm whose underlying primitive is an ARX-based permutation in an Even-Mansour construction. The 12-round variant of Chaskey, Chaskey-12, is standardized in ISO/IEC 29192-6.

For the round function of Chaskey, we find an invalid DC that Leurent's ARX Toolkit cannot detect.

SAT-based automatic search tool for IDs in ARX ciphers. Based on our new modeling method, we propose a SAT-based automatic search tool for IDs in ARX ciphers. We demonstrate the proposed tool on block cipher CHAM-64/128. Then, we find some new IDs ignored by previous methods.

For CHAM-64/128, we search for IDs starting from the 3-rd round:

(i) We find five 20-round and nineteen 19-round IDs for the first time. Among the 24 IDs, only one 19-round ID can be found by using previous methods. However, if we search for IDs **starting from the 1-st round**, we cannot find any 20-round ID, which means that the choice of the round constants can affect the security of ARX ciphers against impossible differential cryptanalysis.

(ii) We find more IDs (47, 704, 2537, and 3836, respectively) for 18 rounds, 16 rounds, 14 rounds, and 12 rounds, respectively, which cover the results obtained by using previous methods.

In the past view, one can place a DC (resp. an ID) **anywhere** in a block cipher to construct a differential attack (resp. an impossible differential attack). However, according to our results, the differential attack (resp. impossible differential attack) may be **invalid**. Therefore, our methods are helpful for better evaluating the security of ARX ciphers against differential and impossible differential cryptanalysis. The comparison of our results with those given by previous methods is listed in Table 2.

Outline. The paper is organized as follows. Some preliminaries are given in Section 2. In Section 3, we propose a more precise evaluation theory of differential propagation for ARX operations and a new modeling method accordingly. Then, we propose a general automatic algorithm to search for DCs with more precise probabilities in ARX ciphers and apply it to CHAM-64/128, Alzette, XTEA, the `quarterround` of Salsa20, and the round function of Chaskey in Section 4. Besides, we propose a new SAT-based automatic search tool for IDs in ARX ciphers and apply it to CHAM-64/128 in Section 5. In Section 6, we conclude this paper.

Table 2. The comparison of our results with those given by previous methods.

Type	**Cipher**	**Round**	**OP**[1] **(Ref.)**	**RP**[2] **(Ref.)**
DC	CHAM-64/128	39	2^{-63} ([31])	0 (this paper)
			2^{-64} ([16])	2^{-64} (this paper)
			2^{-65} (this paper)	2^{-64} (this paper)
	Alzette	4	2^{-23} (this paper)	0 (this paper)
			2^{-23} (this paper)	2^{-22} (this paper)
			2^{-38} (this paper)	0 (this paper)
	XTEA	18	2^{-57} ([5])	2^{-57} (this paper)
		10	2^{-58} (this paper)	0 (this paper)
			2^{-56} (this paper)	2^{-49} (this paper)
		9	2^{-60} (this paper)	0 (this paper)
	the `quarterround` of Salsa20	1	2^{-13} (this paper)	0 (this paper)
	the round function of Chaskey	1	2^{-11} (this paper)	0 (this paper)
Type	**Cipher**	**Round**	$\boldsymbol{NID_{PM}}$[3] **(Ref.)**	$\boldsymbol{NID_{OM}}$[4] **(Ref.)**
ID	CHAM-64/128	20	0 (this paper)	5 (this paper)
		19	1 (this paper)	19 (this paper)
		18	1 [18]	47 (this paper)

[1] The original probability of a DC calculated using previous methods.
[2] The refined probability of a DC calculated using our method.
[3] The number of IDs found using previous methods.
[4] The number of IDs found using our method.

2 Preliminaries

Notations used in this paper are as follows:

- $x\|y$: concatenation of bit strings x and y.
- $x \wedge y$: bitwise AND of x and y.
- $x \oplus y$: bitwise exclusive OR of x and y.
- $x \boxplus y$: addition of x and y modulo 2^n.
- $\overline{x}$: bitwise NOT of x.
- $wt(x)$: the Hamming weight of x.
- $x[i]$: bit at position i of word x, where $i = 0$ is the least significant bit; or the i-th element in set x.
- $x \ll r$: shift of x to the left by r positions.
- $x \gg r$: shift of x to the right by r positions.
- $x \lll r$: rotation of x to the left by r positions.
- $x \ggg r$: rotation of x to the right by r positions.
- Δx: XOR difference of x and x': $\Delta x = x \oplus x'$.
- $\Delta^+ x$: additive difference of x and x': $\Delta^+ x = x - x' \mod 2^n$.
- $\Delta^\pm x$: signed difference of x and x': $\Delta^\pm x[i] = x[i] - x'[i] \in \{-1, 0, 1\}$.
- $\nabla^+ x$: additive sum of x and x': $\nabla^+ x[i] = x[i] + x'[i] \in \{0, 1, 2\}$.
- `InD`: input difference.
- `OutD`: output difference.
- DC: differential characteristic.
- ID: impossible differential.

Definition 1 (Addition modulo 2^n [22]). *Let $x, y \in \mathbb{F}_2^n$, then*

$$x \boxplus y = x \oplus y \oplus carry(x, y),$$

where $carry(x, y) = (c[n-1], \ldots, c[1], c[0]) \in \mathbb{F}_2^n$ is the carry bit vector of $x \boxplus y$, defined recursively as: $c[0] = 0$; $c[i+1] = (x[i] \wedge y[i]) \oplus (x[i] \wedge c[i]) \oplus (y[i] \wedge c[i])$, $0 \leq i \leq n-2$.

Definition 2 (XOR differential probability of modular addition [22]). *An XOR differential of addition modulo 2^n is defined as a triplet $(\alpha, \beta \mapsto \gamma)$, where $\alpha, \beta \in \mathbb{F}_2^n$ are the two input differences and $\gamma \in \mathbb{F}_2^n$ is the output difference. Then, the XOR differential probability of modular addition is defined as*

$$P(\alpha, \beta \mapsto \gamma) = \frac{\#\{(x, y) | (x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma\}}{\#\{(x, y)\}}.$$

In [22], Lipmaa and Moriai studied the XOR differential probability of modular addition and proved that an XOR differential triplet $(\alpha, \beta \mapsto \gamma)$ is valid if and only if

$$eq(\alpha \lll 1, \beta \lll 1, \gamma \lll 1) \wedge (\alpha \oplus \beta \oplus \gamma \oplus (\beta \lll 1)) = 0, \tag{1}$$

where

$$eq(x, y, z) = (\overline{x} \oplus y) \wedge (\overline{x} \oplus z). \tag{2}$$

Then, if $(\alpha, \beta \mapsto \gamma)$ is valid, the XOR differential probability of modular addition can be calculated as follows:

$$P(\alpha, \beta \mapsto \gamma) = 2^{-wt(\overline{eq(\alpha,\beta,\gamma)} \wedge mask(n-1))},$$

where $mask(n-1)$ denotes $0 \| 1^{n-1}$.

Definition 3 (Signed differences [32,34]). *The signed differences $\Delta^{\pm}x$ split up the XOR differences into three possible cases:*

(i) $\Delta^{\pm}x[i] = 0$, for $x[i] = x'[i]$;
(ii) $\Delta^{\pm}x[i] = +1$, for $x[i] = 1$ and $x'[i] = 0$;
(iii) $\Delta^{\pm}x[i] = -1$, for $x[i] = 0$ and $x'[i] = 1$.

A signed difference $\Delta^{\pm}x$ corresponds to exactly one XOR difference Δx and one additive difference $\Delta^{+}x$ as follows:

$$\Delta x = \bigoplus_{i=0}^{n-1} |\Delta^{\pm}x[i]| \cdot 2^i,$$

$$\Delta^{+}x = \sum_{i=0}^{n-1} \Delta^{\pm}x[i] \cdot 2^i \mod 2^n. \tag{3}$$

Then, according to Equation (3), we have the following corollary.

Corollary 1. *Let $x, x', y, y', z, z' \in \mathbb{F}_2^n$, $z = x \boxplus y$, and $z' = x' \boxplus y'$. For a valid differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$, $\Delta^{\pm}x$, $\Delta^{\pm}y$, and $\Delta^{\pm}z$ have the following relations:*

$$\sum_{i=0}^{n-1} \Delta^{\pm}x[i] \cdot 2^i \mod 2^n \boxplus \sum_{i=0}^{n-1} \Delta^{\pm}y[i] \cdot 2^i \mod 2^n = \sum_{i=0}^{n-1} \Delta^{\pm}z[i] \cdot 2^i \mod 2^n. \tag{4}$$

Definition 4 (Additive sums). *For $x, x' \in \mathbb{F}_2^n$, we introduce a new concept of additive sums $\nabla^+ x$ to denote the additive sum of x and x': $\nabla^+ x[i] = x[i] + x'[i]$. The additive sums $\nabla^+ x$ split up the XOR differences into three possible cases:*

(i) $\nabla^+ x[i] = 0$, *for* $x[i] = x'[i] = 0$;
(ii) $\nabla^+ x[i] = 1$, *for* $x[i] \neq x'[i]$;
(iii) $\nabla^+ x[i] = 2$, *for* $x[i] = x'[i] = 1$.

Then, we have

$$\nabla^+ x = \sum_{i=0}^{n-1} \nabla^+ x[i] \cdot 2^i \mod 2^{n+1}. \tag{5}$$

Definition 5 (Markov cipher [19]). *An iterated cipher with round function $Y = f(X, K)$ is a Markov cipher if there is a group operation $\otimes$ for defining differences such that, for all choices of α and β ($\alpha \neq e, \beta \neq e$),*

$$P(\mathit{\Delta} Y = \beta | \mathit{\Delta} X = \alpha, X = \gamma)$$

is independent of γ, where the subkey is uniformly random.

A cipher is always assumed to be a Markov cipher in its security evaluation against differential and impossible differential cryptanalysis. Under the Markov cipher assumption, the probability of a DC in an ARX cipher is the product of the differential probability of each modular addition in each round because modular addition is the only nonlinear operation for an ARX cipher [3, 11–13, 15, 16, 23, 26, 31]. Let N_i and $(\alpha^{i,j}, \beta^{i,j} \mapsto \gamma^{i,j})$ denote the number of modular additions in the i-th round and the XOR differential triplet of the j-th modular addition in the i-th round. For an r-round DC $\mathit{\Omega}$, the probability $P_{\mathit{\Omega}}$ of $\mathit{\Omega}$ is computed by

$$P_{\mathit{\Omega}} = \prod_{i=1}^{r} \prod_{j=1}^{N_i} P(\alpha^{i,j}, \beta^{i,j} \mapsto \gamma^{i,j}).$$

3 More Precise Evaluation Theory of ARX Differential Propagation

In this section, we rebuild the evaluation framework of ARX differential propagation.

3.1 Improved Description Method of XOR Differential Propagation for ARX Operations

We find that some common ARX-based components do not follow the Markov cipher assumption since the operations contained may not keep independent and uniformly random without key injection, which may lead to incorrect probabilities of DCs computed by simple segmented differential probability multiplications. Here, we present such cases in terms of modular additions and XORs, respectively.

Modular Additions. Two consecutive modular additions without any key injection and two parallel modular additions that share an identical input branch are widely used in ARX ciphers. However, these cases do not follow the Markov cipher assumption. We present some counter-examples, which are further explained later.

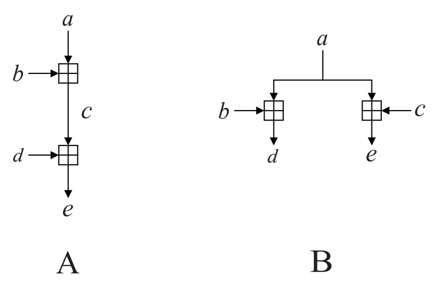

Fig. 1. Two consecutive modular additions and two parallel modular additions that share an identical input branch

Counter-example 1 *For two consecutive modular additions without any key injection shown in Fig. 1.A, we evaluate the probabilities of DCs for two situations using enumeration experiments and get the following results:*

(i) the probability is 0 *for* $\Delta a =$ `0x05`, $\Delta b =$ `0x01`, $\Delta c =$ `0x3c`, $\Delta d =$ `0x01`, *and* $\Delta e =$ `0x35`;
(ii) the probability is 2^{-7} *for* $\Delta a =$ `0x09`, $\Delta b =$ `0x03`, $\Delta c =$ `0x78`, $\Delta d =$ `0x01`, *and* $\Delta e =$ `0x09`.

However, the probabilities of these two situations are 2^{-10} *and* 2^{-11}*, respectively, if we evaluate them under the Markov cipher assumption.*

Counter-example 2 *For two parallel modular additions that share an identical input branch shown in Fig. 1.B, we evaluate the probabilities of DCs for two situations using enumeration experiments and get the following results:*

(i) the probability is 0 *for* $\Delta a =$ `0x78`, $\Delta b =$ `0x03`, $\Delta c =$ `0x01`, $\Delta d =$ `0x09`, *and* $\Delta e =$ `0x69`;
(ii) the probability is 2^{-6} *for* $\Delta a =$ `0x78`, $\Delta b =$ `0x01`, $\Delta c =$ `0x01`, $\Delta d =$ `0x09`, *and* $\Delta e =$ `0x09`.

However, the probabilities of these two situations are 2^{-11} *and* 2^{-10}*, respectively, if we evaluate them under the Markov cipher assumption.*

To explore the reasons for such contradictions and describe the XOR differential propagation more precisely, we use the concept of signed differences and the concept of *additive sums* to add more information of value propagation to XOR differential propagation. For a valid XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$, we can derive the relations between signed differential bits and the relations between additive sum bits of the modular addition using Corollary 1 and the following method, respectively.

Theorem 1. *Let $x, x', y, y', z, z' \in \mathbb{F}_2^n$, $z = x \boxplus y$, and $z' = x' \boxplus y'$. For a valid XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$, $\nabla^+ x$, $\nabla^+ y$, and $\nabla^+ z$ have the following relations:*

for $\boldsymbol{\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] \neq 0}$,

$$(\sum_{i=0}^{n-1} \nabla^+ x[i] \cdot 2^i + \sum_{i=0}^{n-1} \nabla^+ y[i] \cdot 2^i) \mod 2^n = \sum_{i=0}^{n-1} \nabla^+ z[i] \cdot 2^i \mod 2^n; \tag{6}$$

for $\boldsymbol{\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] = 0}$,

$$(\sum_{i=0}^{n-1} \nabla^+ x[i] \cdot 2^i + \sum_{i=0}^{n-1} \nabla^+ y[i] \cdot 2^i) \mod 2^{n+1} = \sum_{i=0}^{n-1} \nabla^+ z[i] \cdot 2^i \mod 2^{n+1}. \tag{7}$$

Remark 1. At ASIACRYPT'20, Azimi et al. [5] presented a bit-vector differential model for the modular addition by a constant. For $x \boxplus c = z$ and $x' \boxplus c = z'$, Azimi et al. proposed a method to evaluate the validity of $(\Delta x \mapsto \Delta z)$ for fixed $\Delta x = x \oplus x'$ and $\Delta z = z \oplus z'$, where c is a constant. However, for $x \boxplus y = z$, $x' \boxplus y' = z'$, and $y = y'$ (y and y' can be constants or not), we can use Theorem 1 to get the relations between the bits of y and y' for fixed $\Delta x = x \oplus x'$ and $\Delta z = z \oplus z'$ when the XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$ is valid. Then, for a valid XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$ and a constant c, where $y = y'$ (y and y' can be constants or not), if the relations between the bits of c and the relations between the bits of y and y' are compatible, $(\Delta x \mapsto \Delta z)$ is valid under the constant c. Therefore, we emphasize that, according to Theorem 1, we can evaluate whether a differential over the constant addition is valid.

Next, we propose a method for precisely calculating the overall probability of a DC.

Theorem 2. *Let P_{r1} and P_{r2} denote the differential probabilities of the first modular addition and the second modular addition, respectively. Then the following statements hold.*

(1) Suppose there are contradictions in the relations between the bits of the two modular additions, then the overall probability P_r of the DC equals 0.

(2) Suppose there are no contradictions in the relations between the bits of the two modular additions. Let N denote the number of relations that have to be satisfied by the second modular addition and have been satisfied by the first modular addition. Then, the overall probability P_r of the DC can be calculated as

$$P_r = P_{r1} \times P_{r2} \times 2^N.$$

Remark 2. For completeness, the proofs of Theorem 1 and Theorem 2 are given in Supplementary Material.

Remark 3. We emphasize that, for the case of two consecutive modular additions without a branch injection in the middle, we treat them as two modular additions rather than a single modular addition with three inputs and one output. This approach can help us capture the relations between bits. Furthermore, we can find the repetitive relations and the incompatible relations between the bits of the two modular additions.

In the rest of this section, since the behavior of signed differences propagating through XORs is similar to the behavior of additive sums propagating through XORs, we only show the counter-example of signed differential propagation.

XORs. The XOR of a branch and a round constant is widely used in ARX ciphers. However, this case may make the distribution of output signed differences (resp. additive sums) not keep independent and uniformly random, and make subsequent operations not follow the Markov cipher assumption. We present a counter-example, which is further explained later.

Counter-example 3 *For $z = x \oplus c$ and $z' = x' \oplus c$, assume that $x =$ `0b01`, $x' =$ `0b10`, and $c =$ `0b01`, where c is the round constant. Therefore, we can easily know that $\Delta^{\pm} z = \texttt{0b}(-1)(-1)$. However, we will get $\Delta^{\pm} z = \texttt{0b}(\pm 1)(\pm 1)$ using only XOR differential propagation.*

To describe differential propagation more precisely, we use signed differences (resp. additive sums) to add more information of value propagation to XOR differential propagation.

Theorem 3. *Let $x, x', z, z', c \in \mathbb{F}_2^n$. For $z = x \oplus c$ and $z' = x' \oplus c$, where c is the round constant, we have $\Delta z = \Delta x$ and the following relations between signed differential (resp. additive sum) bits:*

(i) for $\Delta z[i] = 1$, $\Delta^{\pm} z[i] \neq 0$ (resp. $\nabla^{+} z[i] = 1$);
(ii) for $\Delta z[i] = 0$, $\Delta^{\pm} z[i] = 0$ (resp. $\nabla^{+} z[i] \neq 1$);
(iii) for $c[i] = 0$, $\Delta^{\pm} z[i] = \Delta^{\pm} x[i]$ (resp. $\nabla^{+} z[i] = \nabla^{+} x[i]$);
(iv) for $c[i] = 1$, $\Delta^{\pm} z[i] = -\Delta^{\pm} x[i]$ (resp. $\nabla^{+} z[i] + \nabla^{+} x[i] = 2$);

where $0 \leq i \leq n - 1$.

To simplify our analysis, we process the case of the XOR of two branches (none of the branches are key) following the whitening outputs assumption. That is, for a particular output XOR difference, the values of the output pairs are uniformly distributed [26]. This assumption is valid when we consider very sparse DCs [26].

Remark 4. For all these examples, we have verified them experimentally.

Remark 5. Using the above method, we can handle the case of a branch injection between two consecutive modular additions. In contrast, Mouha et al. [27] treated the case of two consecutive modular additions without a branch injection in the middle as one modular addition with three inputs and one output. Then, they applied it to XTEA and found that the probability of the XOR differential quartet $(\alpha, 0, \alpha \mapsto 0)$ with $\alpha = \texttt{0x80402010}$ over the two consecutive modular additions in one round of XTEA is 2^{-3} instead of 2^{-6} computed by considering the two modular additions to be independent of each other. Although their method can improve the probability because they actually calculate the differential probability instead of the probability of a DC, their method is not suitable for the case of a branch injection between two consecutive modular additions, which means that the calculation result proposed in [27] is incorrect. The details are shown in Appendix A.

3.2 New Modeling Method for ARX Differential Propagation

Based on the theory in Section 3.1, we propose the first modeling method to describe the general ARX differential propagation using Lipmaa et al.'s conditions, signed differences, and additive sums. The method is not based on the Markov cipher assumption. For active bits and non-active bits, our new method can handle the constraints between consecutive bits and the constraints between inconsecutive bits.

Suppose that modular additions, rotations, and XORs all operate on n-bit words, where $n > 1$. Then, one can use the following steps to model more precise ARX differential propagation:

Step 1. For each addition modulo 2^n, $z = x \boxplus y$ and $z' = x' \boxplus y'$, we use Equation (1) and Equation (2) to ensure that $(\Delta x, \Delta y \mapsto \Delta z)$ is valid.

Step 2. For each addition modulo 2^n, $z = x \boxplus y$ and $z' = x' \boxplus y'$, we use Equation (4) to get the relations between the bits of $\Delta^{\pm} x$, $\Delta^{\pm} y$, and $\Delta^{\pm} z$.

Step 3. For each addition modulo 2^n, $z = x \boxplus y$ and $z' = x' \boxplus y'$, we use Equation (6) and Equation (7) to get the relations between the bits of $\nabla^{+} x$, $\nabla^{+} y$, and $\nabla^{+} z$.

Step 4. For each rotation of a word to the left, $y = x \lll t$ and $y' = x' \lll t$, we have $\Delta y = \Delta x \lll t$, $\Delta^{\pm} y = \Delta^{\pm} x \lll t$, and $\nabla^{+} y = \nabla^{+} x \lll t$. We use a similar method to process the cases of rotation of a word to the right and shift of a word.

Step 5. For the XOR of two branches, $z = x \oplus y$ and $z' = x' \oplus y'$, let $J \subseteq \{0, 1, \ldots, n-1\}$. Suppose that (x, x') and $(y[j'], y'[j'])$ are random, and $y[j] = y'[j] = C$, where $j' \notin J$, $0 \leq j' < n$, $j \in J$, and $C \in \{0, 1\}$. According to Theorem 3, we process XOR differential propagation as $\Delta z = \Delta x \oplus \Delta y$, and process signed differential propagation and additive sum propagation as follows:

(1) for $\Delta z[i] = 1$, $\Delta^{\pm} z[i] \neq 0$ and $\nabla^{+} z[i] = 1$;
(2) for $\Delta z[i] = 0$, $\Delta^{\pm} z[i] = 0$ and $\nabla^{+} z[i] \neq 1$;
(3) for $y[j] = y'[j] = 0$, $\Delta^{\pm} z[j] = \Delta^{\pm} x[j]$ and $\nabla^{+} z[j] = \nabla^{+} x[j]$;

(4) for $y[j] = y'[j] = 1$, $\Delta^{\pm}z[j] = -\Delta^{\pm}x[j]$ and $\nabla^{+}z[j] + \nabla^{+}x[j] = 2$;
where $0 \leq i < n$.

Step 6. For propagation between operations (resp. rounds), the output XOR difference, signed difference, and additive sum of the former operation (resp. round) have to equal the input XOR difference, signed difference, and additive sum of the later operation (resp. round), respectively.

4 Automatic Search Tool for Differential Characteristics

In this section, we propose an automatic algorithm to search for DCs with more precise probabilities according to the modeling method in Section 3.2 and further apply the algorithm to several ARX ciphers.

4.1 Automatic Search Algorithm for Differential Characteristics

We list our automatic algorithms with pseudo-codes in Appendix C. The automatic search algorithm for DCs consists of one main algorithm and three sub algorithms. Algorithm 2 is the main algorithm of searching for DCs with more precise probabilities and consists of two parts:

(i) automatically searching for DCs with a weight limit of w using any existing method;
(ii) automatically evaluating the probabilities of DCs.

Algorithm 2 is essentially a recursive algorithm. For automatically searching for DCs with a weight limit of w, the first part of Algorithm 2 calls Algorithm 1 to search for DCs with $Pr_{Original} = 2^{-w}$ using any existing method, where $Pr_{Original}$ denotes the original probabilities computed following the Markov cipher assumption. Since we add the corresponding constraints of Equation (4), Equation (6), and Equation (7) to Algorithm 1, all the DCs found by our algorithm are valid. For automatically evaluating probabilities, the second part of Algorithm 2 performs the following two steps in each recursive call:

(i) calling Algorithm 4 to derive the relations between the bits of each modular addition;
(ii) calling Algorithm 3 to derive the factor *count* that modifies the original probability in each round and derive the sets of relations between the output bits in each round.

When the first part of Algorithm 2 finds a valid DC, the second part of Algorithm 2 automatically derives the relations between the bits of each modular addition, and automatically derives the factor *count* that modifies the original probability for each round, and then refines the original probability of the DC.

This algorithm series, which is based on the theory we built in Section 3, exploits signed differential propagation and additive sum propagation to better evaluate the security of ARX ciphers against differential cryptanalysis. Using this new automatic tool, we find that some DCs that used to be valid become impossible, and some probabilities that used to be underestimated increase.

Then, we present a technique to improve the efficiency of our algorithm. For each modular addition, the value of $\#\{(\Delta^{\pm}x, \Delta^{\pm}y, \Delta^{\pm}z) | \Delta^{\pm}x \boxplus \Delta^{\pm}y = \Delta^{\pm}z \mod 2^n\}$ is important to derive the relations between bits. However, since modular addition usually operates on 16, 32, 48, or 64-bit words, it is not efficient enough if we use an exhaustive search to calculate the value of $\#\{(\Delta^{\pm}x, \Delta^{\pm}y, \Delta^{\pm}z) | \Delta^{\pm}x \boxplus \Delta^{\pm}y = \Delta^{\pm}z \mod 2^n\}$. Therefore, we introduce a new method for efficiently calculating the value of $\#\{(\Delta^{\pm}x, \Delta^{\pm}y, \Delta^{\pm}z) | \Delta^{\pm}x \boxplus \Delta^{\pm}y = \Delta^{\pm}z \mod 2^n\}$ as the following theorem.

Theorem 4. *Let $\Delta x, \Delta y, \Delta z \in \mathbb{F}_2^n$. For a modular addition, if an XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$ is valid, the probability can be calculated as*

$$P(\Delta x, \Delta y \mapsto \Delta z) = \frac{\#\{(\Delta^{\pm}x, \Delta^{\pm}y, \Delta^{\pm}z) | \Delta^{\pm}x \boxplus \Delta^{\pm}y = \Delta^{\pm}z \mod 2^n\}}{2^{wt(\Delta x)+wt(\Delta y)+wt(\Delta z)}}.$$

Remark 6. For completeness, the proof of Theorem 4 is given in Supplementary Material.

For a valid XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$, we can easily get the probability using Lipmaa et al.'s algorithm. Therefore, we can efficiently calculate the value of $\#\{(\Delta^{\pm}x, \Delta^{\pm}y, \Delta^{\pm}z) | \Delta^{\pm}x \boxplus \Delta^{\pm}y = \Delta^{\pm}z \mod 2^n\}$ using Theorem 4. Based on this, using Corollary 1, Theorem 1, and Theorem 2, our automatic tool can efficiently derive the relations between bits for each modular addition and efficiently evaluate the probability of a DC.

Finally, we give a rough estimation of the complexity of the automatic search algorithm. Without loss of generality, we assume that the number of modular additions is m and the word size of each modular addition is n. Let N_i denote the sum of the Hamming weight of the input and output XOR differences of the i-th modular addition, and N_{max} denote the maximum value of N_i, where $1 \leq i \leq m$. As the complexity of the second part of Algorithm 2 is dominated by the value of N_{max}, the complexity of this part has the form $O(2^{N_{max}})$. Thus, the complexity of the automatic search algorithm is dominated by the higher one between the complexity of the second part of Algorithm 2 (i.e. $O(2^{N_{max}})$) and the complexity of the existing search method applied by Algorithm 2. Note that although the second part of Algorithm 2 seems to have relatively high complexity, DCs used in a differential attack have reasonable probabilities, which means that N_{max} is relatively small. Therefore, the automatic search algorithm is actually efficient.

In the rest of this section, we use the method proposed by Mouha et al. [26] to search for DCs and further get more precise probabilities.

Remark 7. In 2012, Leurent [20] introduced the multi-bit constraints for consecutive bits of an XOR difference and proposed a notable automatic tool, ARX Toolkit, to search for valid DCs. Then, Leurent updated the ARX Toolkit in 2013 [21]. However, Leurent's ARX Toolkit cannot filter all invalid DCs because Leurent's ARX Toolkit cannot capture complete relations that involve a larger number of bits (i.e. inconsecutive bits). In contrast, since our automatic tool is based on Lipmaa et al.'s conditions, the complete relations between active bits, and the relations between non-active bits, our automatic tool can better filter invalid DCs and obtain tight bounds for the probabilities of DCs. In the rest of this section, we show some invalid DCs that cannot be detected by Leurent's ARX Toolkit but can be detected by our automatic tool.

4.2 Application to CHAM-64/128

CHAM is a family of lightweight block ciphers designed by Koo et al. [18]. We present the description of CHAM in Supplementary Material. In [16], Huang et al. found a 39-round DC with the original probability of 2^{-64} using an adapted Matsui's algorithm proposed by Liu et al. [23]. Huang et al. claimed that the 39-round DC is the 39-round optimal DC for CHAM-64/128. However, using a SAT-based method, Roh et al. [31] found another 39-round DC with the original probability of 2^{-63}. Although Roh et al. only proposed the input and output differences, we use the same method to find the DC, and we verify that the DC is the only one under the same setting. The two DCs are shown in Table 7 in Appendix B, and we use $\mathcal{DC}_1$ and $\mathcal{DC}_2$ to denote the DC found by Huang et al. and the DC found by Roh et al. We believe that maybe Huang et al.'s application is incorrect because the above two methods both only include Lipmaa et al.'s conditions to ensure the validity of the XOR differential triplets of modular additions and follow the Markov cipher assumption to compute the probabilities of DCs. Therefore, under the Markov cipher assumption, the DC found by Roh et al. is the 39-round optimal DC for CHAM-64/128.

For CHAM-64/128, we use our automatic tool to search for DCs starting from the 1-st round.

Setting $\boldsymbol{r = 39}$ and $\boldsymbol{w = 63}$, we cannot find any valid 39-round DC with the original probability of 2^{-63}. This is an interesting result, which means that the 39-round DC found by Roh et al. is invalid.

Setting $\boldsymbol{r = 39}$ and $\boldsymbol{w = 64}$, we can find all valid 39-round DCs with the original probability of 2^{-64}. Besides the one found by Huang et al. [16], we find seven valid 39-round DCs. Moreover, we confirm that the refined probabilities of the eight valid 39-round DCs are indeed 2^{-64}.

Setting $\boldsymbol{r = 39}$ and $\boldsymbol{w = 65}$, we can find all valid 39-round DCs with the original probability of 2^{-65}. We find eighty-four valid 39-round DCs. It is worth noting that the refined probabilities of two of them are both 2^{-64} and the refined probabilities of the remaining eighty-two DCs are all 2^{-65}. Interestingly, if we search for DCs starting from the 5-th round, the two DCs are invalid, which means that the choice of the round constants can affect the security of ARX ciphers against differential cryptanalysis. The two DCs with a refined probability of 2^{-64} are shown in Table 9 in Appendix B, and we use $\mathcal{DC}_1$ and $\mathcal{DC}_2$ to denote them, respectively.

4.3 Application to Alzette

At CRYPTO'20, Beierle et al. [7] proposed a 64-bit ARX-based S-box called Alzette. We present the description of Alzette in Supplementary Material. Since Alzette is an S-box, it is important to evaluate the probabilities of differentials. Therefore, the probabilities of DCs need to be precisely evaluated.

We use our automatic tool to search for DCs of Alzette with $c = \texttt{0xb7e15162}$, one of the S-boxes used in SPARKLE [8], one of 10 finalists in NIST's lightweight cryptography standardization process.

- **Setting $r = 4$ and $w = 23$**, we can find 4-round DCs with the original probability of 2^{-23}. Furthermore, we find that the refined probabilities of two of them are 0 and 2^{-22}, respectively. The two DCs are shown in Table 11 in Appendix B, and we use $\mathcal{DC}_1$ and $\mathcal{DC}_2$ to denote the invalid DC and the valid DC with incorrect probability computed following the Markov cipher assumption, respectively.
- **Setting $r = 4$ and $w = 38$**, we can find a 4-round DC with the original probability of 2^{-38}, and we find that the refined probability of the DC is 0. However, the invalidity of the DC cannot be detected by Leurent's ARX Toolkit, which means that Leurent's ARX Toolkit cannot filter all invalid DCs. The DC is shown in Table 11 in Appendix B, and we use $\mathcal{DC}_3$ to denote the DC.

Experimental Verification of Probabilities. Since Alzette has no key injection, the probabilities of DCs for Alzette can be well evaluated using experiments. In this section, we evaluate the probabilities of $\mathcal{DC}_1$, $\mathcal{DC}_2$, and $\mathcal{DC}_3$ experimentally. We randomly select 2^{38}, 2^{38}, and 2^{45} samples to evaluate the probabilities of $\mathcal{DC}_1$, $\mathcal{DC}_2$, and $\mathcal{DC}_3$, respectively. The results of our calculations are given in Table 3. According to the results, we claim that cryptanalysts can get more precise probabilities of DCs using our new method.

Table 3. Experimental Probabilities of $\mathcal{DC}_1$, $\mathcal{DC}_2$, and $\mathcal{DC}_3$. w_O: the logarithm of the original probability. w_R: the logarithm of the refined probability. SS: Sample Size. EV: Experimental Value. EVO: Expected Value under the Original probability. EVR: Expected Value under the Refined probability. DEO: Difference between EV and EVO. DER: Difference between EV and EVR.

Name	w_O	w_R	SS	EV	EVO	EVR	DEO	DER
$\mathcal{DC}_1$	-23	$-\infty$	2^{38}	0	32768	0	-32768	0
$\mathcal{DC}_2$	-23	-22	2^{38}	74232	32768	65536	41464	8696
$\mathcal{DC}_3$	-38	$-\infty$	2^{45}	0	128	0	-128	0

For the probability of $\mathcal{DC}_2$, according to the results shown in Table 3, there is a 126.54% deviation between the experimental value and the theoretical value under the original probability. In contrast, there is a 13.27% deviation between the experimental value and the theoretical value under the refined probability. We believe that the deviation between the experimental value and the theoretical value under the refined probability is caused by the XOR of one of the input branches and the output branch of the modular addition in each round, which means that the whitening outputs assumption may affect the accuracy of the calculation. However, in this paper, we emphasize that our main concern is how to automatically find DCs with more precise probabilities than previous methods, how to automatically refine the probabilities of DCs, and how to automatically find new IDs. Therefore, although there is a deviation between the experimental value and the theoretical value under the refined probability, our results are more precise than the results derived by previous methods.

4.4 Application to XTEA

XTEA [28] has a 64-bit block size and a 128-bit key size. We present the description of XTEA in Supplementary Material. At ASIACRYPT'20, Azimi et al. [5] applied an SMT-based method to search for DCs in ARX ciphers, and they found an 18-round related-key DC with a probability of 2^{-57} for XTEA. The related-key DC is shown in Table 8 in Appendix B.

Setting $r = 18$ and $w = 57$, we find that the 18-round related-key DC is valid, and the probability is correct.

Setting $r = 10$ and $w = 58, 56$, we find two 10-round DCs with refined probabilities of 0 and 2^{-49}, respectively. The two DCs are shown in Table 10 in Appendix B, and we use $\mathcal{DC}_1$ and $\mathcal{DC}_2$ to denote the invalid DC and the valid DC with incorrect probability computed following the Markov cipher assumption, respectively.

Setting $r = 9$ and $w = 60$, we find a 9-round invalid DC. However, the invalidity of the DC cannot be detected by Leurent's ARX Toolkit, which means that Leurent's ARX Toolkit cannot filter all invalid DCs. The DC is shown in Table 10 in Appendix B, and we use $\mathcal{DC}_3$ to denote the DC.

4.5 Application to the `quarterround` of Salsa20

Salsa20 [9] is a stream cipher designed by Bernstein in 2005 as a candidate for the eSTREAM competition. The original proposal was for 20 rounds. A Salsa20 round consists of four parallel `quarterround` functions. We present the description of Salsa20 in Supplementary Material.

For `quarterround`, setting $r = 1$ and $w = 13$, we find an invalid DC. However, if we use differences to represent the DC in Leurent's ARX Toolkit, the invalidity of the DC cannot be detected by Leurent's ARX Toolkit, which means that Leurent's ARX Toolkit cannot filter all invalid DCs. The DC is shown in Table 12 in Appendix B, and we use $\mathcal{DC}_{Sa}$ to denote the DC.

4.6 Application to the round function of Chaskey

Chaskey [25] is a lightweight MAC algorithm whose underlying primitive is an ARX-based permutation in an Even-Mansour construction. The permutation operates on four 32-bit words and consists of 12 applications of a round function. We present the description of Chaskey in Supplementary Material.

- **For the round function of Chaskey, setting $r = 1$ and $w = 11$**, we find an invalid DC. However, if we use differences to represent the DC in Leurent's ARX Toolkit, the invalidity of the DC cannot be detected by Leurent's ARX Toolkit, which means that Leurent's ARX Toolkit cannot filter all invalid DCs. The DC is shown in Table 13 in Appendix B, and we use $\mathcal{DC}_{Ch}$ to denote the DC.

Remark 8. We randomly select 2^{29} and 2^{27} samples to experimentally verify the probabilities of $\mathcal{DC}_{Sa}$ and $\mathcal{DC}_{Ch}$, respectively. The results are given in Table 4. According to the results, we claim that cryptanalysts can get more precise probabilities of DCs using our new method.

Table 4. Experimental Probabilities of $\mathcal{DC}_{Sa}$ and $\mathcal{DC}_{Ch}$. w_O: the logarithm of the original probability. w_R: the logarithm of the refined probability. SS: Sample Size. EV: Experimental Value. EVO: Expected Value under the Original probability. EVR: Expected Value under the Refined probability. DEO: Difference between EV and EVO. DER: Difference between EV and EVR.

Name	w_O	w_R	SS	EV	EVO	EVR	DEO	DER
$\mathcal{DC}_{Sa}$	-13	$-\infty$	2^{29}	0	65536	0	-65536	0
$\mathcal{DC}_{Ch}$	-11	$-\infty$	2^{27}	0	65536	0	-65536	0

5 Automatic Search Tool for Impossible Differential Distinguishers

In this section, according to the modeling method in Section 3.2, we propose a SAT-based automatic search tool for IDs and further apply the tool to CHAM-64/128.

5.1 Modeling the Automatic Search Tool with a SAT/SMT Solver

In [26], Mouha et al. used a SAT/SMT solver to automatically search for DCs for Salsa20. Although they found that the probabilities of some DCs were incorrect, their method only includes Lipmaa et al.'s conditions and follows the Markov cipher assumption, which means that they cannot automatically find the correct probability. Similarly, the previous methods of searching for IDs may ignore some

IDs because the previous methods only include Lipmaa et al.'s conditions and follow the Markov cipher assumption.

To better evaluate the security of ARX ciphers against impossible differential cryptanalysis, our automatic search model includes the constraints included in previous methods and includes signed differential propagation and additive sum propagation to add more information of value propagation to XOR differential propagation.

For the description of XOR differences, we use the method proposed by Mouha et al. We refer the readers to [26] for details. In the rest of this section, without loss of generality, we assume that all XOR differential variables Δa have the same word size n.

For the description of signed differences, we create $\mathcal{A} = \{a_0^s, a_1^s, \ldots, a_{n-1}^s\}$ for each Δa, where $a_0^s, a_1^s, \ldots, a_{n-1}^s \in \mathbb{F}_2^m$. Then, we can use a_i^s to represent $\Delta^{\pm} a[i]$. However, since the formula of a SAT/SMT problem does not contain signed variables, we cannot represent signed differences directly. To overcome this problem, when $\Delta^{\pm} a[i] = -1$, we use the complement of a_i^s to represent $\Delta^{\pm} a[i]$, where $0 \leq i \leq n-1$. Besides, since rotations (resp. shifts) may move the most significant bit to another bit position, $\Delta^{\pm} a[n-1] = -1$ is different from $\Delta^{\pm} a[n-1] = 1$. Therefore, we need $m > n$. Then, we have

$$a_i^s = \begin{cases} 0^m & \text{if } \Delta^{\pm} a[i] = 0 \\ 1^{m-i} \parallel 0^i & \text{if } \Delta^{\pm} a[i] = -1 \\ 0^{m-i-1} \parallel 1 \parallel 0^i & \text{if } \Delta^{\pm} a[i] = 1, \end{cases}$$

where $0 \leq i \leq n-1$. For the description of additive sums, the situation is similar to the description of signed differences. We name the new variables as bit-signed differential variables and bit-additive sum variables, respectively.

Then, we can use the method in Section 3.2 to model XOR differential propagation, signed differential propagation, and additive sum propagation.

Finally, for each modular addition, we parse each bit-signed differential variable and each bit-additive sum variable to get the values of corresponding input and output bits. We make the sum (i.e. addition modulo 2^n) of two input texts equal to the corresponding output text to ensure the input and output texts are valid.

So far, we can build an automatic search tool for IDs using the above complete architecture descriptions. Moreover, the automatic tool can also be used to evaluate whether a DC is valid. In applications, we find some new IDs ignored by previous methods.

5.2 Application to CHAM-64/128

To the best of our knowledge, the existing longest ID for CHAM-64/128 covers 18 rounds [18] with $wt(\texttt{InD}) = 1$ and $wt(\texttt{OutD}) = 2$. Using our automatic tool, we search for IDs starting from the 3-rd round. We find five 20-round IDs with $wt(\texttt{InD}) = wt(\texttt{OutD}) = 1$ for the first time. The five 20-round IDs cannot be found by using previous methods. Then, we find nineteen 19-round IDs and one

19-round ID with $wt(\mathtt{InD}) = wt(\mathtt{OutD}) = 1$ for the first time using our method and previous methods, respectively. Besides, using our method, we find 47, 704, 2537, and 3836 IDs with $wt(\mathtt{InD}) = wt(\mathtt{OutD}) = 1$ for 18-round, 16-round, 14-round, and 12-round CHAM-64/128, respectively. In contrast, using previous methods, we find 2, 392, 1915, and 3477 IDs with $wt(\mathtt{InD}) = wt(\mathtt{OutD}) = 1$ for 18-round, 16-round, 14-round, and 12-round CHAM-64/128, respectively. Therefore, using our automatic tool to search for IDs, we can find some new distinguishers ignored by previous methods. Interestingly, if we search for IDs starting from the 1-st round, we cannot find any 20-round ID using our tool or previous methods, which means that the choice of the round constants can affect the security of ARX ciphers against impossible differential cryptanalysis. The numbers of IDs for CHAM-64/128 are shown in Table 5. Furthermore, the 20-round IDs are shown in Table 14 in Appendix B.

Table 5. The numbers of IDs for CHAM-64/128.

Method	$(wt(\mathtt{InD}), wt(\mathtt{OutD}))$	$20r$	$19r$	$18r$	$16r$	$14r$	$12r$	Ref.
Previous methods	$(1, 2)$	-	-	1	-	-	-	[18]
Previous methods	$(1, 1)$	0	1	2	392	1915	3477	this paper
Our method	$(1, 1)$	5	19	47	704	2537	3836	this paper

6 Conclusion

In this paper, we focus on solving the problem that general ARX-based components do not follow the Markov cipher assumption.We improve the evaluation theory of differential propagation for ARX operations by using signed differences and additive sums, and then we propose the first corresponding modeling method. We further propose an automatic search algorithm for DCs with more precise probabilities and a SAT-based automatic search tool for IDs. As an application, we derive more precise probabilities of DCs for CHAM-64/128, Alzette, XTEA, the `quarterround` of Salsa20, and the round function of Chaskey, and we also find some new IDs for CHAM-64/128.

Our new theory presents a profound description of differential propagation for ARX operations. The new proposed tools support more precise automatic search results. According to our results, the differential (resp. impossible differential) attack constructed by the previous methods of placing a DC (resp. an ID) anywhere in a block cipher may be invalid. Our new theory and tools are helpful for evaluating the security of ARX ciphers against differential and impossible differential cryptanalysis. On the other hand, for the design of ARX ciphers, we believe that if two modular additions share an identical branch (e.g. two consecutive modular additions or two parallel modular additions that share an identical input branch), then this branch should be whitened (e.g. XORing the state with a key). Furthermore, we believe that the round constants should be chosen carefully because the choice of the round constants can affect the probabilities of DCs, affect the validity of DCs, and affect the number of rounds covered by IDs.

References

1. National Institute of Standards and Technology. fips 180-1: Secure hash standard, april 1995.
2. National Institute of Standards and Technology: Lightweight Cryptography Project (2019), https://csrc.nist.gov/projects/lightweight-cryptography
3. Ankele, R., Kölbl, S.: Mind the Gap - A Closer Look at the Security of Block Ciphers against Differential Cryptanalysis. In: Cid, C., Jr., M.J.J. (eds.) SAC 2018. LNCS, vol. 11349, pp. 163–190. Springer (2018)
4. Aumasson, J., Meier, W., Phan, R.C., Henzen, L.: The Hash Function BLAKE. Information Security and Cryptography, Springer (2014)
5. Azimi, S.A., Ranea, A., Salmasizadeh, M., Mohajeri, J., Aref, M.R., Rijmen, V.: A Bit-Vector Differential Model for the Modular Addition by a Constant. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part I. LNCS, vol. 12491, pp. 385–414. Springer (2020)
6. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. IACR Cryptology ePrint Archive **2013**, 404 (2013)
7. Beierle, C., Biryukov, A., dos Santos, L.C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: Alzette: A 64-Bit ARX-box - (Feat. CRAX and TRAX). In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 419–448. Springer (2020)
8. Beierle, C., Biryukov, A., dos Santos, L.C., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: SCHWAEMM and ESCH: lightweight authenticated encryption and hashing using the SPARKLE permutation family. NIST round 2 lightweight candidate (2019)
9. Bernstein, D.J.: The Salsa20 Family of Stream Ciphers. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs - The eSTREAM Finalists, LNCS, vol. 4986, pp. 84–97. Springer (2008)
10. Biryukov, A., Perrin, L.: State of the Art in Lightweight Symmetric Cryptography. IACR Cryptology ePrint Archive **2017**, 511 (2017)
11. Biryukov, A., Roy, A., Velichkov, V.: Differential Analysis of Block Ciphers SIMON and SPECK. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 546–570. Springer (2014)
12. Biryukov, A., Velichkov, V., Corre, Y.L.: Automatic Search for the Best Trails in ARX: Application to Block Cipher Speck. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 289–310. Springer (2016)
13. Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 268–288. Springer (2016)
14. Gueron, S., Jha, A., Nandi, M.: COMET: COunter Mode Encryption with authentication Tag. In: Submission to NIST Lightweight Cryptography Project (2019)
15. Han, Y., Wang, M.: Automatically Search for Three Types of Block Cipher Distinguishers Based on SAT/SMT Solver. Ph.D Thesis (2018)
16. Huang, M., Wang, L.: Automatic Tool for Searching for Differential Characteristics in ARX Ciphers and Applications. In: Hao, F., Ruj, S., Gupta, S.S. (eds.) INDOCRYPT 2019. LNCS, vol. 11898, pp. 115–138. Springer (2019)

17. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON Block Cipher Family. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 161–185. Springer (2015)
18. Koo, B., Roh, D., Kim, H., Jung, Y., Lee, D., Kwon, D.: CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices. In: Kim, H., Kim, D. (eds.) ICISC 2017. LNCS, vol. 10779, pp. 3–25. Springer (2017)
19. Lai, X., Massey, J.L., Murphy, S.: Markov Ciphers and Differential Cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT '91. LNCS, vol. 547, pp. 17–38. Springer (1991)
20. Leurent, G.: Analysis of Differential Attacks in ARX Constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 226–243. Springer (2012)
21. Leurent, G.: Construction of differential characteristics in ARX designs application to skein. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 241–258. Springer (2013)
22. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 336–350. Springer (2001)
23. Liu, Z., Li, Y., Jiao, L., Wang, M.: A new method for searching optimal differential and linear trails in ARX ciphers. IEEE Trans. Inf. Theory **67**(2), 1054–1068 (2021)
24. Matsui, M.: On Correlation Between the Order of S-boxes and the Strength of DES. In: Santis, A.D. (ed.) EUROCRYPT '94. LNCS, vol. 950, pp. 366–375. Springer (1994)
25. Mouha, N., Mennink, B., Herrewege, A.V., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In: Joux, A., Youssef, A.M. (eds.) SAC 2014. LNCS, vol. 8781, pp. 306–323. Springer (2014)
26. Mouha, N., Preneel, B.: Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. IACR Cryptology ePrint Archive **2013**, 328 (2013)
27. Mouha, N., Velichkov, V., Cannière, C.D., Preneel, B.: The Differential Analysis of S-Functions. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 36–56. Springer (2010)
28. Needham, R.M., Wheeler, D.J.: TEA extensions. Computer Laboratory, Cambridge University, England (1997)
29. Rivest, R.L.: The MD4 message digest algorithm. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO '90. LNCS, vol. 537, pp. 303–311. Springer (1990)
30. Rivest, R.L.: The MD5 message-digest algorithm. RFC **1321**, 1–21 (1992)
31. Roh, D., Koo, B., Jung, Y., Jeong, I., Lee, D., Kwon, D., Kim, W.: Revised Version of Block Cipher CHAM. In: Seo, J.H. (ed.) ICISC 2019. LNCS, vol. 11975, pp. 1–19. Springer (2019)
32. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer (2005)
33. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer (2005)
34. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer (2005)
35. Wheeler, D.J., Needham, R.M.: TEA, a Tiny Encryption Algorithm. In: Preneel, B. (ed.) FSE '94. LNCS, vol. 1008, pp. 363–366. Springer (1994)
36. Zhou, C., Zhang, W., Ding, T., Xiang, Z.: Improving the MILP-based Security Evaluation Algorithms against Differential Cryptanalysis Using Divide-and-Conquer Approach. IACR Cryptology ePrint Archive **2019**, 19 (2019)

A Inapplicability of the Method Proposed in [27]

In [27], Mouha et al. treated two consecutive modular additions as a single modular addition with three inputs and one output to improve the differential probability. Then, they applied it to the two consecutive modular additions in one round of XTEA. However, we find that their method is not suitable for the case of two consecutive modular additions with a branch injection in the middle, which means that the method cannot be used to evaluate the probabilities of DCs for CHAM, Alzette, and XTEA. We present a counter-example, which is further explained later.

Counter-example 4 *For two consecutive modular additions with a branch injection in the middle shown in Fig. 2, we evaluate the XOR differential (not a DC) probability for each value of the injecting branch f with enumeration experiments, where f can be a key, a round constant, or an internal state. Without loss of generality, suppose that $\Delta a = \texttt{0x1}$, $\Delta b = \texttt{0x0}$, $\Delta d = \texttt{0x1}$, and $\Delta e = \texttt{0x0}$. Then, we find that the XOR differential probability is 2^{-1} iff $f \in \{0, 7, 8, 15\}$. In contrast, for $f \in \mathbb{F}_2^4$, the XOR differential probability is 2^{-1} using Mouha et al.'s method. The results are shown in Table 6.*

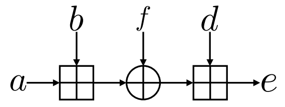

Fig. 2. Two modular additions with a branch injection in the middle

Table 6. Experimental Probabilities of two consecutive modular additions with a branch injection in the middle.

f	#{ (a, a', b, b', d, d')}	Pr
0	2048	2^{-1}
1	1024	2^{-2}
2	1024	2^{-2}
3	1536	$2^{-1.415}$
4	1536	$2^{-1.415}$
5	1024	2^{-2}
6	1024	2^{-2}
7	2048	2^{-1}
8	2048	2^{-1}
9	1024	2^{-2}
10	1024	2^{-2}
11	1536	$2^{-1.415}$
12	1536	$2^{-1.415}$
13	1024	2^{-2}
14	1024	2^{-2}
15	2048	2^{-1}

According to the results shown in Table 6, Mouha et al.'s method is not suitable for the case of two consecutive modular additions with a branch injection in the middle. Thus, the calculation result of the probability of the XOR differential quartet $(\alpha, 0, \alpha \mapsto 0)$ with $\alpha = \texttt{0x80402010}$ over the two consecutive modular additions in one round of XTEA proposed in [27] is incorrect. Furthermore, the method is not suitable for evaluating the probabilities of DCs for CHAM, Alzette, and XTEA. In contrast, using our method, one can easily handle all cases of the injecting branch. Therefore, for the case of two consecutive modular additions with a branch injection in the middle, we believe that the two modular additions should be considered independent, which means that it is more appropriate to evaluate the probability of a DC than the probability of a differential. Then, one can get clear relations between bits using Corollary 1 and Theorem 1, and precisely evaluate the probability using Theorem 2.

B Differential and Impossible Differential Characteristics

In the rest of this section, we use w_O and w_R to denote the logarithm of the probability computed following the Markov cipher assumption and the logarithm of the probability refined by our automatic algorithm, respectively.

Table 7. The 39-round DCs for CHAM-64/128. $\mathcal{DC}_1$ and $\mathcal{DC}_2$ denote the DC proposed in [16] and the DC proposed in [31], respectively.

	$\mathcal{DC}_1$			$\mathcal{DC}_2$		
r	ΔX_{r-1}	w_O	w_R	ΔX_{r-1}	w_O	w_R
1	0x0020 0x0010 0x1020 0x2800	0	0	0x0102 0x0280 0x0000 0x0400	0	0
2	0x0010 0x1020 0x2800 0x0000	-1	-1	0x0280 0x0000 0x0400 0x0204	-3	-3
3	0x1020 0x2800 0x0000 0x4000	-2	-2	0x0000 0x0400 0x0204 0x0500	-2	-2
4	0x2800 0x0000 0x4000 0x2040	-3	-3	0x0400 0x0204 0x0500 0x0008	-1	-1
5	0x0000 0x4000 0x2040 0x5000	-2	-2	0x0204 0x0500 0x0008 0x0004	-2	-2
6	0x4000 0x2040 0x5000 0x0080	0	0	0x0500 0x0008 0x0004 0x0408	-3	-3
7	0x2040 0x5000 0x0080 0x0040	-2	-2	0x0008 0x0004 0x0408 0x0a00	-3	-3
8	0x5000 0x0080 0x0040 0x4080	-2	-2	0x0004 0x0408 0x0a00 0x0000	-1	-1
9	0x0080 0x0040 0x4080 0xa000	-2	-2	0x0408 0x0a00 0x0000 0x1000	-2	-2
10	0x0040 0x4080 0xa000 0x0000	-1	-1	0x0a00 0x0000 0x1000 0x0810	-3	-3
11	0x4080 0xa000 0x0000 0x0001	-1	-1	0x0000 0x1000 0x0810 0x1400	-2	-2
12	0xa000 0x0000 0x0001 0x8100	-3	-3	0x1000 0x0810 0x1400 0x0020	-1	-1
13	0x0000 0x0001 0x8100 0x4001	-1	-1	0x0810 0x1400 0x0020 0x0010	-2	-2
14	0x0001 0x8100 0x4001 0x0200	-1	-1	0x1400 0x0020 0x0010 0x1020	-3	-3
15	0x8100 0x4001 0x0200 0x0100	-2	-2	0x0020 0x0010 0x1020 0x2800	-3	-3
16	0x4001 0x0200 0x0100 0x0201	-2	-2	0x0010 0x1020 0x2800 0x0000	-1	-1
17	0x0200 0x0100 0x0201 0x8003	-3	-3	0x1020 0x2800 0x0000 0x4000	-2	-2
18	0x0100 0x0201 0x8003 0x0000	-1	-1	0x2800 0x0000 0x4000 0x2040	-3	-3
19	0x0201 0x8003 0x0000 0x0004	-2	-2	0x0000 0x4000 0x2040 0x7000	-3	-3
20	0x8003 0x0000 0x0004 0x0402	-4	-4	0x4000 0x2040 0x7000 0x0080	0	0
21	0x0000 0x0004 0x0402 0x0007	-2	-2	0x2040 0x7000 0x0080 0x0040	-2	-2
22	0x0004 0x0402 0x0007 0x0800	-1	-1	0x7000 0x0080 0x0040 0x4000	-3	-3
23	0x0402 0x0007 0x0800 0x0400	-2	-2	0x0080 0x0040 0x4000 0x2000	$\mathbf{-3}$	$\boldsymbol{-\infty}$
24	0x0007 0x0800 0x0400 0x0004	-4	-4	0x0040 0x4000 0x2000 0x0000	-1	-1
25	0x0800 0x0400 0x0004 0x0002	-4	-4	0x4000 0x2000 0x0000 0x0000	-1	-1
26	0x0400 0x0004 0x0002 0x0000	-1	-1	0x2000 0x0000 0x0000 0x0000	-1	-1
27	0x0004 0x0002 0x0000 0x0000	-1	-1	0x0000 0x0000 0x0000 0x4000	-1	-1
28	0x0002 0x0000 0x0000 0x0000	-1	-1	0x0000 0x0000 0x4000 0x0000	0	0
29	0x0000 0x0000 0x0000 0x0004	-1	-1	0x0000 0x4000 0x0000 0x0000	0	0
30	0x0000 0x0000 0x0040 0x0000	0	0	0x4000 0x0000 0x0000 0x0080	0	0
31	0x0000 0x0004 0x0000 0x0000	0	0	0x0000 0x0000 0x0080 0x8000	-1	-1
32	0x0004 0x0000 0x0000 0x0800	-1	-1	0x0000 0x0080 0x8000 0x0000	0	0
33	0x0000 0x0000 0x0800 0x0008	-1	-1	0x0080 0x8000 0x0000 0x0001	0	0
34	0x0000 0x0800 0x0008 0x0000	0	0	0x8000 0x0000 0x0001 0x8100	-2	-2
35	0x0800 0x0008 0x0000 0x0010	-1	-1	0x0000 0x0001 0x8100 0x0001	0	0
36	0x0008 0x0000 0x0010 0x1008	-2	-2	0x0001 0x8100 0x0001 0x0200	-1	-1
37	0x0000 0x0010 0x1008 0x0010	-1	-1	0x8100 0x0001 0x0200 0x0100	-2	-2
38	0x0010 0x1008 0x0010 0x2000	-1	-1	0x0001 0x0200 0x0100 0x0281	-2	-2
39	0x1008 0x0010 0x2000 0x1000	-2	-2	0x0200 0x0100 0x0281 0x0002	-2	-2
40	0x0010 0x2000 0x1000 0x2810	-3	-3	0x0100 0x0281 0x0002 0x0000	-1	-1
$\sum_r w_O$	-64			-63		
$\sum_r w_R$	-64			$-\infty$		

Table 8. The 18-round related-key DC [5] for XTEA. w_K denotes the logarithm of the probability of the round-key DC.

r	ΔL_r	ΔR_r	w_O	w_R	ΔK_r	w_K
1	0xc4310800	0x00010000	0	0	0x00000000	0
2	0x00010000	0x44200000	−9	−9	0x00000000	0
3	0x44200000	0x04000000	−6	−6	0x00000000	0
4	0x04000000	0x80000000	−6	−6	0x80000000	0
5	0x80000000	0x00000000	−2	−2	0x80000000	0
6	0x00000000	0x00000000	0	0	0x00000000	0
7	0x00000000	0x00000000	0	0	0x00000000	0
8	0x00000000	0x00000000	0	0	0x00000000	0
9	0x00000000	0x00000000	0	0	0x00000000	0
10	0x00000000	0x00000000	0	0	0x00000000	0
11	0x00000000	0x00000000	0	0	0x00000000	0
12	0x00000000	0x00000000	0	0	0x00000000	0
13	0x00000000	0x00000000	0	0	0x80000000	0
14	0x00000000	0x80000000	0	0	0x80000000	0
15	0x80000000	0x04000000	−2	−2	0x00000000	0
16	0x04000000	0x44200000	−6	−6	0x00000000	0
17	0x44200000	0x00010000	−6	−6	0x00000000	0
18	0x00010000	0xc4310800	−9	−9	0x00000000	0
19	0xc4310800	0x01010040	−11	−11		
$\sum_r w_O$	−57					
$\sum_r w_R$	−57					

Table 9. The 39-round DCs for CHAM-64/128 with the original probability of 2^{-65} and a refined probability of 2^{-64}.

	$\mathcal{DC}_1$			$\mathcal{DC}_2$		
r	ΔX_{r-1}	w_O	w_R	ΔX_{r-1}	w_O	w_R
1	0x0000 0x4000 0x2040 0x5000	0	0	0x2040 0x5000 0x0080 0x0040	0	0
2	0x4000 0x2040 0x5000 0x0080	0	0	0x5000 0x0080 0x0040 0x4080	−2	−2
3	0x2040 0x5000 0x0080 0x0040	−2	−2	0x0080 0x0040 0x4080 0xa000	−2	−2
4	0x5000 0x0080 0x0040 0x4080	−2	−2	0x0040 0x4080 0xa000 0x0000	−1	−1
5	0x0080 0x0040 0x4080 0xa000	−2	−2	0x4080 0xa000 0x0000 0x0001	−1	−1
6	0x0040 0x4080 0xa000 0x0000	−1	−1	0xa000 0x0000 0x0001 0x8100	−3	−3
7	0x4080 0xa000 0x0000 0x0001	−1	−1	0x0000 0x0001 0x8100 0x4001	−1	−1
8	0xa000 0x0000 0x0001 0x8100	−3	−3	0x0001 0x8100 0x4001 0x0200	−1	−1
9	0x0000 0x0001 0x8100 0x4001	−1	−1	0x8100 0x4001 0x0200 0x0100	−2	−2
10	0x0001 0x8100 0x4001 0x0200	−1	−1	0x4001 0x0200 0x0100 0x0201	−2	−2
11	0x8100 0x4001 0x0200 0x0100	−2	−2	0x0200 0x0100 0x0201 0x8002	−3	−3
12	0x4001 0x0200 0x0100 0x0201	−2	−2	0x0100 0x0201 0x8002 0x0000	−1	−1
13	0x0200 0x0100 0x0201 0x8002	−3	−3	0x0201 0x8002 0x0000 0x0004	−2	−2
14	0x0100 0x0201 0x8002 0x0000	−1	−1	0x8002 0x0000 0x0004 0x0402	−3	−3
15	0x0201 0x8002 0x0000 0x0004	−2	−2	0x0000 0x0004 0x0402 0x0005	−1	−1
16	0x8002 0x0000 0x0004 0x0402	−3	−3	0x0004 0x0402 0x0005 0x0800	−1	−1
17	0x0000 0x0004 0x0402 0x0005	−1	−1	0x0402 0x0005 0x0800 0x0400	−2	−2
18	0x0004 0x0402 0x0005 0x0800	−1	−1	0x0005 0x0800 0x0400 0x0804	−3	−3
19	0x0402 0x0005 0x0800 0x0400	−2	−2	0x0800 0x0400 0x0804 0x000a	−3	−3
20	0x0005 0x0800 0x0400 0x0804	−3	−3	0x0400 0x0804 0x000a 0x0000	−1	−1
21	0x0800 0x0400 0x0804 0x000a	−3	−3	0x0804 0x000a 0x0000 0x0010	−2	−2
22	0x0400 0x0804 0x000a 0x0000	−1	−1	0x000a 0x0000 0x0010 0x1008	−3	−3
23	0x0804 0x000a 0x0000 0x0010	−2	−2	0x0000 0x0010 0x1008 0x001c	−3	−3
24	0x000a 0x0000 0x0010 0x1008	−3	−3	0x0010 0x1008 0x001c 0x2000	−1	−1
25	0x0000 0x0010 0x1008 0x001c	−3	−3	0x1008 0x001c 0x2000 0x1000	−2	−2
26	0x0010 0x1008 0x001c 0x2000	−1	−1	0x001c 0x2000 0x1000 0x0010	−4	−4
27	0x1008 0x001c 0x2000 0x1000	−2	−2	0x2000 0x1000 0x0010 0x0008	−**4**	−**3**
28	0x001c 0x2000 0x1000 0x0010	−4	−4	0x1000 0x0010 0x0008 0x0000	−1	−1
29	0x2000 0x1000 0x0010 0x0008	−**4**	−**3**	0x0010 0x0008 0x0000 0x0000	−1	−1
30	0x1000 0x0010 0x0008 0x0000	−1	−1	0x0008 0x0000 0x0000 0x0000	−1	−1
31	0x0010 0x0008 0x0000 0x0000	−1	−1	0x0000 0x0000 0x0000 0x0010	−1	−1
32	0x0008 0x0000 0x0000 0x0000	−1	−1	0x0000 0x0000 0x0010 0x0000	0	0
33	0x0000 0x0000 0x0000 0x0010	−1	−1	0x0000 0x0010 0x0000 0x0000	0	0
34	0x0000 0x0000 0x0010 0x0000	0	0	0x0010 0x0000 0x0000 0x2000	−1	−1
35	0x0000 0x0010 0x0000 0x0000	0	0	0x0000 0x0000 0x2000 0x0020	−1	−1
36	0x0010 0x0000 0x0000 0x2000	−1	−1	0x0000 0x2000 0x0020 0x0000	0	0
37	0x0000 0x0000 0x2000 0x0020	−1	−1	0x2000 0x0020 0x0000 0x0040	−1	−1
38	0x0000 0x2000 0x0020 0x0000	0	0	0x0020 0x0000 0x0040 0x4020	−2	−2
39	0x2000 0x0020 0x0000 0x0040	−1	−1	0x0000 0x0040 0x4020 0x0040	−1	−1
40	0x0020 0x0000 0x0040 0x4020	−2	−2	0x0040 0x4020 0x0040 0x8000	−1	−1
$\sum_r w_O$	−65			−65		
$\sum_r w_R$	−64			−64		

Table 10. The 10-round and 9-round DCs for XTEA. $\mathcal{DC}_1$ and $\mathcal{DC}_3$ denote the invalid DCs, and $\mathcal{DC}_2$ denotes the valid DC with incorrect probability.

	$\mathcal{DC}_1$				$\mathcal{DC}_2$				$\mathcal{DC}_3$			
r	ΔL_r	ΔR_r	w_O	w_R	ΔL_r	ΔR_r	w_O	w_R	ΔL_r	ΔR_r	w_O	w_R
1	0x00000100	0x00000011	0	0	0x00003109	0x00000100	0	0	0x00000017	0x00000000	0	0
2	0x00000011	0x00000001	-5	-5	0x00000100	0x00000011	-9	-9	0x00000000	0x00000009	-5	-5
3	0x00000001	0x00000000	-4	-4	0x00000011	0x00000001	-5	-5	0x00000009	0x0000000f	-13	-13
4	0x00000000	0x00000001	-1	-1	0x00000001	0x00000000	-4	-4	0x0000000f	0x00000000	$\mathbf{-12}$	$\boldsymbol{-\infty}$
5	0x00000001	0x00000001	-9	-9	0x00000000	0x00000001	-1	-1	0x00000000	0x0000000f	-4	-4
6	0x00000001	0x00000000	-9	-9	0x00000001	0x0000003f	-8	-8	0x0000000f	0x00000001	$\mathbf{-9}$	$\boldsymbol{-\infty}$
7	0x00000000	0x00000001	-1	-1	0x0000003f	0x00000001	$\mathbf{-11}$	$\mathbf{-8}$	0x00000001	0x00000000	$\mathbf{-7}$	$\boldsymbol{-\infty}$
8	0x00000001	0x000000f3	-8	-8	0x00000001	0x00000000	$\mathbf{-8}$	$\mathbf{-4}$	0x00000000	0x00000001	-1	-1
9	0x000000f3	0x00000001	$\mathbf{-12}$	$\boldsymbol{-\infty}$	0x00000000	0x00000001	-1	-1	0x00000001	0x00000011	-4	-4
10	0x00000001	0x00000000	$\mathbf{-8}$	$\mathbf{-4}$	0x00000001	0x00000011	-4	-4	0x00000011	0x00000100	-5	-5
11	0x00000000	0x00000001	-1	-1	0x00000011	0x00000100	-5	-5				
$\sum_r w_O$	-58				-56				-60			
$\sum_r w_R$	$-\infty$				-49				$-\infty$			

Table 11. The 4-round DCs for Alzette. $\mathcal{DC}_1$ and $\mathcal{DC}_3$ denote the invalid DCs, and $\mathcal{DC}_2$ denotes the valid DC with incorrect probability.

	$\mathcal{DC}_1$				$\mathcal{DC}_2$				$\mathcal{DC}_3$			
r	ΔL_r	ΔR_r	w_O	w_R	ΔL_r	ΔR_r	w_O	w_R	ΔL_r	ΔR_r	w_O	w_R
1	0x00000001	0x00000000	0	0	0x00000004	0x00000000	0	0	0x00000000	0x00000007	0	0
2	0x00000003	0x00000300	-2	-2	0x0000000c	0x00000c00	-2	-2	0x00000012	0x00001207	-4	-4
3	0x01800001	0x000083c0	$\mathbf{-4}$	$\boldsymbol{-\infty}$	0x06000004	0x00020f00	$\mathbf{-4}$	$\mathbf{-3}$	0x0903800e	0xc0071686	$\mathbf{-9}$	$\boldsymbol{-\infty}$
4	0x008080c1	0x01018242	-8	-8	0x02020504	0x04060508	-8	-8	0xc9038f80	0x52000987	-14	-14
5	0x0102c240	0xc3418340	-9	-9	0x04070500	0x0106010f	-9	-9	0x49090752	0x5552408e	-11	-11
$\sum_r w_O$	-23				-23				-38			
$\sum_r w_R$	$-\infty$				-22				$-\infty$			

Table 12. The DC for `quarterround` with the original probability of 2^{-13} and a refined probability of 0.

	$\mathcal{DC}_{Sa}$					
r	Δa_0^{r-1}	Δa_1^{r-1}	Δa_2^{r-1}	Δa_3^{r-1}	w_O	w_R
1	0x00000009	0x00000000	0x00000e00	0x00000017	0	0
2	0x00440009	0x00000000	0x00000000	0x00000017	$\mathbf{-13}$	$\boldsymbol{-\infty}$
$\sum_r w_O$	-13					
$\sum_r w_R$	$-\infty$					

Table 13. The DC for the round function of Chaskey with the original probability of 2^{-11} and a refined probability of 0.

	$\mathcal{DC}_{Ch}$					
r	Δv_0^{r-1}	Δv_1^{r-1}	Δv_2^{r-1}	Δv_3^{r-1}	w_O	w_R
1	0x00000000	0x00000000	0x00000017	0x00000000	0	0
2	0x00000007	0x00000009	0x00070000	0x00012009	$\mathbf{-11}$	$\boldsymbol{-\infty}$
$\sum_r w_O$	-11					
$\sum_r w_R$	$-\infty$					

Table 14. The 20-round IDs for CHAM-64/128.

Round	Characteristic
20	(0x8000, 0x0000, 0x0000, 0x0000) $\not\leftrightarrow$ (0x0000, 0x8000, 0x0000, 0x0000)
20	(0x8000, 0x0000, 0x0000, 0x0000) $\not\leftrightarrow$ (0x0000, 0x4000, 0x0000, 0x0000)
20	(0x8000, 0x0000, 0x0000, 0x0000) $\not\leftrightarrow$ (0x0000, 0x2000, 0x0000, 0x0000)
20	(0x8000, 0x0000, 0x0000, 0x0000) $\not\leftrightarrow$ (0x0000, 0x1000, 0x0000, 0x0000)
20	(0x8000, 0x0000, 0x0000, 0x0000) $\not\leftrightarrow$ (0x0000, 0x0001, 0x0000, 0x0000)

C Automatic Search Algorithm for Differential Characteristics

Algorithm 1 Automatic Search Algorithm for Differential Characteristics with Weight Limit

```
Input: The number of round $r$; the weight limit $w$; the set of $\Omega$ $\Omega_S$.
Output: The DC $\Omega$ with the probability of $Pr_{Original} = 2^{-w}$ computed following the Markov cipher assumption.

1: Let $\Omega \leftarrow \emptyset$;
2: Convert Equation (4), Equation (6), Equation (7), and the probability limit $Pr_{Original} = 2^{-w}$ to the corresponding constraints;
3: Add the constraints to any existing automatic search algorithm for differential characteristics in ARX ciphers;
4: for $\Omega \in \Omega_S$ do
5:     Convert $\Omega$ to the corresponding constraints, then add the constraints to any existing automatic search algorithm for differential characteristics in ARX ciphers;
6: end for
7: Apply the improved automatic search algorithm to any ARX cipher;
8: if there is no valid DC $\Omega_r$ then
9:     if $\Omega_S = \emptyset$ then
10:        return $\perp$;
11:    end if
12:    if $\Omega_S \neq \emptyset$ then
13:        $\Omega \leftarrow \emptyset$;
14:    end if
15: end if
16: if there is a valid DC $\Omega_r$ with the probability of $Pr_{Original} = 2^{-w}$ then
17:     $\Omega \leftarrow \Omega_r$;
18: end if
19: return $\Omega$;
```

Remark 9. Note that we use signed differential propagation and additive sum propagation in the first part of Algorithm 2 to ensure that the output DCs are valid. However, in the second part of Algorithm 2, we only use signed differential propagation and the least significant bits (LSB) of additive sums to refine the probabilities of the output DCs. This is because only in rare cases can the probabilities of the DCs be improved by using the relations between the other bits of additive sums. Nevertheless, to the best of our knowledge, our algorithm can get more precise results than previous methods.

Algorithm 2 Automatic Search Algorithm for Differential Characteristics

Input: The number of round r; the weight limit w; $(NMA_1, NMA_2, \ldots, NMA_r)$ the list of the number of modular addition in the 1-st, 2-nd, ..., r-th round.

Output: Ω the set of valid r-round differential characteristics; Pr_{Real} the list of the refined probabilities of differential characteristics in Ω.

```
1:  Procedure Main:
2:  **Begin the program**
3:  Let $\Omega_S \leftarrow \emptyset$, $t \leftarrow 0$, and $flag \leftarrow 1$;
4:  **while** $flag = 1$ **do**
5:      Call Algorithm 1;
6:      **if** $\Omega \neq \emptyset$ **then**
7:          $t \leftarrow t + 1$, $count \leftarrow 0$, $\Omega_S$.add($\Omega$);
8:          Call Procedure Round-1;
9:      **end if**
10:     **if** $\Omega = \emptyset$ **then**
11:         $Pr_{Real} \leftarrow \emptyset$;
12:         $(\Omega, Pr_{Real}) \leftarrow (\Omega_S, (Pr^1_{Real}, Pr^2_{Real}, \ldots, Pr^t_{Real}))$;
13:         $flag \leftarrow 0$;
14:     **end if**
15: **end while**
16: **return** $(\Omega, Pr_{Real})$;
17: **Exit the program**

18: Procedure Round-1:
19: **for** $1 \leq j \leq NMA_1$ **do**
20:     Call Algorithm 4;
21: **end for**
22: Call Algorithm 3;
23: Call Procedure Round-2;
24: $Pr^t_{Real} \leftarrow 2^{-w} \times 2^{count}$;
25: Return to the upper procedure;

26: Procedure Round-$i$ ($2 \leq i \leq r - 1$):
27: Convert the sets $output^P_{i-1}$, $output^N_{i-1}$, and $output^{NA}_{i-1}$ of the relations between the output bits of the $(i-1)$-th round to the corresponding sets $input^P_i$, $input^N_i$, and $input^{NA}_i$ of the relations between the input bits of the $i$-th round, respectively;
28: Let $tmp^P_i \leftarrow input^P_i$, $tmp^N_i \leftarrow input^N_i$, and $tmp^{NA}_i \leftarrow input^{NA}_i$;
29: **for** $u \in tmp^P_i$, $v \in tmp^N_i$, and $w \in tmp^{NA}_i$ **do**
30:     **if** $u$ includes any differential bit of the modular additions in the $i$-th round **then**
31:         $tmp^P_i$.remove($u$);
32:     **end if**
33:     **if** $v$ includes any differential bit of the modular additions in the $i$-th round **then**
34:         $tmp^N_i$.remove($v$);
35:     **end if**
36:     **if** $w$ includes any differential bit of the modular additions in the $i$-th round **then**
37:         $tmp^{NA}_i$.remove($w$);
38:     **end if**
39: **end for**
40: According to Section 3.1, convert $tmp^P_i$, $tmp^N_i$, and $tmp^{NA}_i$ to the corresponding sets $input\_to\_output^P_i$, $input\_to\_output^N_i$, and $input\_to\_output^{NA}_i$ of the relations between the output bits of the $i$-th round not involved in the modular additions in the $i$-th round, respectively;
41: **for** $1 \leq j \leq NMA_i$ **do**
42:     Call Algorithm 4;
43: **end for**
44: Call Algorithm 3;
45: Call Procedure Round-$i + 1$;
46: Return to the upper procedure;

47: Procedure Round-$r$:
48: Convert $output^P_{r-1}$, $output^N_{r-1}$, and $output^{NA}_{r-1}$ to the corresponding $input^P_r$, $input^N_r$, and $input^{NA}_r$ of the r-th round, respectively;
49: **for** $1 \leq j \leq NMA_r$ **do**
50:     Call Algorithm 4;
51: **end for**
52: Call Algorithm 3;
53: Return to the upper procedure;
```

Algorithm 3 Automatic Derivation Algorithm for Modifying Factors

Input: The sets of relations between the bits of the modular additions in the i-th round; the sets of relations between the input bits of the i-th round; the round index i; the modifying factor $count$.

Output: The modifying factor $count$; the sets of relations between the output bits of the i-th round.

```
 1: Let $RR_0 \leftarrow \emptyset$, and $RR_1 \leftarrow \emptyset$;
 2: for $1 \leq k < NMA_i$ do
 3:     for $k < l \leq NMA_i$ do
 4:         for $p \in P_i^k$ do
 5:             for $q \in P_i^l$ do
 6:                 if $p = q$ then
 7:                     $RR_1$.add($p$);
 8:                 end if
 9:             end for
10:         end for
11:         for $p \in N_i^k$ do
12:             for $q \in N_i^l$ do
13:                 if $p = q$ then
14:                     $RR_1$.add($p$);
15:                 end if
16:             end for
17:         end for
18:         if $NA_i^k \neq \emptyset$ AND $NA_i^l \neq \emptyset$ then
19:             if $NA_i^k[0] = NA_i^l[0]$ then
20:                 $RR_0$.add($NA_i^k[0]$);
21:             end if
22:         end if
23:     end for
24: end for
25: if $2 \leq i \leq r$ then
26:     for $1 \leq k \leq NMA_i$ do
27:         for $p \in P_i^k$ do
28:             for $q \in input_i^P$ do
29:                 if $p = q$ then
30:                     $RR_1$.add($p$);
31:                 end if
32:             end for
33:         end for
34:         for $p \in N_i^k$ do
35:             for $q \in input_i^N$ do
36:                 if $p = q$ then
37:                     $RR_1$.add($p$);
38:                 end if
39:             end for
40:         end for
41:         if $NA_i^k \neq \emptyset$ AND $input_i^{NA} \neq \emptyset$ then
42:             for $p \in input_i^{NA}$ do
43:                 if $NA_i^k[0] = p$ then
44:                     $RR_0$.add($NA_i^k[0]$);
45:                 end if
46:             end for
47:         end if
48:     end for
49: end if
50: Remove redundant relations in $RR_0$ and $RR_1$; //Suppose $RR_1 = \{\{``\Delta^{\pm}x[l_1]", ``\Delta^{\pm}x[l_2]"\}, \{``\Delta^{\pm}x[l_1]", ``\Delta^{\pm}x[l_3]"\}, \{``\Delta^{\pm}x[l_2]", ``\Delta^{\pm}x[l_3]"\}\}$, where $0 \leq l_1 \neq l_2 \neq l_3 \leq n-1$. After removing redundant relations, $RR_1 = \{\{``\Delta^{\pm}x[l_1]", ``\Delta^{\pm}x[l_2]"\}, \{``\Delta^{\pm}x[l_1]", ``\Delta^{\pm}x[l_3]"\}\}$.
51: $count \leftarrow count + |RR_0| + |RR_1|$;
52: Let $output_i^P \leftarrow \emptyset$, $output_i^N \leftarrow \emptyset$, and $output_i^{NA} \leftarrow \emptyset$;
53: if $i < r$ then
54:     for $1 \leq j \leq NMA_i$ do
55:         $output_i^P$.add($P_i^j$);
56:         $output_i^N$.add($N_i^j$);
57:         $output_i^{NA}$.add($NA_i^j$);
58:     end for
59: end if
60: if $2 \leq i < r$ then
61:     $output_i^P$.add($input\_to\_output_i^P$);
62:     $output_i^N$.add($input\_to\_output_i^N$);
63:     $output_i^{NA}$.add($input\_to\_output_i^{NA}$);
64: end if
65: return ($count, output_i^P, output_i^N, output_i^{NA}$);
```

Algorithm 4 Automatic Derivation Algorithm for Relations between Bits

Input: The input and output XOR differences of the j-th modular addition in the i-th round. (Without loss of generality, we assume that the word size of the modular addition is n.)
Output: The sets of relations between the bits of the j-th modular addition in the i-th round.

```
 1: Let $Index \leftarrow \emptyset$, $Init_S \leftarrow \emptyset$, $P_i^j \leftarrow \emptyset$, $N_i^j \leftarrow \emptyset$, and $NA_i^j \leftarrow \emptyset$;
 2: Let $(\Delta x, \Delta y \mapsto \Delta z)$ denote the XOR differential triplet of the modular addition;
 3: $Index$.add($\{$“$\Delta^{\pm}x[l]$”$|\Delta^{\pm}x[l] \neq 0, 0 \leq l \leq n-1\}$);
 4: $Index$.add($\{$“$\Delta^{\pm}y[l]$”$|\Delta^{\pm}y[l] \neq 0, 0 \leq l \leq n-1\}$);
 5: $Index$.add($\{$“$\Delta^{\pm}z[l]$”$|\Delta^{\pm}z[l] \neq 0, 0 \leq l \leq n-1\}$);
 6: for each corresponding candidate $\Delta^{\pm}x$, $\Delta^{\pm}y$, and $\Delta^{\pm}z$ do
 7:     if $\Delta^{\pm}x \boxplus \Delta^{\pm}y = \Delta^{\pm}z \mod 2^n$ then
 8:         $Init_S$.add($\{\Delta^{\pm}x[l]|\Delta^{\pm}x[l] \neq 0, 0 \leq l \leq n-1\}$);
 9:         $Init_S$.add($\{\Delta^{\pm}y[l]|\Delta^{\pm}y[l] \neq 0, 0 \leq l \leq n-1\}$);
10:         $Init_S$.add($\{\Delta^{\pm}z[l]|\Delta^{\pm}z[l] \neq 0, 0 \leq l \leq n-1\}$);
11:     end if
12: end for
13: Let $HW \leftarrow wt(\Delta x) + wt(\Delta y) + wt(\Delta z)$;
14: Let $NC \leftarrow \#\{(\Delta^{\pm}x, \Delta^{\pm}y, \Delta^{\pm}z)|\Delta^{\pm}x \boxplus \Delta^{\pm}y = \Delta^{\pm}z \mod 2^n\}$;
15: for $0 \leq p < HW-1$ do
16:     for $p < q \leq HW-1$ do
17:         for $0 \leq s \leq NC-1$ do
18:             Let $PF \leftarrow 0$, and $NF \leftarrow 0$;
19:             if $Init_S[HW \times s + p] \times Init_S[HW \times s + q] = 1$ then
20:                 $PF \leftarrow PF + 1$;
21:             end if
22:             if $Init_S[HW \times s + p] \times Init_S[HW \times s + q] = -1$ then
23:                 $NF \leftarrow NF + 1$;
24:             end if
25:         end for
26:         if $PF = NC$ then
27:             $P_i^j$.add($\{Index[p], Index[q]\}$);
28:         end if
29:         if $NF = NC$ then
30:             $N_i^j$.add($\{Index[p], Index[q]\}$);
31:         end if
32:     end for
33: end for
34: if $\Delta x[0] + \Delta y[0] + \Delta z[0] = 2$ then
35:     if $\Delta x[0] = 0$ then
36:         $NA_i^j$.add($\nabla^{+}x[0] \neq 1$)
37:     end if
38:     if $\Delta y[0] = 0$ then
39:         $NA_i^j$.add($\nabla^{+}y[0] \neq 1$)
40:     end if
41:     if $\Delta z[0] = 0$ then
42:         $NA_i^j$.add($\nabla^{+}z[0] \neq 1$)
43:     end if
44: end if
45: return ($P_i^j$, $N_i^j$, $NA_i^j$);
```

Supplementary Material

A Proofs

A.1 Proof of Theorem 1

Proof. According to Equation (5), we have

$$\sum_{i=0}^{n-1} \nabla^+ x[i] \cdot 2^i = (x + x') \mod 2^{n+1},$$

$$\sum_{i=0}^{n-1} \nabla^+ y[i] \cdot 2^i = (y + y') \mod 2^{n+1},$$

and

$$\sum_{i=0}^{n-1} \nabla^+ z[i] \cdot 2^i = (z + z') \mod 2^{n+1}.$$

Then, we have

$$\nabla^+ x \boxplus \nabla^+ y = (x + x') \mod 2^{n+1} \boxplus (y + y') \mod 2^{n+1} = z \boxplus z'.$$

Let $t, t' \in \mathbb{F}_2^n$, and $t \oplus t' = \Delta z$. Then, for the case of $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] \neq 0$, we have three situations.

(i) For $z[n-1] \oplus z'[n-1] = 1$, we need to prove that $z \boxplus z' = t \boxplus t'$ iff $(z, z') = (t, t')$. Suppose that $z \boxplus z' = t \boxplus t'$ holds for some $(z, z') \neq (t, t')$, then we have

$$z + z' + 2^n = t + t' \quad \textit{or} \quad z + z' = t + t' + 2^n.$$

For the case of $z + z' + 2^n = t + t'$, we have two incompatible conditions

$$z + z' + 2^n \geq 2^n + 2^{n-1} \quad \textit{and} \quad t + t' \leq 2^n + 2^{n-1} - 2.$$

For the case of $z + z' = t + t' + 2^n$, the situation is similar to the above. Therefore, we have $\nabla^+ x \boxplus \nabla^+ y = \nabla^+ z \mod 2^n$.

(ii) For $z[n-1] \oplus z'[n-1] = 0$ and $\Delta x[n-1] = \Delta y[n-1] = 1$, suppose that $\nabla^+ x \boxplus \nabla^+ y = z \boxplus z'$ when $\Delta^\pm x[n-1] \times \Delta^\pm y[n-1] = 1$ (resp. -1). Then we have

$$(x \oplus 1 \parallel 0^{n-1}) \boxplus y = z \boxplus 2^{n-1},\ (x' \oplus 1 \parallel 0^{n-1}) \boxplus y' = z' \boxplus 2^{n-1}$$

and

$$x \boxplus (y \oplus 1 \parallel 0^{n-1}) = z \boxplus 2^{n-1},\ x' \boxplus (y' \oplus 1 \parallel 0^{n-1}) = z' \boxplus 2^{n-1}.$$

Therefore, we have $\nabla^+ x \boxplus \nabla^+ y = \nabla^+ z \mod 2^n$.

(iii) For $z[n-1] \oplus z'[n-1] = 0$ and $\Delta x[n-1] \neq \Delta y[n-1]$, suppose that $\nabla^+ x \boxplus \nabla^+ y = z \boxplus z'$ when $\Delta^{\pm} x[n-1] + \Delta^{\pm} y[n-1] = 1$ (resp. -1). Then we have

$$(x \oplus 1 \parallel 0^{n-1}) \boxplus y = z \boxplus 2^{n-1},\ (x' \oplus 1 \parallel 0^{n-1}) \boxplus y' = z' \boxplus 2^{n-1}$$

and

$$x \boxplus (y \oplus 1 \parallel 0^{n-1}) = z \boxplus 2^{n-1},\ x' \boxplus (y' \oplus 1 \parallel 0^{n-1}) = z' \boxplus 2^{n-1}.$$

Therefore, we have $\nabla^+ x \boxplus \nabla^+ y = \nabla^+ z \mod 2^n$.

Then, for $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] \neq 0$, Equation (6) holds.

For the case of $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] = 0$, we have

$$(\sum_{i=0}^{n-1} \nabla^+ x[i] \cdot 2^i + \sum_{i=0}^{n-1} \nabla^+ y[i] \cdot 2^i) \mod 2^{n+1} \neq \sum_{i=0}^{n-1} \nabla^+ z[i] \cdot 2^i \mod 2^{n+1}$$

iff $x+y \geq 2^n$ and $x'+y' < 2^n$ or $x+y < 2^n$ and $x'+y' \geq 2^n$. Suppose $x+y \geq 2^n$ and $x'+y' < 2^n$ or $x+y < 2^n$ and $x'+y' \geq 2^n$, we have

$$\nabla^+ x[n-1] + \nabla^+ y[n-1] = 2,$$

which means that

$$\sum_{i=0}^{n-2} x[i] \cdot 2^i + \sum_{i=0}^{n-2} y[i] \cdot 2^i \geq 2^n \quad or \quad \sum_{i=0}^{n-2} x'[i] \cdot 2^i + \sum_{i=0}^{n-2} y'[i] \cdot 2^i \geq 2^n.$$

Therefore, we have $(\nabla^+ x + \nabla^+ y) \mod 2^{n+1} = \nabla^+ z \mod 2^{n+1}$. Then, for $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] = 0$, Equation (7) holds. □

A.2 Proof of Theorem 2

Proof. The first statement is obvious, and we prove the second statement for the case of two consecutive modular additions.

Let $w = x \boxplus y$, and $u = w \boxplus z$. Let A denote that $(\Delta x, \Delta y \mapsto \Delta w)$ is a valid triplet, and B denote that $(\Delta w, \Delta z \mapsto \Delta u)$ is a valid triplet. According to Corollary 1, we can use Equation (4) to get the set RsW_s^f of the relations between the bits of $\Delta^{\pm} x$, $\Delta^{\pm} y$, and $\Delta^{\pm} w$, and the set RsW_s^s of the relations between the bits of $\Delta^{\pm} w$, $\Delta^{\pm} z$, and $\Delta^{\pm} u$. Similarly, according to Theorem 1, we can use Equation (6) and Equation (7) to get the set RsW_a^f of the relations between the bits of $\nabla^+ x$, $\nabla^+ y$, and $\nabla^+ w$, and the set RsW_a^s of the relations between the bits of $\nabla^+ w$, $\nabla^+ z$, and $\nabla^+ u$. Let RsW_s denote the set of relations between the bits of $\Delta^{\pm} w$ that appear in RsW_s^f and RsW_s^s simultaneously, and RsW_a denote the set of relations between the bits of $\nabla^+ w$ that appear in RsW_a^f and RsW_a^s simultaneously. Then according to the condition of the theorem, we know that $|RsW_s| + |RsW_a| = N$.

Let $I = \{i | w[i] = 1, 0 \le i < |w|\}$. Since each relation in RsW_s is of the form

$$\prod_{i_{sub} \in I_{sub}} \Delta^{\pm} w[i_{sub}] = 1 \quad or \quad \prod_{i_{sub} \in I_{sub}} \Delta^{\pm} w[i_{sub}] = -1,$$

where $I_{sub} \subseteq I$, we get that

$$P(RsW_s Hold) = 2^{-|RsW_s|},$$

where $RsW_s Hold$ denotes that the relations in RsW_s are held simultaneously.

Let $J = \{j | w[j] = 0, 0 \le j < |w|\}$. Since each relation in RsW_a is of the form

$$\bigoplus_{j_{sub} \in J_{sub}} \nabla^{+} w[j_{sub}] \gg 1 = 0 \quad or \quad \bigoplus_{j_{sub} \in J_{sub}} \nabla^{+} w[j_{sub}] \gg 1 = 1,$$

where $J_{sub} \subseteq J$, we get that

$$P(RsW_a Hold) = 2^{-|RsW_a|},$$

where $RsW_a Hold$ denotes that the relations in RsW_a are held simultaneously.

Let RsW denote the set of relations between the bits of $\Delta^{\pm} w$ and $\nabla^{+} w$ that appear in the two modular additions simultaneously. Then, we have

$$P(RsWHold) = P(RsW_s Hold) \times P(RsW_a Hold) = 2^{-(|RsW_s| + |RsW_a|)} = 2^{-N},$$

where $RsWHold$ denotes that the relations in RsW are held simultaneously. Furthermore, we have

$$\begin{gathered} P((\Delta x, \Delta y) \mapsto (\Delta w, \Delta z) \mapsto \Delta u) = P(A) \cdot P(B | RsWHold) \\ = P(A) \cdot \frac{P(B, RsWHold)}{P(RsWHold)} = P(A) \cdot \frac{P(B)}{P(RsWHold)} = P(A) \cdot P(B) \cdot 2^{N}. \end{gathered}$$

The case of two parallel modular additions that share an identical input branch can be proved similarly. Then we complete the proof. □

A.3 Proof of Theorem 4

Proof. (i) **For $\boldsymbol{\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] = 0}$**, let $\mathcal{S}^{\pm}$ denote the set of signed differential triples $(\Delta^{\pm} x, \Delta^{\pm} y \mapsto \Delta^{\pm} z)$ corresponding to the XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$. According to Definition 3, we have

$$|\mathcal{S}^{\pm}| = 2^{wt(\Delta x) + wt(\Delta y) + wt(\Delta z)}.$$

Let $\mathcal{S}^{+}$ denote the set of additive differential triples $(\Delta^{+} x, \Delta^{+} y \mapsto \Delta^{+} z)$ corresponding to the XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$. According to Equation (3), a signed difference corresponds to exactly one additive difference. Moreover, since $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] = 0$, we have

$$|\mathcal{S}^{+}| = 2^{wt(\Delta x) + wt(\Delta y) + wt(\Delta z)}.$$

Since addition modulo 2^n is a linear operation for additive differences, an additive differential triplet $(\Delta^+x, \Delta^+y \mapsto \Delta^+z)$ is valid iff

$$\Delta^+x \boxplus \Delta^+y = \Delta^+z \mod 2^n.$$

Therefore, according to Equation (3), an additive differential triplet $(\Delta^+x, \Delta^+y \mapsto \Delta^+z)$ is valid iff

$$\Delta^\pm x \boxplus \Delta^\pm y = \Delta^\pm z \mod 2^n.$$

Let

$$\mathcal{SS}^\pm = \{(\Delta^\pm x, \Delta^\pm y, \Delta^\pm z) | \Delta^\pm x \boxplus \Delta^\pm y = \Delta^\pm z \mod 2^n\}$$

and

$$\mathcal{SS}^+ = \{(\Delta^+ x, \Delta^+ y, \Delta^+ z) | \Delta^\pm x \boxplus \Delta^\pm y = \Delta^\pm z \mod 2^n\}.$$

According to Equation (3) and the condition of $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] = 0$, we have

$$|\mathcal{SS}^\pm| = |\mathcal{SS}^+|.$$

Therefore, the probability $P(\Delta x, \Delta y \mapsto \Delta z)$ can be calculated as

$$\begin{aligned} P(\Delta x, \Delta y \mapsto \Delta z) &= \frac{|\mathcal{SS}^+|}{|\mathcal{S}^+|} \\ &= \frac{\#\{(\Delta^\pm x, \Delta^\pm y, \Delta^\pm z) | \Delta^\pm x \boxplus \Delta^\pm y = \Delta^\pm z \mod 2^n\}}{2^{wt(\Delta x)+wt(\Delta y)+wt(\Delta z)}}. \end{aligned}$$

(ii) **For$\boldsymbol{\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] \neq 0}$**, let $\mathcal{S}^\pm$ denote the set of signed differential triples $(\Delta^\pm x, \Delta^\pm y \mapsto \Delta^\pm z)$ corresponding to the XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$. According to Definition 3, we have

$$|\mathcal{S}^\pm| = 2^{wt(\Delta x)+wt(\Delta y)+wt(\Delta z)}.$$

Let $\mathcal{S}^+$ denote the set of additive differential triples $(\Delta^+x, \Delta^+y \mapsto \Delta^+z)$ corresponding to the XOR differential triplet $(\Delta x, \Delta y \mapsto \Delta z)$. According to Equation (3), a signed difference corresponds to exactly one additive difference. However, since $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] \neq 0$, we have

$$|\mathcal{S}^+| < 2^{wt(\Delta x)+wt(\Delta y)+wt(\Delta z)}.$$

Let $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] = i$, where $1 \leq i \leq 3$. According to Equation (3), we have

$$|\mathcal{S}^+| = 2^{wt(\Delta x)+wt(\Delta y)+wt(\Delta z)-i}.$$

Since addition modulo 2^n is a linear operation for additive differences, an additive differential triplet $(\Delta^+x, \Delta^+y \mapsto \Delta^+z)$ is valid iff

$$\Delta^+x \boxplus \Delta^+y = \Delta^+z \mod 2^n.$$

Therefore, according to Equation (3), an additive differential triplet ($\Delta^+ x$, $\Delta^+ y \mapsto \Delta^+ z$) is valid iff

$$\Delta^{\pm} x \boxplus \Delta^{\pm} y = \Delta^{\pm} z \mod 2^n.$$

Let

$$\mathcal{SS}^{\pm} = \{(\Delta^{\pm} x, \Delta^{\pm} y, \Delta^{\pm} z) | \Delta^{\pm} x \boxplus \Delta^{\pm} y = \Delta^{\pm} z \mod 2^n\}$$

and

$$\mathcal{SS}^{+} = \{(\Delta^{+} x, \Delta^{+} y, \Delta^{+} z) | \Delta^{\pm} x \boxplus \Delta^{\pm} y = \Delta^{\pm} z \mod 2^n\}.$$

According to Equation (3) and the condition of $\Delta x[n-1] + \Delta y[n-1] + \Delta z[n-1] \neq 0$, we have

$$|\mathcal{SS}^{\pm}| = 2^i \times |\mathcal{SS}^{+}|.$$

Moreover, since

$$|\mathcal{S}^{\pm}| = 2^i \times |\mathcal{S}^{+}|,$$

the probability $P(\Delta x, \Delta y \mapsto \Delta z)$ can be calculated as

$$\begin{aligned} P(\Delta x, \Delta y \mapsto \Delta z) &= \frac{|\mathcal{SS}^{+}|}{|\mathcal{S}^{+}|} \\ &= \frac{\#\{(\Delta^{\pm} x, \Delta^{\pm} y, \Delta^{\pm} z) | \Delta^{\pm} x \boxplus \Delta^{\pm} y = \Delta^{\pm} z \mod 2^n\}}{2^{wt(\Delta x)+wt(\Delta y)+wt(\Delta z)}}. \end{aligned}$$

Then we complete the proof. □

B Descriptions of CHAM, Alzette, XTEA, Salsa20, and Chaskey

At ICISC'17, Koo et al. [18] presented a family of lightweight block ciphers CHAM. It adopts a 4-branch generalized Feistel structure. Each cipher is denoted by CHAM-n/k, where n and k are the block size and key size, respectively. For CHAM-64/128 (resp. CHAM-128/128 and CHAM-128/256), the word size of each branch is 16 (resp. 32 and 32), and the number of rounds is 80 (resp. 80 and 96).

By applying r iterations of the key-dependent round function, CHAM-n/k encrypts a plaintext $X_0 = X_0^0 \parallel X_0^1 \parallel X_0^2 \parallel X_0^3$ to a ciphertext $X_r = X_r^0 \parallel X_r^1 \parallel X_r^2 \parallel X_r^3$. For $0 \le i < r$, the i-th round outputs

$$X_{i+1}^3 \leftarrow ((X_i^0 \oplus i) \boxplus ((X_i^1 \lll r_a) \oplus RK[i \;\; mod\; 2k/w])) \lll r_b,\;\; X_{i+1}^j \leftarrow X_i^{j+1},$$

where $0 \le j \le 2$, and $RK[i \;\; mod\; 2k/w]$ is the round key. When i is even, $(r_a, r_b) = (8, 1)$, otherwise $(r_a, r_b) = (1, 8)$. The round function of CHAM is shown in Fig. 3.

At CRYPTO'20, Beierle et al. [7] proposed a 64-bit ARX-based S-box called Alzette. It is a 4-round SPECK-like structure with 2-branch and is parameterized

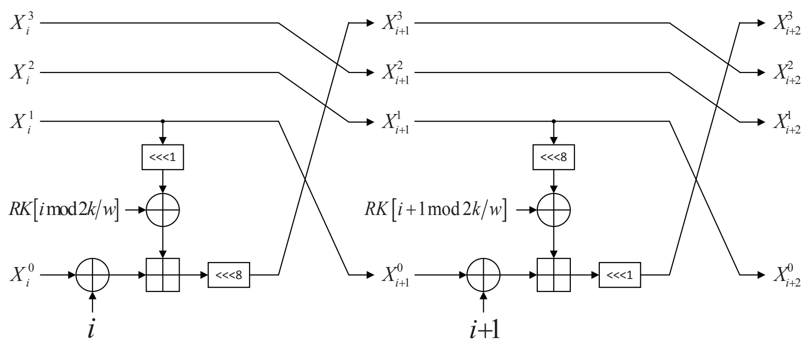

Fig. 3. The round function of CHAM

by an arbitrary constant $c \in \mathbb{F}_2^{32}$. The algorithm evaluating this permutation is given in Algorithm 5 and depicted in Fig. 4.

Algorithm 5 The Alzette instance A_c

```
Input: $(x, y) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$
Output: $(u, v) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$
 1: $x \leftarrow x \boxplus (y \ggg 31)$
 2: $y \leftarrow y \oplus (x \ggg 24)$
 3: $x \leftarrow x \oplus c$
 4: $x \leftarrow x \boxplus (y \ggg 17)$
 5: $y \leftarrow y \oplus (x \ggg 17)$
 6: $x \leftarrow x \oplus c$
 7: $x \leftarrow x \boxplus (y \ggg 0)$
 8: $y \leftarrow y \oplus (x \ggg 31)$
 9: $x \leftarrow x \oplus c$
10: $x \leftarrow x \boxplus (y \ggg 24)$
11: $v \leftarrow y \oplus (x \ggg 16)$
12: $u \leftarrow x \oplus c$
13: return $(u, v)$
```

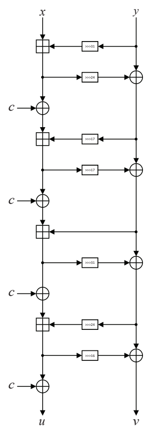

Fig. 4. The Alzette instance A_c

In 1997, Needham et al. [28] proposed a block cipher XTEA which is an extended version of TEA [35]. It has a Feistel structure composed of 64 rounds. Each round operates on 64-bit blocks using a 128-bit key. Let (L_i, R_i) be the input of the i-th round, and the output of the i-th round is computed as follows:

$$R_{i+1} = L_i \boxplus (F(i, \delta, k) \oplus (R_i \boxplus ((R_i \ll 4) \oplus (R_i \gg 5)))),\ L_{i+1} = R_i,$$

where $F(i, \delta, k)$ is the round key and $\delta = 0x9e3779b9$. The round function of XTEA is shown in Fig. 5.

Salsa20 [9] is a stream cipher designed by Bernstein in 2005 as a candidate for the eSTREAM competition. The original proposal was for 20 rounds. The 12-round variant of Salsa20, Salsa20/12 was accepted into the final eSTREAM soft-

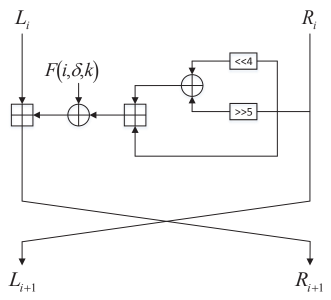

Fig. 5. The round function of XTEA

ware portfolio. A Salsa20 round consists of four parallel `quarterround` functions that operate on 32-bit words. Let $(a_0^0, a_1^0, a_2^0, a_3^0)$ be the input of the `quarterround` function of Salsa20, and the output of the `quarterround` function of Salsa20 is computed as follows:

$$\begin{cases} a_1^1 = a_1^0 \oplus ((a_0^0 \boxplus a_3^0) \lll 7), \\ a_2^1 = a_2^0 \oplus ((a_0^0 \boxplus a_1^1) \lll 9), \\ a_3^1 = a_3^0 \oplus ((a_1^1 \boxplus a_2^1) \lll 13), \\ a_0^1 = a_0^0 \oplus ((a_2^1 \boxplus a_3^1) \lll 18). \end{cases}$$

The `quarterround` function of Salsa20 is shown in Fig. 6.

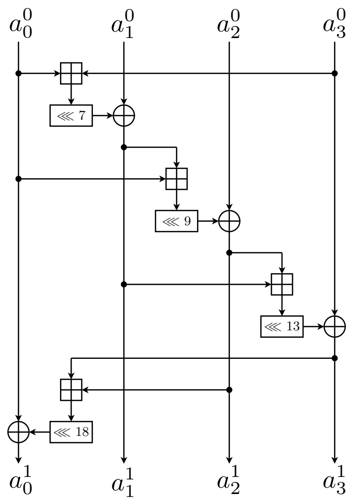

Fig. 6. The `quarterround` function of Salsa20

Chaskey [25] is a lightweight MAC algorithm whose underlying primitive is an ARX-based permutation in an Even-Mansour construction. The permutation operates on four 32-bit words and employs 12 rounds of the form as depicted in Fig. 7.

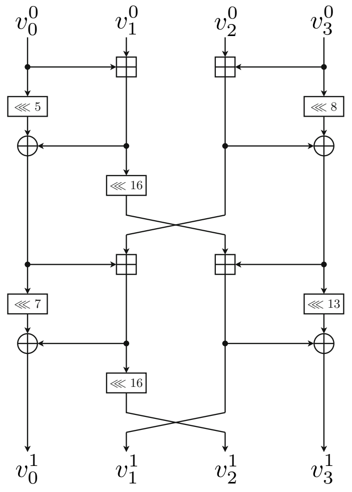

Fig. 7. The round function of Chaskey