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

\texttt{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 [42], 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 [1,3,4,8,9,13,27,31,33,40] 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 \(2^n\) 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 [14,16,18,22,23,28,36,39], 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. [43] also adapted Matsui’s algorithm to search for the optimal linear trails for SPECK. They applied Wallén’s algorithm [41] 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. Liu et al. [25] also applied the SAT solver approach to find the optimal linear trails for 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. 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. The truth value tables of partial sum can be constructed. Unlike the independence of S-boxes in S-box based ciphers, the truth value tables of partial sum are correlated because of the impact of carry bits. The truth value tables of partial sum are defined as carry-bit-dependent S-boxes.

   Similarly, the carry-bit-dependent difference distribution table (CDDT) can be constructed, which is the difference distribution table of partial sum and can be pre-constructed. The relationship of differential probability of addition modulo $2^n$ and CDDTs is characterized. With this characterization, it is very efficient to get all possible output differences and corresponding differential probabilities just by looking up CDDTs when given input differences. Also, the carry-bit-dependent linear approximation table (CLAT) can be constructed, and the linear correlation of addition modulo $2^n$ can be computed by looking up the CLATs.

2. With CDDTs and CLATs, we propose a general automatic search algorithm for optimal differential and linear trails in ARX ciphers. 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 $2^n$. To improve the efficiency of our algorithm, we give a new construction of CDDTs and CLATs with Lipmaa-Moriai’s algorithm [24] and Schulte-Geers’s result [37].

3. Using our algorithm, it is able to find the optimal differential trails for round-reduced versions of block ciphers 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].

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DT</td>
<td>LT</td>
<td>DT</td>
<td>LT</td>
</tr>
<tr>
<td>HIGHT</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SPECK32</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>SPECK48</td>
<td>9</td>
<td>–</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>SPECK64</td>
<td>8</td>
<td>–</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>SPECK96</td>
<td>7</td>
<td>–</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>SPECK128</td>
<td>6</td>
<td>–</td>
<td>19</td>
<td>16</td>
</tr>
</tbody>
</table>

Fu et al. [19] use the MILP-based algorithm to search for differential and linear trails for SPECK, and they apply a splicing heuristic to find better differential and linear trails than existing ones. Although their 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. The comparison of our identified differential and linear trails with those given by other algorithms is list in Table 1. In this table, BA, MA and SA represent the abbreviation of Biryukov et al.’s algorithm, MILP-based algorithm and SAT solver approach respectively. DT and LT represent the abbreviation of differential trails and linear trails respectively.

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

\[
\begin{align*}
c_0 &= 0 \\
c_{i+1} &= (x_i \land y_i) \oplus (x_i \land c_i) \oplus (y_i \land c_i) \text{ for } 0 \leq i \leq n - 2.
\end{align*}
\]

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 \( e \in \mathbb{F}_2 \), \( X \oplus Y \oplus e \) denotes

\[
X \oplus Y \oplus (0, \ldots, 0, e),
\]

which is equal to \( X \oplus Y \oplus \text{carry}^*(X,Y) \), and \( \text{carry}^*(X,Y) = (c_{t-1}^*, \ldots, c_0^*) \) is computed as follows

\[
\begin{align*}
c_0^* &= e \\
c_{i+1}^* &= (x_i \land y_i) \oplus (x_i \land c_i^*) \oplus (y_i \land c_i^*) \text{ for } 0 \leq i \leq t - 1. 
\end{align*}
\]

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:

\[
x \oplus y = x \oplus y \oplus \text{carry}(x,y)
\]

\[
= \sum_{k=0}^{m-1} (X_k \oplus Y_k \oplus C_k)2^{kt} \\
= \sum_{k=0}^{m-1} (X_k \oplus Y_k \oplus c_{kt})2^{kt}, \tag{2}
\]

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

Note that \( c_0 = 0 \) and for \( k \geq 0 \), \( c_{(k+1)t} \) can be computed from \( X_k \oplus Y_k \oplus c_{kt} \) by Formula (1). Then the computation of \( mt \)-bit vectors addition modulo \( 2^m \) 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 \textbf{carry-bit-dependent S-boxes}. We can also build difference distribution tables of addition modulo \( 2^t \) with different correlated values, and these

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \equiv y )</td>
<td>addition of ( x ) and ( y ) modulo ( 2^n )</td>
</tr>
<tr>
<td>( x[i : j] )</td>
<td>the sequence of bits ( x_i, x_{i-1}, \ldots, x_j )</td>
</tr>
<tr>
<td>( x \oplus y )</td>
<td>bitwise exclusive OR (XOR) of ( x ) and ( y )</td>
</tr>
<tr>
<td>( x \land y )</td>
<td>bitwise AND of ( x ) and ( y )</td>
</tr>
<tr>
<td>( x \lor y )</td>
<td>bitwise OR of ( x ) and ( y )</td>
</tr>
<tr>
<td>( x \ll r )</td>
<td>shift of ( x ) to the left by ( r ) positions</td>
</tr>
<tr>
<td>( x \gg r )</td>
<td>shift of ( x ) to the right by ( r ) positions</td>
</tr>
<tr>
<td>( x \lll r )</td>
<td>rotation of ( x ) to the left by ( r ) positions</td>
</tr>
<tr>
<td>( x \ggg r )</td>
<td>rotation of ( x ) to the right by ( r ) positions</td>
</tr>
<tr>
<td>( x | y )</td>
<td>concatenation of bit strings ( x ) and ( y )</td>
</tr>
<tr>
<td>( \text{wt}(x) )</td>
<td>the hamming weight of ( x )</td>
</tr>
<tr>
<td>( \Delta x )</td>
<td>XOR difference of ( x ) and ( x' : \Delta x = x \oplus x' )</td>
</tr>
<tr>
<td>( \text{eq}(x,y,z) )</td>
<td>( (x \oplus y) \land (x \oplus z) )</td>
</tr>
<tr>
<td>( \text{mask}(n) )</td>
<td>( 2^n - 1 )</td>
</tr>
<tr>
<td>( x \cdot y )</td>
<td>dot product of ( x ) and ( y : x \cdot y = \bigoplus_{i=0}^{n-1} x_i y_i )</td>
</tr>
<tr>
<td>( x \preccurlyeq y )</td>
<td>( x_i \preccurlyeq y_i, \forall i \in {0, \ldots, n-1} )</td>
</tr>
</tbody>
</table>
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](image)

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.

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 (α, β ↦→ γ), where α, β, γ ∈ F_2^n. The differential probability of addition modulo 2^n is defined as

\[ P(α, β ↦→ γ) = P_{x,y}[((x ⊕ y) ⊕ (x ⊕ α) ⊕ (y ⊕ β)) = γ]. \]

Hereafter this section, x, y, α, β, γ ∈ F_2^n always mean n-bit vectors. Let z = x ⊕ y, and α, β, γ the XOR difference of x, y, z respectively. This means

\[ x' = x ⊕ α, y' = y ⊕ β, z' = x' ⊕ y', \text{ and } z' = z ⊕ γ. \]

The carry bit vectors of x ⊕ y and x' ⊕ 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 ≤ k ≤ m - 1, let

\[ A_k = α[(k + 1)t - 1 : kt], B_k = β[(k + 1)t - 1 : kt], Γ_k = γ[(k + 1)t - 1 : kt]. \]

Then it holds

\[ α = \sum_{k=0}^{m-1} A_k 2^{kt}, β = \sum_{k=0}^{m-1} B_k 2^{kt}, \text{ and } γ = \sum_{k=0}^{m-1} Γ_k 2^{kt}. \]

For g = (g_1, g_0), h = (h_1, h_0) ∈ F_2^2, A, B, Γ ∈ F_2^t, let

\[ S_g(A, B ↦→ Γ) = \{ (X, Y) \mid (X ⊕ Y ⊕ g_1) ⊕ (X' ⊕ Y' ⊕ g_0) = Γ \}, \]

and

\[ S^h_g(A, B ↦→ Γ) = \{ (X, Y) \mid (X ⊕ Y ⊕ g_1) ⊕ (X' ⊕ Y' ⊕ g_0) = Γ; c'_1 = h_1, c'_0 = h_0 \}. \]
where \( X' = X \oplus A, Y' = Y \oplus B \), and \( c'_t, c'_t^* \) are computed according to Formula (1). Furthermore, let

\[
P_S^h(A, B \mapsto \Gamma) = 2^{-2|S^h(A, B \mapsto \Gamma)|},
\]

and

\[
P_S^b(A, B \mapsto \Gamma) = 2^{-2|S^b(A, B \mapsto \Gamma)|}.
\]

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

**Theorem 1.** Let \( n = mt, x, y, \alpha, \beta, \gamma \in \mathbb{F}_2^n, z = x \oplus y, \) and \( \alpha, \beta, \gamma \) are the XOR differences of \( x, y, z \) respectively. Then

\[
P(\alpha, \beta \mapsto \gamma) = \sum_{i=0}^{m-1} \prod_{k=0}^{m-2} P_{S_g^{k+1}}(A_k, B_k \mapsto \Gamma_k) P_{S_{g_{m-1}}}(A_{m-1}, B_{m-1} \mapsto \Gamma_{m-1}),
\]

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

**Proof.** First, we have

\[
\{(x, y) \mid (x \oplus y) \oplus ((x \oplus \alpha) \oplus (y \oplus \beta)) = \gamma \}
= \bigcup_{i=1, g_i \in \mathbb{F}_2^n} \{(x, y) \mid (x \oplus y) \oplus ((x \oplus \alpha) \oplus (y \oplus \beta)) = \gamma, (c_i, c_i') = g_i, \text{ for } 1 \leq i \leq m-1 \}
= \bigcup_{i=0, g_i \in T_i} \{(x, y) \mid (x \oplus y) \oplus ((x \oplus \alpha) \oplus (y \oplus \beta)) = \gamma, (c_i, c_i') = g_i, \text{ for } 0 \leq i \leq m-1 \},
\]

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

\[
\sum_{i=0}^{m-1} \Gamma_k 2^{kt} = (x \oplus y) \oplus ((x \oplus \alpha) \oplus (y \oplus \beta)) = \gamma
= \sum_{k=0}^{m-1} ((X_k \oplus Y_k \oplus c_{kt}) \oplus ((X_k \oplus A_k) \oplus (Y_k \oplus B_k) \oplus c'_k)) 2^{kt},
\]

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

\[
(X_k \oplus Y_k \oplus c_{kt}) \oplus (X'_k \oplus Y'_k \oplus c'_k) = \Gamma_k,
\]

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

\[
P(\alpha, \beta \mapsto \gamma) = 2^{-2n} |\{(x, y) \mid (x \oplus y) \oplus ((x \oplus \alpha) \oplus (y \oplus \beta)) = \gamma \}|
= \sum_{i=0}^{m-1} \prod_{k=0}^{m-2} 2^{-2t} |\{(X_k, Y_k) \mid (X_k \oplus Y_k \oplus g_k[1]) \oplus (X'_k \oplus Y'_k \oplus g_k[0]) = \Gamma_k, (c_i^*, c_i'^*) = g_{k+1} \}|
\times 2^{-2t} |\{(X_{m-1}, Y_{m-1}) \mid (X_{m-1} \oplus Y_{m-1} \oplus g_{m-1}[1]) \oplus (X'_{m-1} \oplus Y'_{m-1} \oplus g_{m-1}[0]) = \Gamma_{m-1} \}|
= \sum_{i=0}^{m-1} \prod_{k=0}^{m-2} 2^{-2t} |S_{g_k}^{k+1}(A_k, B_k \mapsto \Gamma_k)| \times 2^{-2t} |S_{g_{m-1}}(A_{m-1}, B_{m-1} \mapsto \Gamma_{m-1})|
= \sum_{i=0}^{m-1} \prod_{k=0}^{m-2} P_{S_{g_k}^{k+1}}(A_k, B_k \mapsto \Gamma_k) P_{S_{g_{m-1}}}(A_{m-1}, B_{m-1} \mapsto \Gamma_{m-1}),
\]

where \( c_i^*, c_i'^* \) 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 $PS^h_y(A, B \mapsto \Gamma)$ for $g, h \in \mathbb{F}_2^n$ and 4 tables of $PS_y(A, B \mapsto \Gamma)$ for $g \in \mathbb{F}_2^n$. 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 $z = x \oplus y$. 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 $\mu, \nu$ and $\omega$ 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 $g, h \in \mathbb{F}_2^n$, let

$$S^h_y = \{(X, Y) \in (\mathbb{F}_2^n)^2 \mid c^i_t[X \oplus Y \oplus g] = h\},$$

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

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

Let $S_y = S^0_y \cup S^1_y$, and

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

Then we have the following result.

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

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

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

**Proof.** Note that

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

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

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

$$= 2^{-2n} \sum_{i=0}^{m-1} \prod_{k=0}^{m-2} \left( \sum_{x, y \in S_{g_0, \ldots, g_{m-1}}} (-1)^{\mu \cdot (x \oplus y) \oplus \nu \cdot x \oplus \omega \cdot y} \right)$$

$$= \sum_{i=0}^{m-1} \prod_{k=0}^{m-2} 2^{-2t} \left( \sum_{X_k, Y_k \in S^{g_{k+1}}_{g_{k+1}}} (-1)^{U_k \cdot (X_k \oplus Y_k \oplus g_k) \oplus V_k \cdot X_k \oplus W_k \cdot Y_k} \right)$$

$$\times 2^{-2t} \sum_{X_{m-1}, Y_{m-1} \in S_{g_{m-1}}} (-1)^{U_{m-1} \cdot (X_{m-1} \oplus Y_{m-1} \oplus g_{m-1}) \oplus V_{m-1} \cdot X_{m-1} \oplus W_{m-1} \cdot Y_{m-1}}$$

$$= \sum_{i=0}^{m-1} \prod_{k=0}^{m-2} C^{g_{k+1}}(U_k, V_k, W_k) \cdot C_{g_{m-1}}(U_{m-1}, V_{m-1}, W_{m-1}).$$
Algorithm 1 Log-time algorithm for $P(\alpha, \beta \mapsto \gamma)$

**Input:** $(\alpha, \beta, \gamma)$

**Output:** $P(\alpha, \beta \mapsto \gamma)$

1. if $eq(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \land (\alpha \oplus \beta \oplus \gamma \oplus (\beta \ll 1)) \neq 0, \text{ then}$
2. \hspace{1em} return 0;
3. else \hspace{1em} return $2^{\text{wt}(eq(\alpha, \beta, \gamma) \land \text{mask}(n-1))}$;
4. end if

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

$$f^{a,b,c}(A, B, \Gamma) = eq((A \ll 1) \oplus a, (B \ll 1) \oplus b, (\Gamma \ll 1) \oplus c) \land (A \oplus B \oplus \Gamma \oplus ((B \ll 1) \oplus b)),$$

where

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

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

1. Define $A_{t-1}, B_{t-1}, \Gamma_{t-1}$ as 0. \hspace{1em} Then
   $$eq(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \land (\alpha \oplus \beta \oplus \gamma \oplus (\beta \ll 1)) = 0_n$$
   if and only if
   $$f^{h(A_{k-1}), h(B_{k-1}), h(\Gamma_{k-1})}(A_k, B_k, \Gamma_k) = 0_t$$
   for $0 \leq k \leq m - 1$.
2. Let $t_k = t$ for $0 \leq k \leq m - 2$ and $t_{m-1} = t - 1$. Then
   $$\text{wt}(eq(\alpha, \beta, \gamma) \land \text{mask}(n-1)) = \sum_{k=0}^{m-1} \text{wt}(eq(A_k, B_k, \Gamma_k) \land \text{mask}(t_k)).$$
Proof. Note that \( \alpha = (A_{m-1}, A_{m-2}, \ldots, A_0) = \sum_{k=0}^{m-1} A_k 2^{kt} \), then
\[
\alpha \ll 1 = (\alpha_{n-2}, \ldots, \alpha_0, 0) = \sum_{k=0}^{m-1} (\alpha_{(k+1)t-2}, \ldots, \alpha_{kt}, \alpha_{kt-1}) 2^{kt} = \sum_{k=0}^{m-1} ((A_k \ll 1) \oplus h(A_{k-1})) 2^{kt},
\]
where \( (X \ll 1) \oplus e = (x_{t-2}, \ldots, x_0, e) \) 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, and this implies
\[
\alpha_{-1} = h(A_{-1}) = 0.
\]

Similarly, we have
\[
\beta \ll 1 = \sum_{k=0}^{m-1} ((B_k \ll 1) \oplus h(B_{k-1})) 2^{kt},\quad \gamma \ll 1 = \sum_{k=0}^{m-1} ((I_k \ll 1) \oplus h(I_{k-1})) 2^{kt}.
\]

Then it holds
\[
eq (\alpha \otimes 1) \wedge (\beta \otimes 1) \wedge (\gamma \otimes 1) = \sum_{k=0}^{m-1} f^{h(A_{k-1}), h(B_{k-1}), h(I_{k-1})}(A_k, B_k, I_k) 2^{kt}.
\]

Therefore, the first item holds. The last item is easy to prove, since that
\[
\text{mask}(n-1) = \sum_{k=0}^{n-2} 2^k = \sum_{k=0}^{m-2} (2^t - 1) 2^{kt} + (2^t - 1) 2^{(m-1)t} = \sum_{k=0}^{m-1} (2^t - 1) 2^{kt} = \sum_{k=0}^{m-1} \text{mask}(t_k) 2^{kt},
\]
where \( t_k = t \) for \( 0 \leq k \leq m - 2 \) 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 \( n = mt \), \( \alpha, \beta, \gamma \in \mathbb{F}_2^t \), \( A_k = \alpha[(k + 1)t - 1 : kt] \), \( B_k = \beta[(k + 1)t - 1 : kt] \), and \( I_k = \gamma[(k + 1)t - 1 : kt] \), \( 0 \leq k \leq m - 1 \). For \( A, B, I \in \mathbb{F}_2^t \), \( a, b, c, d \in \mathbb{F}_2 \), let
\[
P_{A,B,\Gamma}^{a,b,c}(A, B, \Gamma) = \begin{cases} 2^{-w_t(\text{eq}(A,B,\Gamma) \wedge \text{mask}(t_d))} & f^{a,b,c}(A, B, \Gamma) = 0; \\ 0, & \text{else}, \end{cases}
\]
where \( t_0 = t \) and \( t_1 = t - 1 \). Then
\[
P(\alpha, \beta \mapsto \gamma) = \prod_{k=0}^{m-1} P_{A,B,\Gamma}^{h(A_{k-1}), h(B_{k-1}), h(I_{k-1})}(A_k, B_k \mapsto I_k),
\]
where \( d_k = 0 \) for \( 0 \leq k \leq m - 2 \), \( d_{m-1} = 1 \), and \( A_{-1}, B_{-1}, I_{-1} \) are defined as 0.

Due to the two different values of \( t_d \), there are two types of CDDTs in the above theorem, which are
\[
P_{A,B,\Gamma}^{a,b,c}(A, B \mapsto \Gamma)
\]
and
\[
P_{A,B,\Gamma}^{a,b,c}(A, B \mapsto \Gamma)
\]
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 \( a, b, c \) is 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.

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 \( P_{A,B,\Gamma}^{a,b,c} \) are pre-constructed. Given two input differences \( \alpha \) and \( \beta \) of addition modulo \( 2^n \), we can compute all possible output differences \( \gamma \) and their probabilities as follows: Firstly, we divide the \( n \)-bit input differences \( \alpha \) and \( \beta \) into \( m \) portions \( A_k \) and \( B_k \) \( (0 \leq k \leq m - 1) \), and each portion contains \( t \) bits (assume \( n = mt \)). Then, we look up the CDDTs \( P_{A,B,\Gamma}^{a,b,c} \) to get all possible output differences \( I_k \) for \( k = 0 \) to \( m - 2 \), and \( PS_{A,B,\Gamma}^{a,b,c} \) to get all possible output differences
Algorithm 2 Constructing CDDT with Lipmaa-Moriai’s algorithm

1: \( \hat{N} = |a|b|c \)
2: for \( N = 0 \) to \( 2^4 - 1 \) do
3:    for \( A, B, \Gamma = 0 \) to \( 2^4 - 1 \) do
4:        \( \text{flag} := \text{eq}(A \ll 1) \oplus a, (B \ll 1) \oplus b, (\Gamma \ll 1) \oplus c \) \( \land \) \( (A \oplus B \oplus \Gamma \oplus ((B \ll 1) \oplus b)) \);
5:        if \( \text{flag} = 0 \), then
6:            \( \text{PS}^N_0(A, B \mapsto \Gamma) = 2^{-\text{wt}(\{A, B, \Gamma\} \land \text{mask}(t))} \);
7:        else
8:            \( \text{PS}^N_0(A, B \mapsto \Gamma) = 0; \)
9:        end if
10:    end for
11: end for
12: end for

\( \Gamma'_{m-1} \). When looking up the CDDTs for \( A_k \) and \( B_k \), we can get all possible output differences \( \Gamma_k \) and their probabilities \( \text{PS}_{\alpha,\beta}^{A_k,B_k}(A_k, B_k \mapsto \Gamma_k) \) 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 \( \gamma \) by concatenating \( \Gamma_k \) \( (0 \leq k \leq m - 1) \), and the corresponding differential probabilities \( P(\alpha, \beta \mapsto \gamma) \) by multiplying \( \text{PS}_{\alpha,\beta}^{A_k,B_k}(A_k, B_k \mapsto \Gamma_k) \).

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 \( \alpha \) and \( \beta \), one can build a system of equations from the condition
\[
eq \text{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \land (\alpha \oplus \beta \oplus \gamma \oplus (\beta \ll 1)) = 0_n,
\]
and then get all possible output differences \( \gamma \) 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 [41], 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 \( \text{CCZ} \)-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”
\[
x = (x_{n-1}, \ldots, x_0) \mapsto M_n(x) = (x_{n-2} \oplus \ldots \oplus x_0, \ldots, x_1 \oplus x_{n-1} \oplus x_0, 0).
\]

Let \( z := M_n^T(\mu \oplus \nu \oplus \omega) \), then
\[
C(\mu, \nu, \omega) = 1_{\mu \oplus \nu \leq z} 1_{\mu \oplus \omega \leq z} (-1)^{\mu \oplus \nu \oplus \omega} 2^{-\text{wt}(z)},
\]
where \( 1_G \) is the indicator function of the graph \( G_f := \{(x, f(x)) \mid x \in \mathbb{F}_2^n \} \), and \( M_n^T(x_{n-1}, \ldots, x_0) = (0, x_{n-1} \oplus x_{n-2}, \ldots, x_{n-1} \oplus \cdots \oplus x_1) \).

According to Theorem 4, it is easy to compute the correlation \( C(\mu, \nu, \omega) \) given the input mask \( \nu, \omega \) and the output mask \( \mu \). 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 $U = (u_{t-1}, \ldots, u_0) \in \mathbb{F}_2^n$ and $e \in \mathbb{F}_2$, let $U \oplus e = (e \oplus u_{t-1}, \ldots, e \oplus u_0)$. For $X = (x_{n-1}, \ldots, x_0) \in \mathbb{F}_2^n$, let $X_k = (x_{(k+1)t-1}, \ldots, x_{kt}) \in \mathbb{F}_2^m$, $0 \leq k \leq m - 1$, and $e_k = \bigoplus_{i=kt}^{(k+1)t-1} x_i$.

Then it holds
$$M^T_t(x_{n-1}, \ldots, x_0) = \{0, x_{n-1}, x_{n-1} \oplus x_{n-2}, \ldots, x_{n-1} \oplus \cdots \oplus x_1\} = (M^T_t(X_{n-1}), M^T_t(X_{m-2}) \oplus e_{m-1}, \ldots, M^T_t(X_0) \oplus e_1^t).$$

For $\mu, \nu, \omega \in \mathbb{F}_2^n$, let $x = \mu \oplus \nu \oplus \omega$, $z = M^T_t(x)$, $e_k = \bigoplus_{i=kt}^{(k+1)t-1} x_i$, $0 \leq k \leq m - 1$, and $e_m = 0$. Then we have
$$1_{\{\mu \oplus \nu \leq z\}} 1_{\{\mu \oplus \omega \leq z\}} = 1$$
if and only if
$$1_{\{u_k \oplus v_k \leq z_k\}} 1_{\{u_k \oplus w_k \leq z_k\}} = 1$$
for $0 \leq k \leq m - 1$, where $Z_k = M^T_t(U_k \oplus V_k \oplus W_k) \oplus e_{k+1}^t$. Furthermore, it also holds
$$wt(z) = \sum_{k=0}^{m-1} wt(Z_k).$$

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 $W_k = \omega[(k + 1)t - 1 : kt]$, $0 \leq k \leq m - 1$. For $U, V, W \in \mathbb{F}_2^n$, $e \in \mathbb{F}_2$, let
$$C_{Le}(U, V, W) = 1_{\{U \oplus V \leq Z\}} 1_{\{U \oplus W \leq Z\}} 2^{-wt(Z)},$$
where $Z = M^T_t(U \oplus V \oplus W) \oplus e^t$. Let $\sigma = (\mu \oplus \nu \oplus \omega)$, $e_m = 0$, and $e_k = (\bigoplus_{i=kt}^{(k+1)t-1} \sigma_i) \oplus e_{k+1}\mu$ 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: if $M_t$ is the left shifted “partial sums mapping”
2: for $e = 0$ to $1$
3: for $U, V, W = 0$ to $2^t - 1$
4: $Z = M^T_t(U \oplus V \oplus W) \oplus e^t$;
5: if $U \oplus V \leq Z$ and $U \oplus W \leq Z$ then
6: $CL_{e}(U, V, W) = 2^{-wt(Z)}$;
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 $\nu$ and $\omega$ of addition modulo $2^n$, we can compute all possible output masks $\mu$ and their correlations as follows: Firstly, we divide the $n$-bit input masks $\nu$ and $\omega$ 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 $e_k = \bigoplus_{i=kt}^{(k+1)t-1} (U_k \oplus V_k \oplus W_k)[i] \oplus e_{k+1}$ which is used to look up the next CLAT. Lastly, after looking up the $m$ CLATs, we get all possible output masks $\mu$ 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 \( B_n \) 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.

Fig. 2. The spread of differential values

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 \( B_n \) 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.

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 (resp. 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.

In the following, we give a rough estimation of the complexity of the differential search algorithm. Let \( m_1 \) be the number of differences \( \alpha_1, \beta_1 \) and \( \gamma_1 \) in the first round, where \( m_1 = \#\{ (\alpha_1, \beta_1 \mapsto \gamma_1) \mid P(\alpha_1, \beta_1 \mapsto \gamma_1) \geq B_n / B_{n-1} \} \). 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
Algorithm 4 Matsui Search for Differential Trails Using CDDTs

1: Procedure Main:
2: Begin the program
3: Let $B_n = 2 \times B_{n-1}$, and $B_n = 1$.
4: Do
5: Let $B_n = 2^{-1} \times B_n$;
6: Call Procedure Round-1;
7: while $B_n \neq B_n$.
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 $p_1 \times B_{n-1} \geq B_n$, 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 $p_i = P(\Delta X_i \mapsto \Delta Z_i)$;
18: If $p_1 \times p_2 \times \cdots \times p_n \times B_{n-1} \geq B_n$, 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 $p_1 \times p_2 \times \cdots \times p_n = B_n$, 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.

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. 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:

\[
F_0(x) = (x \ll 1) \oplus (x \ll 2) \oplus (x \ll 7),
\]

\[
F_1(x) = (x \ll 3) \oplus (x \ll 4) \oplus (x \ll 6).
\]

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 SPECK2n/mn.

All experiments are performed on a PC with a single core (Intel® Core™ i5 – 4570 CPU 3.2GHz). We implicitly assume the independence of inputs when computing the differential probability and correlation of addition.
SPECK has a structure similar to Threefish and utilizes a simple round function consisting of three operations: addition ($\oplus$), XOR ($\oplus$) and rotation ($\ll$). Its round function is defined as:

$$F(x, y) = (x \gg \alpha) \oplus y.$$  

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

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

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 11-round differential trail for HIGHT with probability $2^{-48}$, 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 logarithm of the probability of a differential trail obtained as the sum of the probabilities of its transitions.

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

<table>
<thead>
<tr>
<th>Rounds</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGHT</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-8</td>
<td>-11</td>
<td>-15</td>
<td>-19</td>
<td>-26</td>
<td>-30</td>
<td>-38</td>
<td>$\geq$ -48</td>
</tr>
</tbody>
</table>

Table 4. Probabilities of the optimal differential trails for SPECK. The probabilities are given as $\log_2 p$ ("$\geq$" indicates a lower bound).

<table>
<thead>
<tr>
<th>Rounds</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECK32</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-5</td>
<td>-9</td>
<td>-13</td>
<td>-18</td>
<td>-24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK48</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-6</td>
<td>-10</td>
<td>-14</td>
<td>-19</td>
<td>-26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK64</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-6</td>
<td>-10</td>
<td>-15</td>
<td>-21</td>
<td>-29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK96</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-6</td>
<td>-10</td>
<td>-15</td>
<td>-21</td>
<td>-30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK128</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-6</td>
<td>-10</td>
<td>-15</td>
<td>-21</td>
<td>-30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rounds</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK32</td>
<td>$\geq$ -30</td>
<td>$\geq$ -34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK48</td>
<td>$\geq$ -33</td>
<td>$\geq$ -40</td>
<td>$\geq$ -45</td>
<td>$\geq$ -49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK64</td>
<td>$\geq$ -34</td>
<td>$\geq$ -38</td>
<td>$\geq$ -42</td>
<td>$\geq$ -46</td>
<td>$\geq$ -50</td>
<td>$\geq$ -56</td>
<td>$\geq$ -62</td>
<td>$\geq$ -70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK96</td>
<td>$\geq$ -39</td>
<td>$\geq$ -49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECK128</td>
<td>$\geq$ -39</td>
<td>$\geq$ -49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $2^{-20}$, 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, $\sum_r \log_2 c_r$ represents the logarithm of the absolute correlation of a linear trail obtained as the sum of the correlations of its transitions.

Table 5. Correlations of the optimal linear trails for HIGHT. The correlations are given as $\log_2 c$ ("$\geq$" indicates a lower bound).

<table>
<thead>
<tr>
<th>Rounds</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGHT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-6</td>
<td>-8</td>
<td>-12</td>
<td>-15</td>
<td>$\geq$ -20</td>
</tr>
</tbody>
</table>

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.

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

<table>
<thead>
<tr>
<th>Rounds</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECK32</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−3</td>
<td>−5</td>
<td>−7</td>
<td>−9</td>
<td>−12</td>
<td>−14</td>
<td>−17</td>
<td>−19</td>
</tr>
<tr>
<td>SPECK48</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−3</td>
<td>−6</td>
<td>−8</td>
<td>−12</td>
<td>−15</td>
<td>−19</td>
<td>−22</td>
<td>−25</td>
</tr>
<tr>
<td>SPECK64</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−3</td>
<td>−6</td>
<td>−9</td>
<td>−13</td>
<td>−17</td>
<td>−19</td>
<td>−21</td>
<td>−24</td>
</tr>
<tr>
<td>SPECK96</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−3</td>
<td>−6</td>
<td>−9</td>
<td>−13</td>
<td>−18</td>
<td>−22</td>
<td>−22</td>
<td>−22</td>
</tr>
<tr>
<td>SPECK128</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−3</td>
<td>−6</td>
<td>−9</td>
<td>−13</td>
<td>−18</td>
<td>−22</td>
<td>−22</td>
<td>−22</td>
</tr>
<tr>
<td>Rounds</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>SPECK32</td>
<td>−20</td>
<td>−22</td>
<td>−24</td>
<td>−26</td>
<td>−28</td>
<td>−30</td>
<td>−34</td>
<td>−36</td>
<td>−38</td>
<td>−40</td>
<td>−42</td>
</tr>
<tr>
<td>SPECK48</td>
<td>−28</td>
<td>−30</td>
<td>−32</td>
<td>−34</td>
<td>−36</td>
<td>−38</td>
<td>−40</td>
<td>−42</td>
<td>−44</td>
<td>−46</td>
<td>−48</td>
</tr>
<tr>
<td>SPECK64</td>
<td>−27</td>
<td>−30</td>
<td>−33</td>
<td>−36</td>
<td>−39</td>
<td>−42</td>
<td>−45</td>
<td>−48</td>
<td>−51</td>
<td>−54</td>
<td>−57</td>
</tr>
</tbody>
</table>

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


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 \( x, y, \alpha, \beta, \gamma \in \mathbb{F}_2^4 \), \( z = x \oplus y \), and \( \alpha, \beta, \gamma \) 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 \( x \oplus y \) and \( (x \oplus \alpha) \oplus (y \oplus \beta) \) 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 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

\[
\begin{align*}
x &= x_3x_2x_1x_0, & X_1 &= x_3x_2, & X_0 &= x_1x_0, \\
y &= y_3y_2y_1y_0, & Y_1 &= y_3y_2, & Y_0 &= y_1y_0, \\
z &= z_3z_2z_1z_0, & Z_1 &= z_3z_2, & Z_0 &= z_1z_0, \\
\alpha &= \alpha_3\alpha_2\alpha_1\alpha_0, & A_1 &= \alpha_3\alpha_2, & A_0 &= \alpha_1\alpha_0, \\
\beta &= \beta_3\beta_2\beta_1\beta_0, & B_1 &= \beta_3\beta_2, & B_0 &= \beta_1\beta_0, \\
\gamma &= \gamma_3\gamma_2\gamma_1\gamma_0, & \Gamma_1 &= \gamma_3\gamma_2, & \Gamma_0 &= \gamma_1\gamma_0.
\end{align*}
\]

Then \((A_0, B_0, \Gamma_0)\) are the input and output difference of the CDDT \( PS_0^d \), and \((A_1, B_1, \Gamma_1)\) 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,

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

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

$$P(\alpha, \beta \mapsto \gamma) = \sum_{d \in \mathbb{F}_2^2} PS_d^0(A_0, B_0 \mapsto \Gamma_0) PS_d(A_1, B_1 \mapsto \Gamma_1)$$

$$= PS_0^0(1001 \mapsto 01) PS_0(1011 \mapsto 01) + PS_1^0(1001 \mapsto 01) PS_1(1011 \mapsto 01)$$
$$+ PS_2^0(1001 \mapsto 01) PS_2(1011 \mapsto 01) + PS_3^0(1001 \mapsto 01) PS_3(1011 \mapsto 01)$$

$$= 1/8 \times 1/2 + 1/8 \times 0 + 1/8 \times 0 + 1/8 \times 1/2$$

$$= 1/8.$$

**Algorithm 5** Constructing CDDT $PS_d^0$

1: for $A_0, B_0, \Gamma_0 = 0$ to 3 do
2: for $X_0, Y_0 = 0$ to 3 do
3: $Z_0 = X_0 \parallel Y_0$;
4: $Z_0 = (X_0 \oplus A_0) \parallel (Y_0 \oplus B_0)$;
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: end for
13: $PS_d^0(A_0, B_0 \mapsto \Gamma_0) = 2^{-4} \times S^0_d(A_0, B_0 \mapsto \Gamma_0)$;
14: 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: $Z_1 = X_1 \parallel Y_1 \parallel c_2$;
5: $Z_1' = (Z_1 \oplus A_1) \parallel (Y_1 \oplus B_1) \parallel c_2'$;
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 $P_{S_0}$

<table>
<thead>
<tr>
<th>$P_{S_0}$</th>
<th>$P_{S_1}$</th>
<th>$P_{S_2}$</th>
<th>$P_{S_3}$</th>
<th>$P_{S_4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>0 5/8 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>5/8 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 0 3/8 0 1/8</td>
<td>0 0 0 1/8</td>
<td>0 0 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
</tr>
<tr>
<td>2 0 1/8 0 0</td>
<td>0 0 1/4 0</td>
<td>0 0 1/4 0</td>
<td>0 1/4 0</td>
<td>0 1/4 0</td>
</tr>
<tr>
<td>3 0 1/8 0 1/4</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td></td>
</tr>
<tr>
<td>4 0 1/8 0 1/8</td>
<td>0 0 0 1/8</td>
<td>0 0 0 1/8</td>
<td>0 0 0 1/8</td>
<td>0 1/8 0 1/8</td>
</tr>
<tr>
<td>5 1/8 0 1/8 0</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td></td>
</tr>
<tr>
<td>6 0 1/8 0 1/4</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td></td>
</tr>
<tr>
<td>7 1/8 0 1/4 0</td>
<td>1/8 0 1/8 0</td>
<td>1/8 0 1/8 0</td>
<td>1/8 0 1/8 0</td>
<td></td>
</tr>
<tr>
<td>8 0 0 1/8 0</td>
<td>0 0 1/4 0</td>
<td>0 0 1/4 0</td>
<td>0 0 1/4 0</td>
<td>0 1/8 0</td>
</tr>
<tr>
<td>9 0 1/8 0 1/4</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td></td>
</tr>
<tr>
<td>10 3/8 0 0 0</td>
<td>1/4 0 0 0</td>
<td>1/4 0 0 0</td>
<td>1/4 0 0 0</td>
<td>1/4 0 0 0</td>
</tr>
<tr>
<td>11 0 1/4 0 0</td>
<td>0 1/8 0 1/4</td>
<td>0 1/8 0 1/4</td>
<td>0 1/8 0 1/4</td>
<td></td>
</tr>
<tr>
<td>12 0 1/8 0 1/4</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td>0 1/8 0 1/8</td>
<td></td>
</tr>
<tr>
<td>13 1/8 0 1/4 0</td>
<td>1/8 0 1/8 0</td>
<td>1/8 0 1/8 0</td>
<td>1/8 0 1/8 0</td>
<td></td>
</tr>
<tr>
<td>14 0 1/4 0 0</td>
<td>0 1/8 0 1/4</td>
<td>0 1/8 0 1/4</td>
<td>0 1/8 0 1/4</td>
<td></td>
</tr>
<tr>
<td>15 1/4 0 0 0</td>
<td>1/8 0 1/4 0</td>
<td>1/8 0 1/4 0</td>
<td>1/8 0 1/4 0</td>
<td></td>
</tr>
</tbody>
</table>

Table 8. The CDDT $P_{S_d}$

<table>
<thead>
<tr>
<th>$P_{S_0}$</th>
<th>$P_{S_1}$</th>
<th>$P_{S_2}$</th>
<th>$P_{S_3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 0 0</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>1 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>2 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>3 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>4 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>5 1/2 0 1/2 0</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>1/2 0 1/2</td>
</tr>
<tr>
<td>6 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>7 1/2 0 1/2 0</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
<td>1/2 0 1/2</td>
</tr>
<tr>
<td>8 0 0 1 0</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>9 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>10 1 0 0 0</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>11 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>12 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>13 1/2 0 1/2 0</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
<td>1/2 0 1/2</td>
</tr>
<tr>
<td>14 0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
<td>0 1/2 0 1/2</td>
</tr>
<tr>
<td>15 1/2 0 1/2 0</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>1/2 0 1/2</td>
</tr>
</tbody>
</table>

B Differential Trails for HIGHT and SPECK

Table 9. Differential Trail for HIGHT

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\Delta X_7$</th>
<th>$\Delta X_6$</th>
<th>$\Delta X_5$</th>
<th>$\Delta X_4$</th>
<th>$\Delta X_3$</th>
<th>$\Delta X_2$</th>
<th>$\Delta X_1$</th>
<th>$\Delta X_0$</th>
<th>$\log_2 p_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>20</td>
<td>−6</td>
</tr>
<tr>
<td>1</td>
<td>4a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>$b_8$</td>
<td>−7</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_8$</td>
<td>$c_8$</td>
<td>−4</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$c_8$</td>
<td>1</td>
<td>−4</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>−3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−2</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>9c</td>
<td>82</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−3</td>
</tr>
<tr>
<td>8</td>
<td>9c</td>
<td>7a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>−8</td>
</tr>
<tr>
<td>9</td>
<td>7a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$c_8$</td>
<td>3</td>
<td>$b_c$</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>$c_8$</td>
<td>0</td>
<td>$b_c$</td>
<td>$f_8$</td>
<td>−6</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>90</td>
<td>2</td>
<td>$c_8$</td>
<td>0</td>
<td>$b_6$</td>
<td>$f_8$</td>
<td>1</td>
<td>−5</td>
</tr>
</tbody>
</table>

$\sum_r \log_2 p_r = −48$
### Table 10. Differential Trails for SPECK

<table>
<thead>
<tr>
<th>r</th>
<th>∆L</th>
<th>∆R</th>
<th>log₂p_r</th>
<th></th>
<th>∆L</th>
<th>∆R</th>
<th>log₂p_r</th>
<th></th>
<th>∆L</th>
<th>∆R</th>
<th>log₂p_r</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50</td>
<td>8402</td>
<td>-3</td>
<td>1202</td>
<td>20002</td>
<td>-3</td>
<td>4004092</td>
<td>10420040</td>
<td>-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2402</td>
<td>3408</td>
<td>-3</td>
<td>1000000</td>
<td>1000</td>
<td>-1</td>
<td>900000</td>
<td>1000</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>50c</td>
<td>80c</td>
<td>-7</td>
<td>0</td>
<td>800000</td>
<td>-1</td>
<td>800000</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>181</td>
<td>203</td>
<td>-4</td>
<td>800000</td>
<td>800000</td>
<td>-1</td>
<td>800000</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>800</td>
<td>-5</td>
<td>808004</td>
<td>808020</td>
<td>-2</td>
<td>80</td>
<td>80</td>
<td>-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>0</td>
<td>-3</td>
<td>8400a</td>
<td>801a4</td>
<td>-4</td>
<td>800000</td>
<td>800000</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>40</td>
<td>-1</td>
<td>608da</td>
<td>60880</td>
<td>-9</td>
<td>80480</td>
<td>802084</td>
<td>-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8040</td>
<td>8140</td>
<td>-1</td>
<td>42003</td>
<td>2400</td>
<td>-11</td>
<td>80806080</td>
<td>848164a0</td>
<td>-6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>542</td>
<td>-2</td>
<td>12020</td>
<td>20</td>
<td>-5</td>
<td>40/2400</td>
<td>20040104</td>
<td>-13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8542</td>
<td>904a</td>
<td>-4</td>
<td>200100</td>
<td>200000</td>
<td>-3</td>
<td>20000820</td>
<td>20200001</td>
<td>-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2020</td>
<td>0</td>
<td>-3</td>
<td>900000</td>
<td>0</td>
<td>-2</td>
<td>800000</td>
<td>0</td>
<td>-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>210020</td>
<td>200021</td>
<td>-4</td>
<td>800000</td>
<td>0</td>
<td>-2</td>
<td>800000</td>
<td>0</td>
<td>-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8000</td>
<td>10</td>
<td>-4</td>
<td>900000</td>
<td>0</td>
<td>-2</td>
<td>800000</td>
<td>0</td>
<td>-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>80</td>
<td>0</td>
<td>-2</td>
<td>80</td>
<td>0</td>
<td>-2</td>
<td>80</td>
<td>0</td>
<td>-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>800000000000</td>
<td>800000000000</td>
<td>-1</td>
<td>808000000000</td>
<td>808000000000</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>808000000000</td>
<td>808000000000</td>
<td>-1</td>
<td>808000000000</td>
<td>808000000000</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 11. Linear trail for HIGHT

<table>
<thead>
<tr>
<th>r</th>
<th>ΓX_r</th>
<th>ΓX_r</th>
<th>ΓX_r</th>
<th>ΓX_r</th>
<th>ΓX_r</th>
<th>ΓX_r</th>
<th>ΓX_r</th>
<th>log₂c_r</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>61</td>
<td>0</td>
<td>80</td>
<td>61</td>
<td>b</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>61</td>
<td>0</td>
<td>80</td>
<td>61</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>67</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>c2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>f6</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>36</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>f4</td>
<td>15</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### C. Linear Trails for HIGHT and SPECK
## Table 12. Linear Trails for SPECK

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\Gamma_L$</th>
<th>$\Gamma_R$</th>
<th>$\log_2 c_r$</th>
<th>$\Gamma_L$</th>
<th>$\Gamma_R$</th>
<th>$\log_2 c_r$</th>
<th>$\Gamma_L$</th>
<th>$\Gamma_R$</th>
<th>$\log_2 c_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>280</td>
<td>5226</td>
<td>−0</td>
<td>900</td>
<td>20018c</td>
<td>−0</td>
<td>101000</td>
<td>1013</td>
<td>−0</td>
</tr>
<tr>
<td>1</td>
<td>4880</td>
<td>4885</td>
<td>−1</td>
<td>6000</td>
<td>6008c</td>
<td>−1</td>
<td>1800</td>
<td>18</td>
<td>−2</td>
</tr>
<tr>
<td>2</td>
<td>2000</td>
<td>2071</td>
<td>−2</td>
<td>30200</td>
<td>30240c</td>
<td>−1</td>
<td>0</td>
<td>1</td>
<td>−1</td>
</tr>
<tr>
<td>3</td>
<td>4000</td>
<td>49c1</td>
<td>−2</td>
<td>49c06</td>
<td>49c06c</td>
<td>−2</td>
<td>88000000c</td>
<td>88000000c</td>
<td>−2</td>
</tr>
<tr>
<td>4</td>
<td>516100</td>
<td>501000</td>
<td>−3</td>
<td>50c000c</td>
<td>4c694c</td>
<td>−5</td>
<td>13001000c</td>
<td>310601c</td>
<td>−5</td>
</tr>
<tr>
<td>5</td>
<td>604100</td>
<td>60486c</td>
<td>−3</td>
<td>604100c</td>
<td>60486c</td>
<td>−3</td>
<td>4065000000c</td>
<td>4065000000c</td>
<td>−3</td>
</tr>
</tbody>
</table>

\[ \sum \log_2 c_r = -14 \] for SPECK32, SPECK48, SPECK64

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\Gamma_L$</th>
<th>$\Gamma_R$</th>
<th>$\log_2 c_r$</th>
<th>$\Gamma_L$</th>
<th>$\Gamma_R$</th>
<th>$\log_2 c_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>180001320</td>
<td>4000000018021</td>
<td>−2</td>
<td>180001320</td>
<td>4000000018021</td>
<td>−2</td>
</tr>
<tr>
<td>1</td>
<td>18100</td>
<td>20000000101</td>
<td>−2</td>
<td>18100</td>
<td>20000000101</td>
<td>−2</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>1</td>
<td>−1</td>
<td>100</td>
<td>1</td>
<td>−1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>−0</td>
<td>1</td>
<td>0</td>
<td>−0</td>
</tr>
<tr>
<td>4</td>
<td>d0000000000c</td>
<td>d0000000000c</td>
<td>−1</td>
<td>d0000000000c</td>
<td>d0000000000c</td>
<td>−1</td>
</tr>
<tr>
<td>5</td>
<td>604000000000c</td>
<td>604000000000c</td>
<td>−4</td>
<td>6040000000000c</td>
<td>6040000000000c</td>
<td>−4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>60000000003</td>
<td>−8</td>
<td>3</td>
<td>60000000003</td>
<td>−8</td>
</tr>
<tr>
<td>7</td>
<td>1800000000180</td>
<td>1800000000180</td>
<td>−1</td>
<td>1800000000180</td>
<td>1800000000180</td>
<td>−1</td>
</tr>
<tr>
<td>8</td>
<td>9000000000c0</td>
<td>1880000000c0</td>
<td>−2</td>
<td>9000000000c0</td>
<td>1880000000c0</td>
<td>−2</td>
</tr>
<tr>
<td>9</td>
<td>406500000060c</td>
<td>406500000060c</td>
<td>−3</td>
<td>406500000060c</td>
<td>406500000060c</td>
<td>−3</td>
</tr>
</tbody>
</table>

\[ \sum \log_2 c_r = -22 \] for SPECK96, SPECK128