A new method for Searching Optimal Differential and Linear Trails in ARX Ciphers

Zhengbin Liu[1,3], Yongqiang Li[1,3⋆], Lin Jiao[2], and Mingsheng Wang[1,3]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] State Key Laboratory of Cryptology, Beijing, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
yongq.lee@gmail.com, jiaolin_jl@126.com

Abstract. In this paper, we propose an automatic tool to search for optimal differential and linear trails in ARX ciphers. It's shown that a modulo addition can be divided into sequential small modulo additions with carry bit, which turns an ARX cipher into an S-box-like cipher. From this insight, we introduce the concepts of carry-bit-dependent difference distribution table (CDDT) and carry-bit-dependent linear approximation table (CLAT). Based on them, we give efficient methods to trace all possible output differences and linear masks of a big modulo addition, with returning their differential probabilities and linear correlations simultaneously. Then an adapted Matsui's algorithm is introduced, which can find the optimal differential and linear trails in ARX ciphers.
Besides, the superiority of our tool's potency is also confirmed by experimental results for round-reduced versions of HIGHT and SPECK. More specifically, we find the optimal differential trails for up to 10 rounds of HIGHT, reported for the first time. We also find the optimal differential trails for 10, 12, 16, 8 and 8 rounds of SPECK32/48/64/96/128, and report the provably optimal differential trails for SPECK48 and SPECK64 for the first time. The optimal linear trails for up to 9 rounds of HIGHT are reported for the first time, and the optimal linear trails for 22, 13, 15, 9 and 9 rounds of SPECK32/48/64/96/128 are also found respectively. These results evaluate the security of HIGHT and SPECK against differential and linear cryptanalysis. Also, our tool is useful to estimate the security in the design of ARX ciphers.

Keywords: automatic search, differential trail, linear trail, ARX, HIGHT, SPECK

1 Introduction

Many cryptographic primitives have employed a combination of modular addition, bit rotation and XOR (ARX). The advantage of these designs is that they are very simple, efficient and easy to implement in software and hardware. Modular addition is a nonlinear operation, and bit rotation and XOR are linear. By combining these simple operations, lots of ARX algorithms have been proposed since 1980s. Here are some notable examples: the block ciphers FEAL [38], TEA [43], XTEA [32], RC5 [35], HIGHT [21], SPECK [5], LEA [20] and SPARX [15], the stream cipher Salsa20 [6], the hash functions from MD and SHA families [34], Skein [17], BLAKE [2], and the MAC algorithm Chaskey [29].

Differential cryptanalysis [7] and linear cryptanalysis [26] are the two most powerful techniques in the cryptanalysis of symmetric cryptographic primitives. For modern ciphers, security against differential and linear cryptanalysis is a major design criterion. As for S-box based ciphers, there exist a variety of automatic search algorithms to evaluate their security against differential and linear cryptanalysis, see [1,3,4,8,9,13,27,31,33,41] for details. This is because S-box based ciphers use typical S-boxes operating on 8 or 4-bit words, and it is easy to evaluate the differential and linear property of an S-box by computing its difference distribution table (DDT) and linear approximation table (LAT). As for ARX ciphers, modular addition is the source of nonlinearity. Constructing a DDT and LAT for addition of n-bit words requires 2^{3n} bytes of memory. This is infeasible for a typical word size of 32 bits.

Although some automatic search algorithms have been proposed for differential trails in the MD and SHA families of hash functions [14,16,18,22,23,28,36,40], none of them were applied to ARX primitives except hash functions. To fill this gap, Biryukov et al. [11] proposed a threshold

⋆ Corresponding Author.

search algorithm and introduced the concept of partial difference distribution table (pDDT), which contains a collection of differences whose probabilities are beyond a fixed threshold. Although it is impossible to compute a full DDT of addition modulo 2^n, it is still possible to compute a pDDT efficiently. Using the pDDT, they firstly extended Matsui's algorithm to ARX ciphers and proposed a threshold search algorithm. They applied their algorithm to the block ciphers TEA, XTEA, SPECK and SIMON, and got some improved differential trails [10,11]. However, their algorithm may not obtain the optimal differential trails, because they use heuristics in order to find high-probability trails.

In [12], Biryukov et al. adapted Matsui's algorithm and proposed an automatic search algorithm for the best trails in ARX ciphers. They found some best differential trails for round-reduced versions of SPECK. Since the complexity of linear search is much higher than differential search, it is infeasible to search for the best linear trails for other versions of SPECK except SPECK32. As for SPECK32, they only found the best linear trails up to 6 rounds. Yao et al. [44] also adapted Matsui's algorithm to search for the optimal linear trails for SPECK. They applied Wallén's algorithm [42] to Matsui's branch-and-bound framework, and found the optimal linear trails for full rounds of SPECK32, and short linear trails for other versions of SPECK.

Fu et al. [19] proposed a MILP-based automatic search algorithm, and got some improved differential and linear trails for SPECK. Although MILP-based algorithm is able to find the optimal trails, its running time is not well understood. Mouha et al. [30] translated the problem of finding optimal differential trails into the Boolean satisfiability problem (SAT), and then used a SAT solver to solve it. They applied the SAT solver approach to Salsa20 for the first time, and found a 3-round optimal differential trail. Song et al. [39] adopted Mouha et al.'s framework to search the differential trails in ARX ciphers by constructing long trails from short ones, and obtained improved differential trails of SPECK and LEA. Liu et al. [25] also applied the SAT solver approach to search the linear trails in ARX ciphers, and found the optimal linear trails for round-reduced versions of SPECK and Chaskey.

Biryukov et al.'s algorithm, MILP-based algorithm and the SAT solver approach are all general automatic search algorithms in ARX ciphers, and they can all find the optimal trails on round-reduced variants of SPECK. However, the high time complexity makes Biryukov et al.'s algorithm and MILP-based algorithm hard for other ARX ciphers except SPECK. As for the SAT solver approach, it needs to write equations of addition, rotation and XOR in the cipher, and then solve equations with the SAT solver. This method may encounter difficulties when applied to more rounds of ARX ciphers or ARX ciphers with more logic operations, since more variables and equations appear. Therefore, designing an automatic search algorithm in ARX ciphers that can give optimal differential and linear trails of as many rounds as possible is an important problem that needs further study.

Our Contributions. We investigate the above problem in the present paper, and our main contributions are summarized as follows.

1. Firstly, we propose the concepts of the carry-bit-dependent difference distribution table (CDDT) and the carry-bit-dependent linear approximation table (CLAT), which are very helpful for computing the differential probability and linear correlation of modular addition respectively. The core idea is that addition modulo 2^n can be divided into partial sums on small bit words such as 8 or 4-bit words, and they are correlated by carry bits. Then the difference distribution table (resp. linear approximation tables) of partial sums and can be pre-constructed, which we called CDDTs (resp. CLATs), and the relationship of differential probability (resp. absolute correlations) of addition modulo 2^n and CDDTs (resp. CLATs) is characterized. To improve the efficiency, we also give the constructions of CDDTs and CLATs with Lipmaa-Moriai's algorithm [24] and Schulte-Geers's result [37] respectively.
2. Secondly, we propose a general automatic search algorithm for optimal differential and linear trails in ARX ciphers with CDDTs and CLATs. Our algorithm is based on Matsui's branch-and-bound algorithm [27], and it looks up CDDTs (resp. CLATs) to get all possible output differences (resp. linear masks) and their probabilities (resp. absolute correlations) when computing the differential probability (resp. absolute correlation) of addition modulo 2^n. Furthermore, we sort the CDDTs (resp. CLATs) according to the differential probability (resp. absolute correlation). Then we can always get the output difference (reps. output mask) with highest differential probability (resp. absolute correlation), and break the unnecessary branches as soon as possible.
3. At last, we apply our algorithm to round-reduced versions of block ciphers HIGHT and SPECK. Our algorithm is able to find the optimal differential trails for round-reduced versions HIGHT

and SPECK. For HIGHT, a 10-round optimal differential trail with probability 2^{-38} is found, which is reported for the first time. For SPECK with block size 32, 48, 64, 96 and 128 bits, we find the optimal differential trails on 10, 12, 16, 8 and 8 rounds with probability 2^{-34}, 2^{-49}, 2^{-70}, 2^{-30} and 2^{-30} respectively. Our results cover more rounds than the results given by Biryukov et al. [12] for SPECK48, SPECK64, SPECK96 and SPECK128, and the same as theirs for SPECK32. Meanwhile we report the provably optimal differential trails for SPECK48 and SPECK64 for the first time.

4. Our algorithm can also find the optimal linear trails for round-reduced versions of HIGHT and SPECK. For HIGHT, we find for the first time the optimal linear trail for reduced number of rounds, that is a 9-round linear trail with correlation 2^{-15}. As for SPECK with block size 32, 48, 64, 96 and 128 bits, we find the provably optimal linear trails on 22, 13, 15, 9 and 9 rounds respectively, which confirm the optimal linear trails given by Liu et al. [25].

The comparison of our identified differential and linear trails with those given by other algorithms is listed in Table 1. In this table, BA, MA and SA represent the abbreviation of Biryukov et al.'s algorithm [12], MILP-based algorithm [19] and SAT solver approach [25] respectively. DT and LT represent the abbreviation of differential trails and linear trails respectively. Although Fu et al.'s results are the same as ours for SPECK32, SPECK48 and SPECK64, and cover more rounds for SPECK96 and SPECK128, they can't ensure their results are optimal, since they apply a splicing heuristic to find better differential and linear trails than existing ones [19].

Table 1. The number of covered rounds of differential and linear trails for HIGHT and SPECK.

	BA[12]		MA[19]		SA[25]		our algorithm	
Cipher	DT	LT	DT	LT	DT	LT	DT	LT
HIGHT	–	–	–	–	–	–	10	9
SPECK32	10	6	9	9	–	22	10	22
SPECK48	9	–	11	10	–	11	12	13
SPECK64	8	–	15	13	–	13	16	15
SPECK96	7	–	16	15	–	9	8	9
SPECK128	6	–	19	16	–	9	8	9
	optimal		heuristic		optimal		optimal	

Outline. The paper is organized as follows. In Section 2, we introduce the concept of carry-bit-dependent S-box, carry-bit-dependent difference distribution table (CDDT) and carry-bit-dependent linear approximation table (CLAT). A method for their computations is also given in this section. In Section 3, an improved method for constructing CDDTs and CLATs is given. In Section 4, we propose an automatic search algorithm for optimal differential and linear trails in ARX ciphers, which is an extension of Matsui's algorithm using CDDTs and CLATs. In Section 5,we apply the proposed algorithm to block ciphers HIGHT and SPECK, and show the experimental results. A short conclusion is given in Section 6.

Notations used in the present paper are defined in Table 2.

2 Carry-bit-dependent S-box, Difference Distribution Table and Linear Approximation Table

Throughout this section, a n-bit vector $x = (x_{n-1}, \dots, x_1, x_0) \in \mathbb{F}_2^n$ means a binary representation of an integer $\sum_{i=0}^{n-1} x_i 2^i$ in $\mathbb{Z}_{2^n}$. First, we introduce the definition of addition modulo 2^n, and then give a method to compute the differential probability and linear correlation of addition.

2.1 Addition and Carry-bit-dependent S-box

Definition 1 (Addition modulo 2^n [24]). *Let $x, y \in \mathbb{F}_2^n$. Then*

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

Table 2. Notations

Notation	Description
$x \boxplus y$	addition of x and y modulo 2^n
$\overline{x}$	bitwise NOT of x
$x[i:j]$	the sequence of bits $x_i, x_{i-1}, \ldots, x_j$
$x \oplus y$	bitwise exclusive OR (XOR) of x and y
$x \wedge y$	bitwise AND of x and y
$x \vee y$	bitwise OR of x and y
$x \ll r$	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
$x \| y$	concatenation of bit strings x and y
$\mathrm{wt}(x)$	the hamming weight of x
Δx	XOR difference of x and x' : $\Delta x = x \oplus x'$
$eq(x,y,z)$	$(\overline{x} \oplus y) \wedge (\overline{x} \oplus z)$
$mask(n)$	$2^n - 1$
$x \cdot y$	dot product of x and y : $x \cdot y = \bigoplus_{i=0}^{n-1} x_i y_i$
$x \preceq y$	$x_i \leq y_i, \forall i \in \{0, \ldots, n-1\}$

where $carry(x,y) = (c_{n-1}, \ldots, c_1, c_0) \in \mathbb{F}_2^n$ denotes the carry bit vector of $x \boxplus y$. It is defined recursively as follows:

$$\begin{aligned} &c_0 = 0 \\ &c_{i+1} = (x_i \wedge y_i) \oplus (x_i \wedge c_i) \oplus (y_i \wedge c_i) \text{ for } 0 \leq i \leq n-2. \end{aligned}$$

Next, we give another representation of vectors in $\mathbb{F}_2^n$, which is helpful for charactering the differential probability of addition modulo 2^n from a new viewpoint.

Suppose $n = mt$. For a n-bit vector $x \in \mathbb{F}_2^n$, we define $X_k = x[(k+1)t - 1 : kt]$. Then it holds

$$x = (X_{m-1}, \ldots, X_0) = \sum_{k=0}^{m-1} X_k 2^{kt}.$$

For two t-bit vectors $X, Y \in \mathbb{F}_2^t$ and a bit $e \in \mathbb{F}_2$, $X \boxplus Y \boxplus e$ denotes

$$X \boxplus Y \boxplus (0, \ldots, 0, e),$$

which is equal to $X \oplus Y \oplus carry^*(X,Y)$, and $carry^*(X,Y) = (c^*_{t-1}, \ldots, c^*_0)$ is computed as follows

$$\begin{aligned} &c^*_0 = e \\ &c^*_{i+1} = (x_i \wedge y_i) \oplus (x_i \wedge c^*_i) \oplus (y_i \wedge c^*_i) \text{ for } 0 \leq i \leq t-1. \end{aligned} \quad (1)$$

It should be noticed that c^*_t can also be computed according to the above formula.

Therefore, it is easy to check that addition modulo 2^n can be written as follows:

$$\begin{aligned} x \boxplus y &= x \oplus y \oplus carry(x,y) \\ &= \sum_{k=0}^{m-1} (X_k \oplus Y_k \oplus C_k) 2^{kt} \\ &= \sum_{k=0}^{m-1} (X_k \boxplus Y_k \boxplus c_{kt}) 2^{kt}, \end{aligned} \quad (2)$$

where $C_k = carry(x,y)[(k+1)t - 1 : kt]$, and c_{kt} is the kt-th bit of $carry(x,y)$, which is the carry bit vector of $x \boxplus y$.

Note that $c_0 = 0$ and for $k \geq 0$, $c_{(k+1)t}$ can be computed from $X_k \boxplus Y_k \boxplus c_{kt}$ by Formula (1). Then the computation of mt-bit vectors addition modulo 2^{mt} can be divided into m portions of t-bit vectors addition modulo 2^t, and the $(k+1)$-th portion is correlated to the k-th portion by the carry bit $c_{(k+1)t}$.

We can build truth value tables of addition modulo 2^t with different correlated values (carry bits), and these truth value tables are called **carry-bit-dependent S-boxes**. We can also build difference distribution tables of addition modulo 2^t with different correlated values, and these difference distribution tables are called **carry-bit-dependent difference distribution tables** (CDDTs). Similarly, linear approximation tables of addition modulo 2^t with different correlated values can be built, and these linear approximation tables are called **carry-bit-dependent linear approximation tables** (CLATs). The relationship of addition modulo 2^{mt} and m carry-bit-dependent S-boxes is depicted in Fig 1.

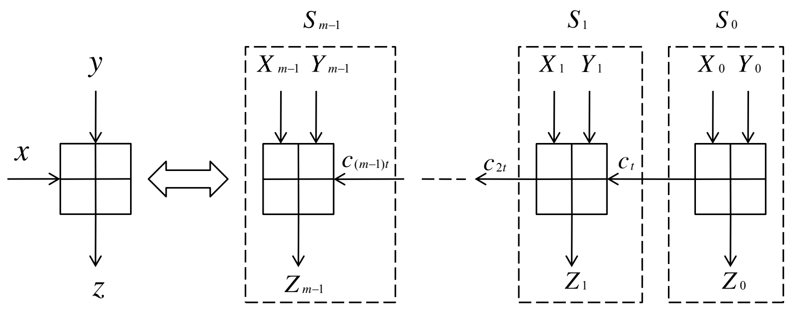

Fig. 1. The carry-bit-dependent S-boxes

The size of CDDTs (resp. CLATs) can be any integer only if it can divide n, that is, m and t can be any integer only if $mt = n$. By experiment we find that the more bits it has, the faster the search algorithm runs and the more memory it requires. Taking tradeoff of time and memory, we use 8-bit CDDTs (resp. CLATs) in our applications. For CDDTs, it calls for

$$\begin{aligned}&(2\ \texttt{types of S-boxes}) \times (2^3\ \texttt{different carry values})\times\\&\quad(2^{2\times 8+8}\ \texttt{input-output differential values})\times\\&\qquad(1\ \texttt{byte to store the logarithm of differential probability}) = 2^{28}\ \texttt{bytes}.\end{aligned}$$

For CLATs, it calls for

$$\begin{aligned}&(2\ \texttt{different carry values}) \times \quad (2^{8+2\times 8}\ \texttt{input-output masking values})\times\\&\qquad(1\ \texttt{byte to store the logarithm of correlation coefficient}) = 2^{25}\ \texttt{bytes}.\end{aligned}$$

Next, we introduce the idea of computing the differential probability of addition with CDDTs and the linear correlation of addition with CLATs.

2.2 Computing Differential Probability of Addition with CDDTs

The differential probability of addition modulo 2^n is defined as follows.

Definition 2 (Differential Probability of Addition [24]). *A differential of addition modulo 2^n is defined as a triplet of two input differences and one output difference, which is denoted as $(\alpha, \beta \mapsto \gamma)$, where $\alpha, \beta, \gamma \in \mathbb{F}_2^n$. The differential probability of addition modulo 2^n is defined as*

$$P(\alpha, \beta \mapsto \gamma) = P_{x,y}[(x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma].$$

Hereafter this section, $x, y, \alpha, \beta, \gamma \in \mathbb{F}_2^n$ always mean n-bit vectors. Let $z = x \boxplus y$, and α, β, γ be the XOR difference of x, y, z respectively. This means

$$x' = x \oplus \alpha, y' = y \oplus \beta, z' = x' \boxplus y' \text{ and } z' = z \oplus \gamma.$$

The carry bit vectors of $x \boxplus y$ and $x' \boxplus y'$ are denoted simply by

$$c = (c_{n-1}, \ldots, c_1, c_0) \text{ and } c' = (c'_{n-1}, \ldots, c'_1, c'_0)$$

respectively. For $0 \le k \le m-1$, let

$$A_k = \alpha[(k+1)t - 1 : kt], B_k = \beta[(k+1)t - 1 : kt], \Gamma_k = \gamma[(k+1)t - 1 : kt].$$

Then it holds

$$\alpha = \sum_{k=0}^{m-1} A_k 2^{kt}, \beta = \sum_{k=0}^{m-1} B_k 2^{kt}, \text{ and } \gamma = \sum_{k=0}^{m-1} \Gamma_k 2^{kt}.$$

For $g = (g_1, g_0), h = (h_1, h_0) \in \mathbb{F}_2^2$, $A, B, \Gamma \in \mathbb{F}_2^t$, let

$$S_g(A, B \mapsto \Gamma) = \{(X, Y) \mid (X \boxplus Y \boxplus g_1) \oplus (X' \boxplus Y' \boxplus g_0) = \Gamma\},$$

and

$$S_g^h(A, B \mapsto \Gamma) = \{(X, Y) \mid (X \boxplus Y \boxplus g_1) \oplus (X' \boxplus Y' \boxplus g_0) = \Gamma, c_t^* = h_1, c_t'^* = h_0\},$$

where

$$X' = X \oplus A, Y' = Y \oplus B,$$

and c_t^*, $c_t'^*$ are computed according to Formula (1). Furthermore, let

$$PS_g(A, B \mapsto \Gamma) = 2^{-2t}|S_g(A, B \mapsto \Gamma)|,$$

and

$$PS_g^h(A, B \mapsto \Gamma) = 2^{-2t}|S_g^h(A, B \mapsto \Gamma)|.$$

Then we have the following result to compute the differential probability of addition.

Theorem 1. *Let $n = mt$, $x, y, \alpha, \beta, \gamma \in \mathbb{F}_2^n$, $z = x \boxplus y$, and α, β, γ are the XOR differences of x, y, z respectively. Then*

$$P(\alpha, \beta \mapsto \gamma) = \sum_{i=0, g_i \in T_i}^{m-1} \prod_{k=0}^{m-2} PS_{g_k}^{g_{k+1}}(A_k, B_k \mapsto \Gamma_k) PS_{g_{m-1}}(A_{m-1}, B_{m-1} \mapsto \Gamma_{m-1}),$$

where $T_0 = \{(0, 0)\}$ and $T_i = \mathbb{F}_2^2$ for $i = 1, \ldots, m-1$.

Proof. First, we have

$$\begin{aligned}
&\{(x, y) \mid (x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma\} \\
=& \bigcup_{i=1, g_i \in \mathbb{F}_2^2}^{m-1} \{(x, y) \mid (x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma, (c_{it}, c'_{it}) = g_i \text{ for } 1 \leq i \leq m-1\} \\
=& \bigcup_{i=0, g_i \in T_i}^{m-1} \{(x, y) \mid (x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma, (c_{it}, c'_{it}) = g_i \text{ for } 0 \leq i \leq m-1\},
\end{aligned}$$

where $T_0 = \{(0, 0)\}$ and $T_i = \mathbb{F}_2^2$ for $1 \leq i \leq m-1$. Furthermore, according to formula (2), we have

$$\begin{aligned}
\sum_{i=0}^{m-1} \Gamma_k 2^{kt} &= \gamma = (x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) \\
&= \sum_{k=0}^{m-1} ((X_k \boxplus Y_k \boxplus c_{kt}) \oplus ((X_k \oplus A_k) \boxplus (Y_k \oplus B_k) \boxplus c'_{kt}))2^{kt},
\end{aligned}$$

which is equivalent to that for $k = 0, \ldots, m-1$, it holds

$$(X_k \boxplus Y_k \boxplus c_{kt}) \oplus (X'_k \boxplus Y'_k \boxplus c'_{kt}) = \Gamma_k,$$

where $X'_k = X_k \oplus A_k$, $Y'_k = Y_k \oplus B_k$, c and c' are the carry bit vectors of $x \boxplus y$ and $(x \oplus \alpha) \boxplus (y \oplus \beta)$ respectively. Therefore,

$$
\begin{aligned}
& P(\alpha, \beta \mapsto \gamma) \\
&= 2^{-2n} |\{(x,y) \mid (x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma\}| \\
&= \sum_{i=0, g_i \in T_i}^{m-1} 2^{-2n} |\{(x,y) \mid (x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma, (c_{it}, c'_{it}) = g_i \text{ for } 0 \le i \le m-1\}| \\
&= \sum_{i=0, g_i \in T_i}^{m-1} \prod_{k=0}^{m-2} 2^{-2t} |\{(X_k, Y_k) \mid (X_k \boxplus Y_k \boxplus g_k[1]) \oplus (X'_k \boxplus Y'_k \boxplus g_k[0]) = \mathit{\Gamma}_k, (c^*_t, c'^*_t) = g_{k+1}\}| \\
&\quad \times 2^{-2t} |\{(X_{m-1}, Y_{m-1}) \mid (X_{m-1} \boxplus Y_{m-1} \boxplus g_{m-1}[1]) \oplus (X'_{m-1} \boxplus Y'_{m-1} \boxplus g_{m-1}[0]) = \mathit{\Gamma}_{m-1}\}| \\
&= \sum_{i=0, g_i \in T_i}^{m-1} \prod_{k=0}^{m-2} 2^{-2t} |S_{g_k}^{g_{k+1}}(A_k, B_k \mapsto \mathit{\Gamma}_k)| \times 2^{-2t} |S_{g_{m-1}}(A_{m-1}, B_{m-1} \mapsto \mathit{\Gamma}_{m-1})| \\
&= \sum_{i=0, g_i \in T_i}^{m-1} \prod_{k=0}^{m-2} PS_{g_k}^{g_{k+1}}(A_k, B_k \mapsto \mathit{\Gamma}_k) PS_{g_{m-1}}(A_{m-1}, B_{m-1} \mapsto \mathit{\Gamma}_{m-1}),
\end{aligned}
$$

where c^*_t, c'^*_t are computed according to Formula (1). Then we complete the proof.

As for computing the differential probability of addition modulo 2^n, it only needs to build 16 tables of $PS_g^h(A, B \mapsto \mathit{\Gamma})$ for $g, h \in \mathbb{F}_2^2$, and 4 tables of $PS_g(A, B \mapsto \mathit{\Gamma})$ for $g \in \mathbb{F}_2^2$. Then the differential probability $P(\alpha, \beta \mapsto \gamma)$, where $\alpha, \beta, \gamma \in \mathbb{F}_2^n$, can be computed by looking up the above tables according to Theorem 1.

We give a toy example to illustrate how to compute the differential probability of addition with Theorem 1 in Appendix A.

2.3 Computing Linear Correlation of Addition with CLATs

The linear correlation of addition modulo 2^n is defined as follows.

Definition 3 (Linear Correlation of Addition [37]). *Let $x, y, z \in \mathbb{F}_2^n$, and $z = x \boxplus y$. The linear correlation of addition modulo 2^n is defined as the correlation of the linear approximation $\nu \cdot x \oplus \omega \cdot y = \mu \cdot z$, where μ, ν and ω are n-bit linear masks. It is computed as follows:*

$$C(\mu, \nu, \omega) = 2^{-2n} \sum_{x \in \mathbb{F}_2^n, y \in \mathbb{F}_2^n} (-1)^{\mu \cdot z \oplus \nu \cdot x \oplus \omega \cdot y}.$$

For $g, h \in \mathbb{F}_2$, let

$$S_g^h = \{(X, Y) \in (\mathbb{F}_2^t)^2 \mid c^*_t[X \boxplus Y \boxplus g] = h\},$$

where $c^*_t[X \boxplus Y \boxplus g]$ means the t-th carry bit of $X \boxplus Y \boxplus g$ computed according to Formula (1). For $U, V, W \in \mathbb{F}_2^t$, let

$$C_g^h(U, V, W) = 2^{-t} \sum_{X, Y \in S_g^h} (-1)^{U \cdot (X \boxplus Y \boxplus g) \oplus V \cdot X \oplus W \cdot Y}.$$

Let $S_g = S_g^0 \cup S_g^1$, and

$$C_g(U, V, W) = 2^{-t} \sum_{X, Y \in S_g} (-1)^{U \cdot (X \boxplus Y \boxplus g) \oplus V \cdot X \oplus W \cdot Y}.$$

Then we have the following result.

Theorem 2. *Let $n = mt$, $x, y, \mu, \nu, \omega \in \mathbb{F}_2^n$, $z = x \boxplus y$, and μ, ν, ω are the linear masks of z, x, y respectively. Then*

$$C(\mu, \nu, \omega) = \sum_{i=0, g_i \in T_i}^{m-1} \prod_{k=0}^{m-2} C_{g_k}^{g_{k+1}}(U_k, V_k, W_k) C_{g_{m-1}}(U_{m-1}, V_{m-1}, W_{m-1}),$$

where $T_0 = \{0\}$, and $T_i = \mathbb{F}_2$ for $1 \le i \le m-1$.

Proof. Note that

$$\{(x,y) \mid x,y \in \mathbb{F}_2^n\} = \bigcup_{i=1, g_i \in \mathbb{F}_2}^{m-1} \{(x,y) \mid x,y \in \mathbb{F}_2^n, c_{it} = g_i \text{ for } 1 \le i \le m-1\} = \bigcup_{i=0, g_i \in T_i}^{m-1} S_{g_0,\ldots,g_{m-1}},$$

where $S_{g_0,\ldots,g_{m-1}} = \{(x,y) \mid x,y \in \mathbb{F}_2^n, c_{it} = g_i \text{ for } 0 \le i \le m-1\}$. Therefore,

$$\begin{aligned} C(\mu,\nu,\omega) &= 2^{-2n} \sum_{x\in\mathbb{F}_2^n, y\in\mathbb{F}_2^n} (-1)^{\mu\cdot(x\boxplus y)\oplus\nu\cdot x\oplus\omega\cdot y} \\ &= 2^{-2n} \sum_{i=0, g_i\in T_i}^{m-1} \left(\sum_{x,y\in S_{g_0,\ldots,g_{m-1}}} (-1)^{\mu\cdot(x\boxplus y)\oplus\nu\cdot x\oplus\omega\cdot y} \right) \\ &= \sum_{i=0, g_i\in T_i}^{m-1} \prod_{k=0}^{m-2} 2^{-2t} \left(\sum_{X_k, Y_k \in S_{g_k}^{g_{k+1}}} (-1)^{U_k\cdot(X_k\boxplus Y_k\boxplus g_k)\oplus V_k\cdot X_k\oplus W_k\cdot Y_k} \right) \\ &\quad \times 2^{-2t} \sum_{X_{m-1}, Y_{m-1}\in S_{g_{m-1}}} (-1)^{U_{m-1}\cdot(X_{m-1}\boxplus Y_{m-1}\boxplus g_{m-1})\oplus V_{m-1}\cdot X_{m-1}\oplus W_{m-1}\cdot Y_{m-1}} \\ &= \sum_{i=0, g_i\in T_i}^{m-1} \prod_{k=0}^{m-2} C_{g_k}^{g_{k+1}}(U_k, V_k, W_k) \cdot C_{g_{m-1}}(U_{m-1}, V_{m-1}, W_{m-1}). \end{aligned}$$

Then we complete the proof.

When computing the linear correlation of addition modulo 2^n, it only needs to build 4 tables of $C_g^h(U,V,W)$ for $g,h \in \mathbb{F}_2$, and 2 tables of $C_g(U,V,W)$ for $g \in \mathbb{F}_2$. Then the linear correlation $C(\mu,\nu,\omega)$, where $\mu,\nu,\omega \in \mathbb{F}_2^n$, can be computed by looking up these tables according to Theorem 2.

Theorem 1 is an illustration of the idea of using CDDTs to compute the differential probability of addition. However, it is still not efficient enough to compute the differential probability in search algorithm, since it needs to traverse all 4^{m-1} carry bit flags and sum the multiplications of corresponding probabilities together. This is still complicated when we want to search all possible differential trails to get the optimal differential trail. It is the same case for computing the linear correlation of addition according to Theorem 2.

Next, we give an improved method to construct CDDTs and CLATS, which can be used in the search algorithm efficiently.

3 Improved Method for Constructing CDDTs and CLATS

3.1 Constructing CDDTs with Lipmaa-Moriai's Algorithm

In this section, we give a new construction of CDDTs with Lipmaa-Moriai's algorithm [24], which is a log-time algorithm for computing the differential probability of addition. By looking up these CDDTs, it is easy to get all possible output differences and their differential probabilities when given input differences.

A pseudo-code of Lipmaa-Moriai's algorithm is listed in Algorithm 1. For an integer j, 0_j denotes a j-bit vector with all entries equal 0 throughout this section.

Algorithm 1 Log-time algorithm for $P(\alpha,\beta\mapsto\gamma)$

```
Input: $(\alpha,\beta,\gamma)$
Output: $P(\alpha,\beta\mapsto\gamma)$
1: if $eq(\alpha\lll 1, \beta\lll 1, \gamma\lll 1) \wedge (\alpha\oplus\beta\oplus\gamma\oplus(\beta\lll 1)) \neq 0_n$ then
2:     return 0;
3: else
4:     return $2^{-\mathrm{wt}(\overline{eq(\alpha,\beta,\gamma)}\wedge mask(n-1))}$;
5: end if
```

For a t-bit vector $X = (x_{t-1}, \ldots, x_0)$, we define $h(X) = x_{t-1}$. For $A, B, \varGamma \in \mathbb{F}_2^t$ and $a, b, c \in \mathbb{F}_2$, let

$$f^{a,b,c}(A, B, \varGamma) = eq((A \lll 1) \oplus a, (B \lll 1) \oplus b, (\varGamma \lll 1) \oplus c) \\ \wedge (A \oplus B \oplus \varGamma \oplus ((B \lll 1) \oplus b)),$$

where

$$(X \lll 1) \oplus e = (x_{t-2}, \ldots, x_0, e)$$

for $X = (x_{t-1}, x_{t-2}, \ldots, x_0) \in \mathbb{F}_2^t$ and $e \in \mathbb{F}_2$.

First, we have the following result.

Lemma 1. *Let $n = mt$, $\alpha, \beta, \gamma \in \mathbb{F}_2^n$, $A_k = \alpha[(k+1)t - 1 : kt]$, $B_k = \beta[(k+1)t - 1 : kt]$, and $\varGamma_k = \gamma[(k+1)t - 1 : kt]$, $0 \le k \le m-1$. Then the following statements hold.*

1. *Define $A_{-1}, B_{-1}, \varGamma_{-1}$ as 0_t. Then*

$$eq(\alpha \lll 1, \beta \lll 1, \gamma \lll 1) \wedge (\alpha \oplus \beta \oplus \gamma \oplus (\beta \lll 1)) = 0_n$$

if and only if

$$f^{h(A_{k-1}),h(B_{k-1}),h(\varGamma_{k-1})}(A_k, B_k, \varGamma_k) = 0_t$$

for $0 \le k \le m-1$.

2. *Let $t_k = t$ for $0 \le k \le m-2$ and $t_{m-1} = t - 1$. Then*

$$\mathrm{wt}(\overline{eq(\alpha, \beta, \gamma)} \wedge mask(n-1)) = \sum_{k=0}^{m-1} \mathrm{wt}(\overline{eq(A_k, B_k, \varGamma_k)} \wedge mask(t_k)).$$

Proof. Note that $\alpha = (A_{m-1}, A_{m-2}, \ldots, A_0) = \sum_{k=0}^{m-1} A_k 2^{kt}$, then

$$\alpha \lll 1 = (\alpha_{n-2}, \ldots, \alpha_0, 0) = \sum_{k=0}^{m-1} (\alpha_{(k+1)t-2}, \ldots, \alpha_{kt}, \alpha_{kt-1}) 2^{kt} = \sum_{k=0}^{m-1} ((A_k \lll 1) \oplus h(A_{k-1})) 2^{kt},$$

where $(X \lll 1) \oplus e = (x_{t-2}, \ldots, x_0, e)$ for $X = (x_{t-1}, x_{t-2}, \ldots, x_0) \in \mathbb{F}_2^t$, $e \in \mathbb{F}_2$, and A_{-1} is defined as 0_t, and this implies

$$\alpha_{-1} = h(A_{-1}) = 0.$$

Similarly, we have

$$\beta \lll 1 = \sum_{k=0}^{m-1} ((B_k \lll 1) \oplus h(B_{k-1})) 2^{kt}, \gamma \lll 1 = \sum_{k=0}^{m-1} ((\varGamma_k \lll 1) \oplus h(\varGamma_{k-1})) 2^{kt}.$$

Then it holds

$$eq(\alpha \lll 1, \beta \lll 1, \gamma \lll 1) \wedge (\alpha \oplus \beta \oplus \gamma \oplus (\beta \lll 1)) = \sum_{k=0}^{m-1} f^{h(A_{k-1}),h(B_{k-1}),h(\varGamma_{k-1})}(A_k, B_k, \varGamma_k) 2^{kt}.$$

Therefore, the first item holds. The last item is easy to prove, since that

$$mask(n-1) = \sum_{k=0}^{n-2} 2^k = \sum_{k=0}^{m-2} (2^t - 1) 2^{kt} + (2^{t-1} - 1) 2^{(m-1)t} = \sum_{k=0}^{m-1} (2^{t_k} - 1) 2^{kt} = \sum_{k=0}^{m-1} mask(t_k) 2^{kt},$$

where $t_k = t$ for $0 \le k \le m-2$ and $t_{m-1} = t - 1$. Then we complete the proof.

Based on Lemma 1 and Algorithm 1, it is easy to prove the following result, which plays an important role in our search algorithm.

Theorem 3. *Let $n = mt$, $\alpha, \beta, \gamma \in \mathbb{F}_2^n$, $A_k = \alpha[(k+1)t-1 : kt]$, $B_k = \beta[(k+1)t-1 : kt]$, and $\Gamma_k = \gamma[(k+1)t-1 : kt]$, $0 \le k \le m-1$. For $A, B, \Gamma \in \mathbb{F}_2^t$, $a, b, c, d \in \mathbb{F}_2$, let*

$$PS_d^{a,b,c}(A, B \mapsto \Gamma) = \begin{cases} 2^{-\mathrm{wt}(\overline{eq(A,B,\Gamma)} \wedge mask(t_d))}, & f^{a,b,c}(A,B,\Gamma) = 0_t; \\ 0, & else, \end{cases}$$

where $t_0 = t$ and $t_1 = t-1$. Then

$$P(\alpha, \beta \mapsto \gamma) = \prod_{k=0}^{m-1} PS_{d_k}^{h(A_{k-1}),h(B_{k-1}),h(\Gamma_{k-1})}(A_k, B_k \mapsto \Gamma_k),$$

where $d_k = 0$ for $0 \le k \le m-2$, $d_{m-1} = 1$, and $A_{-1}, B_{-1}, \Gamma_{-1}$ are defined as 0_t.

Due to the two different values of t_d, there are two types of CDDTs in the above theorem, which are

$$PS_0^{a,b,c}(A, B \mapsto \Gamma)$$

and

$$PS_1^{a,b,c}(A, B \mapsto \Gamma)$$

respectively. Note that $a, b, c \in \mathbb{F}_2$, then each of them contains 2^3 tables. These tables can be constructed with Algorithm 2, where a, b, c are represented by an integer $N = a||b||c$. Then all differential probability of addition modulo 2^n can be computed by looking up these 16 tables according to Theorem 3.

Algorithm 2 Constructing CDDT with Lipmaa-Moriai's algorithm

```
1: //$N = a||b||c$
2: for $N = 0$ to $2^3 - 1$ do
3:   for $A, B, \Gamma = 0$ to $2^t - 1$ do
4:     $flag := eq((A \lll 1) \oplus a, (B \lll 1) \oplus b, (\Gamma \lll 1) \oplus c) \wedge (A \oplus B \oplus \Gamma \oplus ((B \lll 1) \oplus b))$;
5:     if $flag = 0_t$ then
6:       $PS_0^N(A, B \mapsto \Gamma) = 2^{-\mathrm{wt}(\overline{eq(A,B,\Gamma)} \wedge mask(t))}$;
7:       $PS_1^N(A, B \mapsto \Gamma) = 2^{-\mathrm{wt}(\overline{eq(A,B,\Gamma)} \wedge mask(t-1))}$;
8:     else
9:       $PS_0^N(A, B \mapsto \Gamma) = 0$;
10:      $PS_1^N(A, B \mapsto \Gamma) = 0$;
11:    end if
12:  end for
13: end for
```

Similar as the search algorithms for S-box based ciphers, it is very efficient to search for differential trails in ARX ciphers using CDDTs constructed with Theorem 3. In our search algorithm, the CDDTs $PS_0^{a,b,c}$ and $PS_1^{a,b,c}$ are pre-constructed. Given two input differences α and β of addition modulo 2^n, we can compute all possible output differences γ and their probabilities as follows: Firstly, we divide the n-bit input differences α and β into m portions A_k and B_k $(0 \le k \le m-1)$, and each portion contains t bits (assume $n = mt$). Then, we look up the CDDTs $PS_0^{a,b,c}$ to get all possible output differences Γ_k for $k = 0$ to $m-2$, and $PS_1^{a,b,c}$ to get all possible output differences Γ_{m-1}. When looking up the CDDTs for A_k and B_k, we can get all possible output differences Γ_k and their probabilities $PS_{d_k}^{a,b,c}(A_k, B_k \mapsto \Gamma_k)$ simultaneously. Meanwhile, we can compute $h(A_k), h(B_k)$ and $h(\Gamma_k)$ which are used to look up the next CDDT. Lastly, after looking up the m CDDTs, we get all possible output differences γ by concatenating Γ_k $(0 \le k \le m-1)$, and the corresponding differential probabilities $P(\alpha, \beta \mapsto \gamma)$ by multiplying $PS_{d_k}^{a,b,c}(A_k, B_k \mapsto \Gamma_k)$.

Remark 1. When searching for differential trails in ARX ciphers, it needs to compute all possible output differences and their differential probabilities given input differences of addition. There are two ways doing this with Lipmaa-Moriai's algorithm directly. The first way is to exhaustively search all output differences and check whether they are possible or not. This is very time-consuming. Another way is that given input difference α and β, one can build a system of equations from the condition

$$eq(\alpha \lll 1, \beta \lll 1, \gamma \lll 1) \wedge (\alpha \oplus \beta \oplus \gamma \oplus (\beta \lll 1)) = 0_n,$$

and then get all possible output differences γ by solving the system of equations. This is equivalent to the case of $t = 1$ of our method in this section. Furthermore, solving equations is less efficient than looking up tables directly in search algorithm. This is the reason that we do not use Lipmaa-Moriai's algorithm directly in our search algorithm.

3.2 Constructing CLATs with Schulte-Geers's algorithm

In [42], Wallén presented an algorithm to compute linear approximation of addition modulo 2^n. Schulte-Geers [37] also gave a simple explicit formula for the correlation of addition modulo 2^n with CCZ-equivalence.

In this section, we give a new construction of CLATs with Schulte-Geers's algorithm. By looking up these CLATs, it is easy to get all possible output masks and their correlations when given input masks. We list Schulte-Geers's theorem as follows.

Theorem 4 (Walsh transform of addition modulo 2^n [37]). *Let $\mu, \nu, \omega \in \mathbb{F}_2^n$, $M_n : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ denotes the left shifted "partial sums mapping"*

$$x = (x_{n-1}, \ldots, x_0) \mapsto M_n(x) = (x_{n-2} \oplus \ldots \oplus x_0, \ldots, x_1 \oplus x_0, x_0, 0).$$

Let $z := M_n^T(\mu \oplus \nu \oplus \omega)$, then

$$C(\mu, \nu, \omega) = 1_{\{\mu\oplus\nu\preceq z\}} 1_{\{\mu\oplus\omega\preceq z\}} (-1)^{(\mu\oplus\nu)\cdot(\mu\oplus\omega)} 2^{-\mathrm{wt}(z)},$$

where 1_{G_f} is the indicator function of the graph $G_f := \{(x, f(x)) \mid x \in \mathbb{F}_2^n\}$, and $M_n^T(x_{n-1}, \ldots, x_0) = (0, x_{n-1}, x_{n-1} \oplus x_{n-2}, \ldots, x_{n-1} \oplus \cdots \oplus x_1)$.

According to Theorem 4, it is easy to compute the correlation $C(\mu, \nu, \omega)$ given the input mask ν, ω and the output mask μ. However, when searching for linear trails in ARX ciphers, only input masks are given, and it needs to compute all possible output masks and their correlations. So it is not a very good choice to use Schulte-Geers's algorithm directly in the search algorithm.

Similar to the construction of CDDTs with Lipmaa-Moriai's algorithm, we can build CLATs with Schulte-Geers's algorithm. Then in the search algorithm, it can compute the correlation of addition by looking up CLATs efficiently, just as the case in S-box based ciphers.

In the following, we give the construction of CLATs with Schulte-Geers's algorithm. Suppose $n = mt$. For $U = (u_{t-1}, \ldots, u_0) \in \mathbb{F}_2^t$ and $e \in \mathbb{F}_2$, let $U \oplus e^t = (e \oplus u_{t-1}, \ldots, e \oplus u_0)$. For $X = (x_{n-1}, \ldots, x_0) \in \mathbb{F}_2^n$, let $X_k = (x_{(k+1)t-1}, \ldots, x_{kt}) \in \mathbb{F}_2^t$, $0 \le k \le m-1$, and $e_k = \bigoplus_{i=kt}^{n-1} x_i$. Then it holds

$$\begin{aligned} M_n^T(x_{n-1}, \ldots, x_0) &= (0, x_{n-1}, x_{n-1} \oplus x_{n-2}, \ldots, x_{n-1} \oplus \cdots \oplus x_1) \\ &= (M_t^T(X_{m-1}), M_t^T(X_{m-2}) \oplus e_{m-1}^t, \ldots, M_t^T(X_0) \oplus e_1^t). \end{aligned}$$

For $\mu, \nu, \omega \in \mathbb{F}_2^n$, let $x = \mu \oplus \nu \oplus \omega$, $z = M_n^T(x)$, $e_k = \oplus_{i=kt}^{n-1} x_i$, $0 \le k \le m-1$, and $e_m = 0$. Then we have

$$1_{\{\mu\oplus\nu\preceq z\}} 1_{\{\mu\oplus\omega\preceq z\}} = 1$$

if and only if

$$1_{\{U_k\oplus V_k\preceq Z_k\}} 1_{\{U_k\oplus W_k\preceq Z_k\}} = 1$$

for $0 \le k \le m-1$, where $Z_k = M_t^T(U_k \oplus V_k \oplus W_k) \oplus e_{k+1}^t$. Furthermore, it also holds

$$\mathrm{wt}(z) = \sum_{k=0}^{m-1} \mathrm{wt}(Z_k).$$

Then according to Theorem 4, we have the following result.

Theorem 5. *Let $n = mt$, $\mu, \nu, \omega \in \mathbb{F}_2^n$, $U_k = \mu[(k+1)t-1 : kt]$, $V_k = \nu[(k+1)t-1 : kt]$, and $W_k = \omega[(k+1)t-1 : kt]$, $0 \le k \le m-1$. For $U, V, W \in \mathbb{F}_2^t$, $e \in \mathbb{F}_2$, let*

$$CL_e(U, V, W) = 1_{\{U\oplus V\preceq Z\}} 1_{\{U\oplus W\preceq Z\}} 2^{-\mathrm{wt}(Z)},$$

where $Z = M_t^T(U \oplus V \oplus W) \oplus e^t$. Let $\sigma = (\mu \oplus \nu \oplus \omega)$, $e_m = 0$, and $e_k = (\bigoplus_{i=kt}^{(k+1)t-1} \sigma_i) \oplus e_{k+1}$ for $k = m-1, m-2, \ldots, 0$. Then

$$|C(\mu, \nu, \omega)| = \prod_{k=0}^{m-1} CL_{e_{k+1}}(U_k, V_k, W_k).$$

The carry-bit-dependent linear approximation table can be constructed with Algorithm 3, where M_t is the left shifted "partial sums mapping". Then all linear correlation of addition modulo 2^n can be computed by looking up CLATs according to Theorem 5. The computation is very similar to the case of computing differential probability with CDDTs.

Algorithm 3 Constructing CLAT with Schulte-Geers's algorithm

```
1: //$M_t$ is the left shifted "partial sums mapping"
2: for $e = 0$ to $1$ do
3:    for $U, V, W = 0$ to $2^t - 1$ do
4:       $Z = M_t^T(U \oplus V \oplus W) \oplus e^t$;
5:       if $U \oplus V \preceq Z$ and $U \oplus W \preceq Z$ then
6:          $CL_e(U, V, W) = 2^{-\mathrm{wt}(Z)}$;
7:       else
8:          $CL_e(U, V, W) = 0$;
9:       end if
10:   end for
11: end for
```

In the search algorithm, the CLATs CL_e are pre-constructed. Given two input masks ν and ω of addition modulo 2^n, we can compute all possible output masks μ and their correlations as follows: Firstly, we divide the n-bit input masks ν and ω into m portions V_k and W_k $(0 \leq k \leq m-1)$, and each portion contains t bits (assume $n = mt$). Then, we look up the CLATs CL_e to get all possible output masks U_k for $k = m-1$ to 0. When looking up the CLATs for V_k and W_k, we can get all possible output masks U_k and their correlations $CL_{e_{k+1}}(U_k, V_k, W_k)$ simultaneously. Meanwhile, we can compute $e_k = \bigoplus_{i=0}^{t-1}(U_k \oplus V_k \oplus W_k)[i] \oplus e_{k+1}$ which is used to look up the next CLAT. Lastly, after looking up the m CLATs, we get all possible output masks μ by concatenating U_k $(0 \leq k \leq m-1)$, and the corresponding correlations $C(\mu, \nu, \omega)$ by multiplying $CL_{e_{k+1}}(U_k, V_k, W_k)$.

4 Automatic Search Algorithm for Optimal Differential and Linear Trails with CDDTs and CLATs

In 1994, Matsui [27] proposed a practical algorithm to search for the best differential trails (resp. linear trails) of DES. The algorithm performs a recursive search for differential trails (resp. linear trails) over a given number of rounds n $(n \geq 1)$. It derives the best n-round differential probability B_n (resp. absolute correlation) from the knowledge of the best i-round probability B_i $(1 \leq i \leq n-1)$ and the initial estimate $\overline{B_n}$ for B_n. However, Matsui's algorithm is only applicable to block ciphers that have S-boxes. Recently, Biryukov et al. [12] firstly adapted Matsui's algorithm for finding the best differential and linear trails in ARX ciphers. Their algorithm found some best differential trails on round-reduced variants of SPECK. However, it is very time-consuming to find these best differential and linear trails. It only reported the best linear trails for SPECK32 reduced up to 6 rounds, and it is hard to find the best linear trails for other versions of SPECK.

In this section, we also extend Matsui's algorithm to ARX ciphers by introducing CDDTs and CLATs. Since "XOR branch" and "three-forked branch" are mutually dual operations in regard to differentials and linear masks, the search for optimal differential trails is essentially the same as that for optimal linear trails, and we focus on searching for optimal differential trails in the following. We assume the cipher has a Feistel structure and its round function is depicted in Fig 2. In Fig 2, F stands for the linear layer including rotation, shift and XOR.

Our algorithm is similar to Matsui's algorithm except that it looks up the CDDTs to get all possible output differences and their probabilities when computing the differential probability of addition modulo 2^n. Besides, we make $\overline{B_n}$ equal to B_{n-1}, so our algorithm starts with a high probability and searches for a differential trail with this probability. If such a differential trail doesn't exist, we lower the probability. Because our algorithm is capable of searching for all possible differential trails, we can get the best differential trail. The pseudo-code of our algorithm for differential trails is listed in Algorithm 4. The search algorithm for optimal linear trails is analogous to that for differential trails, and it only needs to look up the CLATs to compute the possible output masks and their correlations.

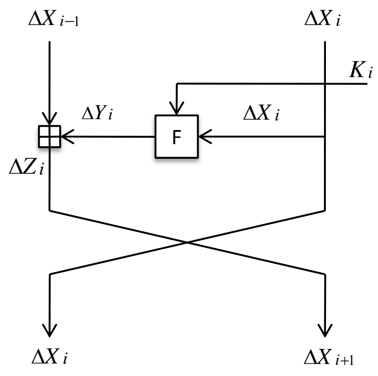

Fig. 2. The spread of differential values

To improve the efficiency of our search algorithm, we introduce some optimizing strategies including what Matsui's algorithm did. Besides, we sort the CDDTs (resp. CLATs) according to the differential probability (resp. absolute correlation). So we can always get the output difference (reps. output mask) with highest differential probability (resp. absolute correlation), and once we find some difference (resp. mask) whose probability (resp. absolute correlation) doesn't satisfy the search condition, we can break the unnecessary branches as soon as possible. For the best n-round differential probability B_n (resp. absolute correlation), there must exist one round whose differential probability (resp. absolute correlation) is greater than or equal to $\lceil B_n/n \rceil$. Therefore, it can start from the round with probability greater than or equal to $\lceil B_n/n \rceil$, and search the differential trails (resp. linear trails) forward and backward, which makes the search space very small and improves the efficiency significantly.

In the following, we give a rough estimation of the complexity of the differential search algorithm. Let m_1 be the number of differences α_1, β_1 and γ_1 in the first round, where $m_1 = \#\{(\alpha_1, \beta_1 \mapsto \gamma_1) \mid P(\alpha_1, \beta_1 \mapsto \gamma_1) \geq \overline{B}_n/B_{n-1}\}$. As the complexity of the search is dominated by the number of differences in the first round, the complexity of Algorithm 4 has the form $\mathcal{O}(m_1)$, which is significantly lower than the complexity of full search 2^{2n} according to our experiments, where $2n$ is the block size. However, it is difficult to get the precise value of m_1, since it changes dynamically in the search. As for the linear search, it has a similar estimation of the complexity.

5 Differential and Linear Trails for HIGHT and SPECK

In this section, we apply Algorithm 4 to block ciphers HIGHT and SPECK [4]. In the experiments, we use 8-bit CDDTs and CLATs. The differential and linear trails found by our algorithm are optimal. As for differential trails, we focus on the XOR difference of HIGHT and SPECK.

5.1 Description of HIGHT and SPECK

At CHES 2006, Hong et al. [21] presented a lightweight block cipher HIGHT with 64-bit block size and 128-bit key. It is a 8-branch generalized Feistel block cipher and consists of 32 rounds. Its round function is composed of addition modulo 2^8, rotation and XOR. The round function uses two auxiliary functions F_0 and F_1 defined as:

$$F_0(x) = (x \lll 1) \oplus (x \lll 2) \oplus (x \lll 7),$$

$$F_1(x) = (x \lll 3) \oplus (x \lll 4) \oplus (x \lll 6).$$

Let $X_{i-1} = X_{i-1,7} || \cdots || X_{i-1,0}$ be the input of the i-th round, $(SK_{4i-1}, SK_{4i-2}, SK_{4i-3}, SK_{4i-4})$ be the i-th round subkey, and the output of the i-th round $X_i = X_{i,7} || \cdots || X_{i,0}$ is computed as follows:

$$X_{i,7} = X_{i-1,6},\ \ X_{i,5} = X_{i-1,4},\ \ X_{i,3} = X_{i-1,2},\ \ X_{i,1} = X_{i-1,0},$$

[4] All experiments are performed on a PC with a single core (Intel® Core$^{\text{TM}}$ i5 − 4570 CPU 3.2GHz). We implicitly assume the independence of inputs when computing the differential probability and correlation of addition.

Algorithm 4 Matsui Search for Differential Trails Using CDDTs

```
 1: Procedure Main:
 2:   **Begin the program**
 3:   Let $\overline{B}_n = 2 \times B_{n-1}$, and $B_n = 1$.
 4:   Do
 5:     Let $\overline{B}_n = 2^{-1} \times \overline{B}_n$;
 6:     Call Procedure Round-1;
 7:   while $\overline{B}_n \neq B_n$.
 8:   **Exit the program**

 9: Procedure Round-1:
10:   For each candidate for $\Delta X_0, \Delta X_1$, do the following:
11:     $\Delta Y_1 = F(\Delta X_1)$;
12:     Let $p_1 = P(\Delta X_0, \Delta Y_1 \mapsto \Delta Z_1)$;
13:     If $p_1 \times B_{n-1} \geq \overline{B}_n$, then Call Procedure Round-2;

14: Procedure Round-$i$ ($2 \leq i \leq n-1$):
15:   For each candidate for $\Delta Z_i$, do the following:
16:     Let $\Delta X_i = \Delta Z_{i-1}$ and $\Delta Y_i = F(\Delta X_i)$;
17:     Let $p_i = P(\Delta X_{i-1}, \Delta Y_i \mapsto \Delta Z_i)$;
18:     If $p_1 \times p_2 \times \cdots \times p_i \times B_{n-i} \geq \overline{B}_n$, then Call Procedure Round-$(i+1)$;
19:   Return to the upper procedure;

20: Procedure Round-$n$:
21:   Let $\Delta X_n = \Delta Z_{n-1}$ and $\Delta Y_n = F(\Delta X_n)$;
22:   Let $p_n = \max_{\Delta Z_n} P(\Delta X_{n-1}, \Delta Y_n \mapsto \Delta Z_n)$;
23:   If $p_1 \times p_2 \times \cdots \times p_n = \overline{B}_n$, then $B_n = \overline{B}_n$;
24:   Return to the upper procedure;
```

Note: When computing $P(\Delta X_{i-1}, \Delta Y_i \mapsto \Delta Z_i)$, it only needs to look up the CDDTs to get the probability. When searching for linear trails, it needs to look up the CLATs to compute the absolute correlation.

$$X_{i,6} = X_{i-1,5} \boxplus (F_1(X_{i-1,4}) \oplus SK_{4i-2}),\ X_{i,4} = X_{i-1,3} \oplus (F_0(X_{i-1,2}) \boxplus SK_{4i-3}),$$
$$X_{i,2} = X_{i-1,1} \boxplus (F_1(X_{i-1,0}) \oplus SK_{4i-4}),\ X_{i,0} = X_{i-1,7} \oplus (F_0(X_{i-1,6}) \boxplus SK_{4i-1}),$$

The round function of HIGHT is shown in Fig 3.

The SPECK family of lightweight block cipher was designed by NSA in 2013 [5]. It consists of five instances SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128 with block sizes 32, 48, 64, 96 and 128 bits respectively. The instance with block size $2n$ ($n \in \{16, 24, 32, 48, 64\}$) and key size mn ($m \in \{2, 3, 4\}$ depending on n) is denoted by SPECK$2n/mn$.

SPECK has a structure similar to Threefish and utilizes a simple round function consisting of three operations: addition ($\boxplus$) , XOR ($\oplus$) and rotation ($\lll$). Its round function is defined as:

$$F(x, y) = (x \ggg \alpha) \boxplus y.$$

Let (L_{i-1}, R_{i-1}) be the input of the i-th round, K_i be the i-th round subkey, and the output of the i-th round (L_i, R_i) is computed as follows:

$$L_i = F(L_{i-1}, R_{i-1}) \oplus K_i,\ R_i = (R_{i-1} \lll \beta) \oplus L_i,$$

where the rotation amounts are $\alpha = 7$ and $\beta = 2$ if the block size is 32-bit and $\alpha = 8$ and $\beta = 3$ otherwise. The round function of SPECK is shown in Fig 4.

5.2 Differential Trails for HIGHT and SPECK

The optimal differential trail found for HIGHT covers 10 rounds and has probability 2^{-38}, which is the first public result of optimal differential trail for HIGHT. We also find an 12-round differential trail for HIGHT with probability 2^{-53}, but we can't ensure it is the optimal differential trail, because we limit the Hamming weight of the input differences and don't traverse the input differences. The probabilities of the optimal differential trails for HIGHT are shown in Table 3. And the differential trail found for HIGHT is shown in Table 9 in Appendix B. In this table, $\sum_r log_2 p_r$ represents the

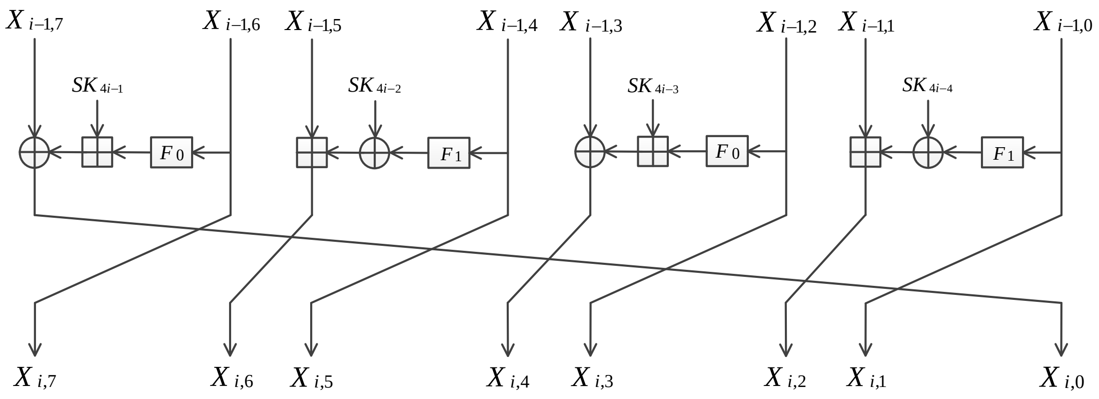

Fig. 3. The round function of HIGHT

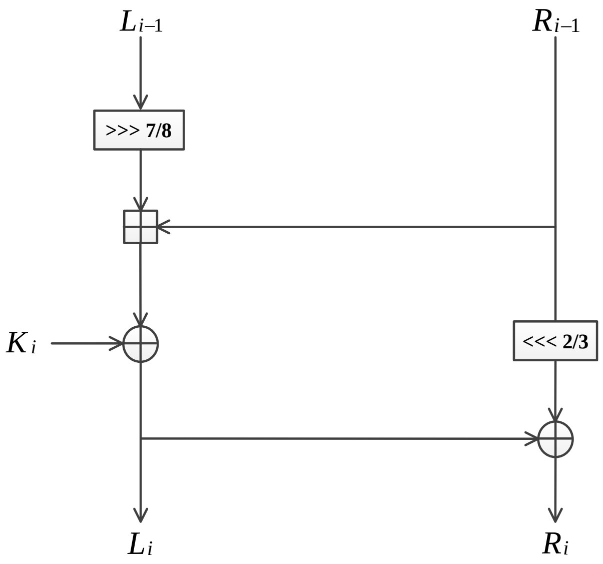

Fig. 4. The round function of SPECK

logarithm of the probability of a differential trail obtained as the sum of the probabilities of its transitions. *Time* represents the time consumed to find the differential trails with corresponding rounds, where 'h' is the abbreviation of hours.

Table 3. Probabilities of the optimal differential trails for HIGHT. The probabilities are given as $log_2 p$ ("$\geq$" indicates a lower bound).

Rounds	1	2	3	4	5	6	7	8	9	10	11	12
HIGHT	0	0	-3	-8	-11	-15	-19	-26	-30	-38	≥ -45	≥ -53

The optimal differential trails found for SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128 covers 10, 12, 16, 8 and 8 rounds with probability 2^{-34}, 2^{-49}, 2^{-70}, 2^{-30} and 2^{-30} respectively. For SPECK48, SPECK64, SPECK96 and SPECK128, the optimal differential trails found by our algorithm cover more rounds than that given by Biryukov et al. [12]. Meanwhile we give the provably optimal differential trails for SPECK48 and SPECK64 for the first time. As for SPECK32, our result is the same as that of Biryukov et al.. Due to the length of paper, we present 1- to 16- round differential trails of SPECK32/48/64 with max differential probability and procedure running time at https://github.com/jiaolin2019/differential-trails-and-linear-trails-of-SPECK.git.

The probabilities of the optimal differential trails for SPECK are shown in Table 4. And the differential trails found for SPECK are shown in Table 10 in Appendix B.

Table 4. Probabilities of the optimal differential trails for SPECK. The probabilities are given as log_2p ("$\geq$" indicates a lower bound).

Rounds	1	2	3	4	5	6	7	8
SPECK32	0	-1	-3	-5	-9	-13	-18	-24
SPECK48	0	-1	-3	-6	-10	-14	-19	-26
SPECK64	0	-1	-3	-6	-10	-15	-21	-29
SPECK96	0	-1	-3	-6	-10	-15	-21	-30
SPECK128	0	-1	-3	-6	-10	-15	-21	-30
Rounds	9	10	11	12	13	14	15	16
SPECK32	-30	-34						
SPECK48	-33	-40	-45	-49				
SPECK64	-34	-38	-42	-46	-50	-56	-62	-70
SPECK96	≥ -39	≥ -49						
SPECK128	≥ -39	≥ -49						

5.3 Linear Trails for HIGHT and SPECK

As for linear trails, we report for the first time the optimal linear trail for up to 9 rounds of HIGHT with correlation 2^{-15}. We also find a 10-round linear trail for HIGHT with correlation 2^{-19}, but we can't ensure it is the optimal linear trail, because we limit the Hamming weight of the input masks and don't traverse the input masks. The correlations of the optimal linear trails for HIGHT are shown in Table 5. And the linear trail found for HIGHT is shown in Table 11 in Appendix C. In this table, $\sum_r log_2c_r$ represents the logarithm of the absolute correlation of a linear trail obtained as the sum of the correlations of its transitions. *Time* represents the time consumed to find the differential trails with corresponding rounds, where 's', 'h' and 'd' represent the abbreviation of seconds, hours and days respectively.

Table 5. Correlations of the optimal linear trails for HIGHT. The correlations are given as log_2c ("$\geq$" indicates a lower bound).

Rounds	1	2	3	4	5	6	7	8	9	10
HIGHT	0	0	0	-2	-4	-6	-8	-12	-15	≥ -19

For SPECK with block size 32, 48, 64, 96 and 128 bits, we find the optimal linear trails for up to 22, 13, 15, 9 and 9 rounds. We give the provably optimal linear trails for SPECK32, SPECK48 and SPECK64, which confirm the optimal linear trails given by Liu et al. [25]. The correlations of the optimal linear trails for SPECK are shown in Table 6. And the optimal linear trails found for SPECK are shown in Table 12 in Appendix C. Due to the length of paper, we present 1- to 16- round linear trails of SPECK32/48/64 with max linear correlation and procedure running time at https://github.com/jiaolin2019/differential-trails-and-linear-trails-of-SPECK.git.

6 Conclusion

In this paper, we adapt Matsui's algorithm and propose an automatic search algorithm for optimal differential and linear trails in ARX ciphers. We use the block ciphers HIGHT and SPECK as a

Table 6. Correlations of the optimal linear trails for SPECK. The correlations are given as $log_2 c$.

Rounds	1	2	3	4	5	6	7	8	9	10	11
SPECK32	0	0	−1	−3	−5	−7	−9	−12	−14	−17	−19
SPECK48	0	0	−1	−3	−6	−8	−12	−15	−19	−22	−25
SPECK64	0	0	−1	−3	−6	−9	−13	−17	−19	−21	−24
SPECK96	0	0	−1	−3	−6	−9	−13	−18	−22		
SPECK128	0	0	−1	−3	−6	−9	−13	−18	−22		
Rounds	12	13	14	15	16	17	18	19	20	21	22
SPECK32	−20	−22	−24	−26	−28	−30	−34	−36	−38	−40	−42
SPECK48	−28	−30									
SPECK64	−27	−30	−33	−37							

test platform for demonstrating the practical application of our algorithm. Using the proposed algorithm, we find a 10-round optimal differential trail for HIGHT, which is reported for the first time. Optimal differential trails for $10, 12, 16, 8$ and 8 rounds of SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128 are found, where the provably optimal differential trails for SPECK48 and SPECK64 are presented for the first time so far. The provably optimal linear trail for 9-round HIGHT is also reported for the first time. As for SPECK with block size 32, 48, 64, 96 and 128 bits, the provably optimal linear trails on 22, 13, 15, 9 and 9 rounds are found. We hope that the algorithm proposed in this paper is helpful for evaluating the security of ARX ciphers against differential and linear cryptanalysis, and also useful in the design of ARX ciphers.

Acknowledgements. This work was supported by National Natural Science Foundation of China (Grant Nos. 61772517, 61902030, 61772516), Youth Innovation Promotion Association CAS.

References

1. K. Aoki, K. Kobayashi, and S. Moriai. Best differential characteristic search of FEAL. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, pages 41–53, 1997.
2. J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to NIST (Round 3), 2010.
3. A. Bannier, N. Bodin, and E. Filiol. Automatic search for a maximum probability differential characteristic in a substitution-permutation network. In *48th Hawaii International Conference on System Sciences, HICSS 2015, Kauai, Hawaii, USA, January 5-8, 2015*, pages 5165–5174, 2015.
4. Z. Bao, W. Zhang, and D. Lin. Speeding up the search algorithm for the best differential and best linear trails. In *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, pages 259–285, 2014.
5. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.
6. D. J. Bernstein. The salsa20 family of stream ciphers. In *New Stream Cipher Designs - The eSTREAM Finalists*, pages 84–97. 2008.
7. E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.
8. A. Biryukov and I. Nikolic. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, camellia, khazad and others. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 322–344, 2010.
9. A. Biryukov and I. Nikolic. Search for related-key differential characteristics in DES-like ciphers. In *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, pages 18–34, 2011.
10. A. Biryukov, A. Roy, and V. Velichkov. Differential analysis of block ciphers SIMON and SPECK. In *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, pages 546–570, 2014.
11. A. Biryukov and V. Velichkov. Automatic search for differential trails in ARX ciphers. In *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, pages 227–250, 2014.

12. A. Biryukov, V. Velichkov, and Y. L. Corre. Automatic search for the best trails in ARX: application to block cipher SPECK. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 289–310, 2016.
13. C. Bouillaguet, P. Derbez, and P. Fouque. Automatic search of attacks on round-reduced AES and applications. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 169–187, 2011.
14. C. D. Cannière and C. Rechberger. Finding SHA-1 characteristics: General results and applications. In *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, pages 1–20, 2006.
15. D. Dinu, L. Perrin, A. Udovenko, V. Velichkov, J. Großschädl, and A. Biryukov. Design strategies for ARX with provable bounds: Sparx and LAX. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 484–513, 2016.
16. C. Dobraunig, M. Eichlseder, and F. Mendel. Heuristic tool for linear cryptanalysis with applications to CAESAR candidates. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 490–509, 2015.
17. N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The skein hash function family. Submission to NIST (Round 3), 2010.
18. P. Fouque, G. Leurent, and P. Q. Nguyen. Automatic search of differential path in MD4. *IACR Cryptology ePrint Archive*, 2007:206, 2007.
19. K. Fu, M. Wang, Y. Guo, S. Sun, and L. Hu. MILP-Based automatic search algorithms for differential and linear trails for SPECK. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 268–288, 2016.
20. D. Hong, J. Lee, D. Kim, D. Kwon, K. H. Ryu, and D. Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In *Information Security Applications - 14th International Workshop, WISA 2013, Jeju Island, Korea, August 19-21, 2013, Revised Selected Papers*, pages 3–27, 2013.
21. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. HIGHT: A new block cipher suitable for low-resource device. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, pages 46–59, 2006.
22. G. Leurent. Analysis of differential attacks in ARX constructions. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 226–243, 2012.
23. G. Leurent. Construction of differential characteristics in ARX designs application to skein. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 241–258, 2013.
24. H. Lipmaa and S. Moriai. Efficient algorithms for computing differential properties of addition. In *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, pages 336–350, 2001.
25. Y. Liu, Q. Wang, and V. Rijmen. Automatic search of linear trails in ARX with applications to SPECK and Chaskey. In *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, pages 485–499, 2016.
26. M. Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, pages 386–397, 1993.
27. M. Matsui. On correlation between the order of s-boxes and the strength of DES. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, pages 366–375, 1994.
28. F. Mendel, T. Nad, and M. Schläffer. Finding SHA-2 characteristics: Searching through a minefield of contradictions. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 288–307, 2011.
29. N. Mouha, B. Mennink, A. V. Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, pages 306–323, 2014.
30. N. Mouha and B. Preneel. Towards finding optimal differential characteristics for ARX: Application to salsa20. Cryptology ePrint Archive, Report 2013/328, 2013.
31. N. Mouha, Q. Wang, D. Gu, and B. Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, pages 57–76, 2011.
32. R. M. Needham and D. J. Wheeler. TEA extensions. Computer Laboratory, Cambridge University, England, 1997. http://www.movable-type.co.uk/scripts/xtea.pdf.

33. K. Ohta, S. Moriai, and K. Aoki. Improving the search algorithm for the best linear expression. In *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, pages 157–170, 1995.
34. R. L. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, pages 303–311, 1990.
35. R. L. Rivest. The RC5 encryption algorithm. In *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, pages 86–96, 1994.
36. M. Schläffer and E. Oswald. Searching for differential paths in MD4. In *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, pages 242–261, 2006.
37. E. Schulte-Geers. On ccz-equivalence of addition mod 2^n. *Des. Codes Cryptography*, 66(1-3):111–127, 2013.
38. A. Shimizu and S. Miyaguchi. Fast data encipherment algorithm FEAL. In *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, pages 267–278, 1987.
39. L. Song, Z. Huang, and Q. Yang. Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In J. K. Liu and R. Steinfeld, editors, *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II*, volume 9723 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2016.
40. M. Stevens, A. K. Lenstra, and B. de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, pages 1–22, 2007.
41. S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 158–178, 2014.
42. J. Wallén. Linear approximations of addition modulo 2^n. In *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, pages 261–273, 2003.
43. D. J. Wheeler and R. M. Needham. TEA, a tiny encryption algorithm. In *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, pages 363–366, 1994.
44. Y. Yao, B. Zhang, and W. Wu. Automatic search for linear trails of the SPECK family. In *Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings*, pages 158–176, 2015.

A Example of Computing Differential Probability of Addition with CDDTs

In the following, we take 4-bit modulo addition and 2-bit CDDTs to illustrate how to compute the differential probability of addition with Theorem 1, which can be generalized to n-bit modulo addition easily. As for the generalized case, it needs to list 20 CDDT tables. Due to the space limitation, we take a particular case in the example, which only needs to list 8 tables.

Example 1. Let $x, y, \alpha, \beta, \gamma \in \mathbb{F}_2^4$, $z = x \boxplus y$, and α, β, γ be the XOR difference of x, y, z respectively. Let $\alpha = (1010)_2$, $\beta = (1101)_2$ and $\gamma = (0101)_2$. Then according to the definition of differential probability, it can be computed that

$$P(\alpha, \beta \mapsto \gamma) = 1/8.$$

Next we use Theorem 1 to compute the differential probability $P(\alpha, \beta \mapsto \gamma)$. Let c and c' be the carry vector of $x \boxplus y$ and $(x \oplus \alpha) \boxplus (y \oplus \beta)$ respectively. Since x, y are 4-bit vectors, then we choose $m = t = 2$. Note that $T_0 = \{(0, 0)\}$, then according to Theorem 1, we only need to build two types of 2-bit CDDTs PS_0^d and PS_d for $d \in \mathbb{F}_2^2$. Both PS_0^d and PS_d contains 4 tables and these tables are also denoted by PS_0^d and PS_d.

Let

$$\begin{array}{lll}
x = x_3x_2x_1x_0, & X_1 = x_3x_2, & X_0 = x_1x_0, \\
y = y_3y_2y_1y_0, & Y_1 = y_3y_2, & Y_0 = y_1y_0, \\
z = z_3z_2z_1z_0, & Z_1 = z_3z_2, & Z_0 = z_1z_0, \\
\alpha = \alpha_3\alpha_2\alpha_1\alpha_0, & A_1 = \alpha_3\alpha_2, & A_0 = \alpha_1\alpha_0, \\
\beta = \beta_3\beta_2\beta_1\beta_0, & B_1 = \beta_3\beta_2, & B_0 = \beta_1\beta_0, \\
\gamma = \gamma_3\gamma_2\gamma_1\gamma_0, & \mathit{\Gamma}_1 = \gamma_3\gamma_2, & \mathit{\Gamma}_0 = \gamma_1\gamma_0.
\end{array}$$

Then (A_0, B_0, Γ_0) are the input and output difference of the CDDT PS_0^d, and (A_1, B_1, Γ_1) are the input and output difference of the CDDT PS_d. The input difference (A_0, B_0) and (A_1, B_1) are stored as $(A_0||B_0)$ and $(A_1||B_1)$ respectively in the corresponding tables. The CDDTs PS_0^d and PS_d can be constructed with Algorithm 5 and Algorithm 6, and the tables are listed in Table 7 and Table 8.

As for the example,

$$\begin{aligned}&A_1 = \alpha_3\alpha_2 = (10)_2,\ A_0 = \alpha_1\alpha_0 = (10)_2,\\ &B_1 = \beta_3\beta_2 = (11)_2,\ B_0 = \beta_1\beta_0 = (01)_2,\\ &\Gamma_1 = \gamma_3\gamma_2 = (01)_2,\ \ \Gamma_0 = \gamma_1\gamma_0 = (01)_2.\end{aligned}$$

Then according to Theorem 1 and by looking up the CDDTs, we have

$$\begin{aligned}P(\alpha, \beta \mapsto \gamma) &= \sum_{d\in\mathbb{F}_2^2} PS_0^d(A_0, B_0 \mapsto \Gamma_0)PS_d(A_1, B_1 \mapsto \Gamma_1)\\ &= PS_0^0(1001 \mapsto 01)PS_0(1011 \mapsto 01) + PS_0^1(1001 \mapsto 01)PS_1(1011 \mapsto 01)\\ &\quad + PS_0^2(1001 \mapsto 01)PS_2(1011 \mapsto 01) + PS_0^3(1001 \mapsto 01)PS_3(1011 \mapsto 01)\\ &= 1/8 \times 1/2 + 1/8 \times 0 + 1/8 \times 0 + 1/8 \times 1/2\\ &= 1/8.\end{aligned}$$

Algorithm 5 Constructing CDDT PS_0^d

```
 1: for $A_0, B_0, \Gamma_0 = 0$ to 3 do
 2:   for $X_0, Y_0 = 0$ to 3 do
 3:     $Z_0 = X_0 \boxplus Y_0$;
 4:     $Z'_0 = (X_0 \oplus A_0) \boxplus (Y_0 \oplus B_0)$;
 5:     $c_2 = \lfloor (X_0 + Y_0)/4 \rfloor$;
 6:     $c'_2 = \lfloor ((X_0 \oplus A_0) + (Y_0 \oplus B_0))/4 \rfloor$;
 7:     $d = c_2||c'_2$;
 8:     if $Z_0 \oplus Z'_0 = \Gamma_0$ then
 9:       $S_0^d(A_0, B_0 \mapsto \Gamma_0) = S_0^d(A_0, B_0 \mapsto \Gamma_0) + 1$;
10:     end if
11:   end for
12:   $PS_0^d(A_0, B_0 \mapsto \Gamma_0) = 2^{-4} \times S_0^d(A_0, B_0 \mapsto \Gamma_0)$;
13: end for
```

Algorithm 6 Constructing CDDT PS_d

```
 1: for $A_1, B_1, \Gamma_1 = 0$ to 3 do
 2:   for $X_1, Y_1 = 0$ to 3 do
 3:     for $c_2, c'_2 = 0$ to 1 do
 4:       $Z_1 = X_1 \boxplus Y_1 \boxplus c_2$;
 5:       $Z'_1 = (Z_1 \oplus A_1) \boxplus (Y_1 \oplus B_1) \boxplus c'_2$;
 6:       $d = c_2||c'_2$;
 7:       if $Z_1 \oplus Z'_1 = \Gamma_1$ then
 8:         $S_d(A_1, B_1 \mapsto \Gamma_1) = S_d(A_1, B_1 \mapsto \Gamma_1) + 1$;
 9:       end if
10:     end for
11:   end for
12:   $PS_d(A_1, B_1 \mapsto \Gamma_1) = 2^{-4} \times S_d(A_1, B_1 \mapsto \Gamma_1)$;
13: end for
```

Table 7. The CDDT PS_0^d

	PS_0^0				PS_0^1				PS_0^2				PS_0^3			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	5/8	0	0	0	0	0	0	0	0	0	0	0	3/8	0	0	0
1	0	3/8	0	1/8	0	0	0	1/8	0	0	0	1/8	0	1/8	0	1/8
2	0	0	3/8	0	0	0	1/4	0	0	0	1/4	0	0	0	1/8	0
3	0	1/8	0	1/4	0	1/8	0	1/8	0	1/8	0	1/8	0	1/8	0	0
4	0	3/8	0	1/8	0	0	0	1/8	0	0	0	1/8	0	1/8	0	1/8
5	3/8	0	1/8	0	0	0	1/8	0	0	0	1/8	0	1/8	0	1/8	0
6	0	1/8	0	1/4	0	1/8	0	1/8	0	1/8	0	1/8	0	1/8	0	0
7	1/8	0	1/4	0	1/8	0	1/8	0	1/8	0	1/8	0	1/8	0	0	0
8	0	0	3/8	0	0	0	1/4	0	0	0	1/4	0	0	0	1/8	0
9	0	1/8	0	1/4	0	1/8	0	1/8	0	1/8	0	1/8	0	1/8	0	0
10	3/8	0	0	0	1/4	0	0	0	1/4	0	0	0	1/8	0	0	0
11	0	1/4	0	0	0	1/8	0	1/4	0	1/8	0	1/4	0	0	0	0
12	0	1/8	0	1/4	0	1/8	0	1/8	0	1/8	0	1/8	0	1/8	0	0
13	1/8	0	1/4	0	1/8	0	1/8	0	1/8	0	1/8	0	1/8	0	0	0
14	0	1/4	0	0	0	1/8	0	1/4	0	1/8	0	1/4	0	0	0	0
15	1/4	0	0	0	1/8	0	1/4	0	1/8	0	1/4	0	0	0	0	0

Table 8. The CDDT PS_d

	PS_0				PS_1				PS_2				PS_3			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	1	0	0	0	0	1/2	0	1/2	0	1/2	0	1/2	1	0	0	0
1	0	1/2	0	1/2	1/2	0	1/2	0	1/2	0	1/2	0	0	1/2	0	1/2
2	0	0	1	0	0	1/2	0	1/2	0	1/2	0	1/2	0	0	1	0
3	0	1/2	0	1/2	1/2	0	1/2	0	1/2	0	1/2	0	0	1/2	0	1/2
4	0	1/2	0	1/2	1/2	0	1/2	0	1/2	0	1/2	0	0	1/2	0	1/2
5	1/2	0	1/2	0	0	0	0	1	0	0	0	1	1/2	0	1/2	0
6	0	1/2	0	1/2	1/2	0	1/2	0	1/2	0	1/2	0	0	1/2	0	1/2
7	1/2	0	1/2	0	0	1	0	0	0	1	0	0	1/2	0	1/2	0
8	0	0	1	0	0	1/2	0	1/2	0	1/2	0	1/2	0	0	1	0
9	0	1/2	0	1/2	1/2	0	1/2	0	1/2	0	1/2	0	0	1/2	0	1/2
10	1	0	0	0	0	1/2	0	1/2	0	1/2	0	1/2	1	0	0	0
11	0	1/2	0	1/2	1/2	0	1/2	0	1/2	0	1/2	0	0	1/2	0	1/2
12	0	1/2	0	1/2	1/2	0	1/2	0	1/2	0	1/2	0	0	1/2	0	1/2
13	1/2	0	1/2	0	0	1	0	0	0	1	0	0	1/2	0	1/2	0
14	0	1/2	0	1/2	1/2	0	1/2	0	1/2	0	1/2	0	0	1/2	0	1/2
15	1/2	0	1/2	0	0	0	0	1	0	0	0	1	1/2	0	1/2	0

B Differential Trails for HIGHT and SPECK

Table 9. Differential Trail for HIGHT

	HIGHT								
r	ΔX_7	ΔX_6	ΔX_5	ΔX_4	ΔX_3	ΔX_2	ΔX_1	ΔX_0	$log_2 p_r$
0	0	0	81	e2	b0	0	c0	3	−0
1	0	0	e2	b0	0	e8	3	0	−3
2	0	0	b0	2	e8	7	0	0	−8
3	0	0	2	79	7	0	0	0	−4
4	0	0	79	7	0	0	0	0	−1
5	0	0	7	0	0	0	0	0	−5
6	0	1	0	0	0	0	0	0	−3
7	1	0	0	0	0	0	0	82	−2
8	0	0	0	0	0	9c	82	1	−3
9	0	0	0	3	9c	7a	1	0	−8
10	0	e8	3	bc	7a	1	0	0	−5
11	e8	0	bc	f8	1	0	0	2	−6
12	0	b6	f8	1	0	90	2	e8	−5
$\sum_r log_2 p_r$	−53								
$Time$	478.89h								

Table 10. Differential Trails for SPECK

	SPECK32			SPECK48			SPECK64		
r	ΔL	ΔR	$log_2 p_r$	ΔL	ΔR	$log_2 p_r$	ΔL	ΔR	$log_2 p_r$
0	50	8402	−0	1202	20002	−0	40004092	10420040	−0
1	2402	3408	−3	10	100000	−3	82020000	120200	−5
2	50c0	80e0	−7	0	800000	−1	900000	1000	−4
3	181	203	−4	800000	800004	−0	8000	0	−2
4	c	800	−5	808004	808020	−2	80	80	−1
5	2000	0	−3	8400a0	8001a4	−4	80000080	80000480	−1
6	40	40	−1	608da4	608080	−9	800480	802084	−3
7	8040	8140	−1	42003	2400	−11	80806080	848164a0	−6
8	40	542	−2	12020	20	−5	40f2400	20040104	−13
9	8542	904a	−4	200100	200000	−3	20000820	20200001	−8
10				202001	202000	−3	9	1000000	−4
11				210020	200021	−4	8000000	0	−2
12							80000	80000	−1
13							80800	480800	−2
14							480008	2084008	−4
15							a080808	1a4a0848	−6
$\sum_r log_2 p_r$	−30			−45			−62		
$Time$	0.15h			4.66h			0.24h		

	SPECK96			SPECK128		
r	ΔL	ΔR	$log_2 p_r$	ΔL	ΔR	$log_2 p_r$
0	800a080808	800124a0848	−0	800a080808	8000000124a0848	−0
1	92400040	400000104200	−10	92400040	4000000000104200	−10
2	820200	1202	−6	820200	1202	−6
3	9000	10	−4	9000	10	−4
4	80	0	−2	80	0	−2
5	800000000000	800000000000	−0	8000000000000000	8000000000000000	−0
6	808000000000	808000000004	−1	8080000000000000	8080000000000004	−1
7	800080000004	840080000020	−3	8000800000000004	8400800000000020	−3
8	808080800020	a08480800124	−5	8080808000000020	a084808000000124	−5
9	800400008124	842004008801	−9	8004000080000124	8420040080000801	−9
10	a0a000008880	81a02004c88c	−9	a0a0000080800800	81a020048080480c	−9
$\sum_r log_2 p_r$	−49			−49		
$Time$	48.3h			76.86h		

C Linear Trails for HIGHT and SPECK

Table 11. Linear Trail for HIGHT

	HIGHT								
r	ΓX_7	ΓX_6	ΓX_5	ΓX_4	ΓX_3	ΓX_2	ΓX_1	ΓX_0	$log_2 c_r$
0	0	0	0	81	80	11	41	39	−0
1	0	0	81	80	c0	61	0	0	−2
2	0	81	a3	c0	0	0	0	0	−2
3	81	a3	0	0	0	0	0	0	−3
4	0	0	0	0	0	0	0	81	−1
5	0	0	0	0	0	0	81	0	−0
6	0	0	0	0	0	81	23	0	−1
7	0	0	0	0	81	23	99	0	−2
8	0	0	0	81	80	91	60	0	−3
9	0	0	81	80	40	40	d	0	−2
10	0	81	a3	40	f0	d	45	0	−3
$\sum_r log_2 c_r$	−19								
$Time$	128.66h								

Table 12. Linear Trails for SPECK

	SPECK32			SPECK48			SPECK64		
r	ΓL	ΓR	$log_2 c_r$	ΓL	ΓR	$log_2 c_r$	ΓL	ΓR	$log_2 c_r$
0	280	5226	−0	900	20018c	−0	101000	1013	−0
1	4880	4885	−1	c00	c09	−1	1800	18	−2
2	20a0	2071	−2	6000	6008	−1	18	0	−1
3	40a0	c1	−2	30200	30240	−1	d8000000	c0000000	−1
4	80	4001	−3	180012	180210	−2	4100006	4800006	−2
5	0	1	−0	20080	109080	−4	34d030	430c030	−3
6	4	4	−0	49e06	849c06	−2	1870101	21872781	−6
7	3810	3010	−1	c000	4c694	−5	1300100	310601	−5
8	2180	1c0	−3	263020	2630a0	−1	1800001	181b000	−2
9	6cf	68c	−2	20	302400	−4	1000000	18000	−2
10				212000	12000	−1	10000	0	−1
11							d00	c00	−1
12							6041	6048	−1
13							4d030123	c030143	−3
$\sum_r log_2 c_r$	−14			−22			−30		
$Time$	7.76s			27.31h			8.44h		

	SPECK96			SPECK128		
r	ΓL	ΓR	$log_2 c_r$	ΓL	ΓR	$log_2 c_r$
0	1800130	40000018021	−0	1800130	400000000018021	−0
1	18100	200000000101	−2	18100	2000000000000101	−2
2	100	1	−1	100	1	−1
3	1	0	−0	1	0	−0
4	d0000000000	c0000000000	−1	d00000000000000	c00000000000000	−1
5	604000000000	604c00000000	−4	6040000000000000	604c000000000000	−4
6	3	6000000003	−8	3	60000000000003	−8
7	180000000018	1b0000000018	−1	1800000000000018	1b00000000000018	−1
8	90000000c0	1880000000c0	−2	900000000000c0	18800000000000c0	−2
9	40650000606	c406c0000606	−3	406500000000606	c406c00000000606	−3
$\sum_r log_2 c_r$	−22			−22		
$Time$	43d			56d		