Automatic Search for Linear Trails of the SPECK Family

Yuan Yao[1,2](✉), Bin Zhang[1], and Wenling Wu[1]

[1] TCA Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China
{yaoyuan,zhangbin,wwl}@tca.iscas.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, China

Abstract. SPECK is a lightweight block cipher family designed by the U.S. National Security Agency and published in 2013. Although several cryptanalyses have been applied since then, no linear results have been proposed. In this paper, we apply Wallén's enumeration algorithm to Matsui's branch-and-bound framework and find the best correlations of SPECK reduced to various rounds, i.e. full rounds of SPECK-32 and 7/ 5/ 4/ 4 rounds of SPECK-48/ 64/ 96/ 128. Since the best 10-round correlation of SPECK-32 is as small as 2^{-17} already, SPECK-32 is immune to the 1-dimensional linear cryptanalysis. Moreover, we present several distinguishers and key recovery attacks as an application of the linear trails. Besides the search for linear trails, we also discuss possible implementations of the Wallén's algorithm and provide an implementation which is faster than the straightforward implementations.

Keywords: Automatic search · Linear cryptanalysis · SPECK · Modulo addition

1 Introduction

The SPECK family [1] is based on a Feistel-like structure and belongs to the ARX ciphers, i.e. primitives composed of modulo addition, bitwise rotation and bitwise XOR only. It is designed to provide optimal software performance on resource constrained devices and is comprised of five variants according to the block size. Despite of its simple structure, no cryptanalysis has threatened its security and particularly no linear cryptanalysis has been proposed due to the intrinsic property of modulo addition. The best previously published attacks are the improved differential cryptanalysis provided by Dinur at SAC 2014 [4].

Generally, good linear trails/approximations should be found in advance in order to launch linear attacks. A widely used approach to search for linear trails of block ciphers is the general framework proposed by Matsui at EUROCRYPT 1994 [8] and it is straightforward to apply as long as the linear approximation

This work is supported by the National Basic Research Program of China (No. 2013CB338002).

table (LAT) of sub-components is obtained. However, the complexity to compute the LAT varies greatly from cipher to cipher. In particular, the time/memory complexity of addition modulo 2^n is $O(2^{3n})$ for a plain enumeration which is nearly impractical even with $n = 16$. Whereas the problem exists in the search for differential trails as well, Biryukov [2] has recently proposed a technique using partial differential distribution tables, called the *threshold search*, and successfully conquered this problem. Fortunately, Wallén has already provided an efficient algorithm to enumerate the LAT of modulo addition at FSE 2003 [10], thus linear approximations could be generated on the fly until it is necessary. The algorithm is further rediscovered in [9] using another approach and its efficiency has been proved by the application to SNOW 2.0 [9] and SOSEMANUK [3]. In case of possible confusions, it should be noted that another algorithm which determines the correlation of a given linear approximation with $O(\log(n))$ time was presented by Wallén in [10] as well. As the latter algorithm is never used in this paper, the Wallén's algorithm in this paper always refers to the algorithm to enumerate the LAT.

By combining Wallén's algorithm and Matsui's branch-and-bound framework, we are able to find the best linear trail of SPECK-32 of full rounds and the best linear trail of SPECK-48/ 64/ 96/ 128 reduced to 7/ 5/ 4/ 4 rounds respectively, shown in Tables 1 and 2 where "$\geq$" denotes a lower bound of the best correlation. Since the data complexity of a 1-dimensional linear cryptanalysis is inversely proportional to the square of the correlation, the best 10-round correlation in Table 1 suggests that SPECK-32 is secure under this method. Indeed, the data complexity of a 1-dimensional linear cryptanalysis against SPECK-32 using the 10-round linear trail is 2^{34}, greater than the size of the code book which is 2^{32}. Moreover, we provide several distinguishers and key recovery attacks as an application of the linear trails. Yet, they do not pose a threat to SPECK and are worse than the differential cryptanalyses of Dinur. After all, this is the first linear cryptanalysis against the SPECK family, evaluating the security in a different perspective.

We additionally find a set of necessary conditions for correlations to be nonzero which allows us to develop an alternative implementation of Wallén's algorithm. According to experiments, this implementation is faster than straightforward implementations derived from the Wallén's theorem and thus useful when called for a tremendous number of times.

The rest of this paper is organized as follows. Section 2 introduces SPECK, Matsui's branch-and-bound framework and the previous Wallén's results on lin-

Table 1. Best correlations for SPECK-32

Rounds(r)	1	2	3	4	5	6	7	8	9	10	11
$\lvert B[r]\rvert$	1	1	2^{-1}	2^{-3}	2^{-5}	2^{-7}	2^{-9}	2^{-12}	2^{-14}	2^{-17}	2^{-19}
Rounds(r)	12	13	14	15	16	17	18	19	20	21	22
$\lvert B[r]\rvert$	2^{-20}	2^{-22}	2^{-24}	2^{-26}	2^{-28}	2^{-30}	2^{-34}	2^{-36}	2^{-38}	2^{-40}	2^{-42}

Table 2. Best correlations for SPECK48/ 64/ 96/ 128 ("≥" indicates a lower bound)

$\lvert B[r] \rvert$		Rounds(r)											
		1	2	3	4	5	6	7	8	9	10	11	12
Block length	48	1	1	2^{-1}	2^{-3}	2^{-6}	2^{-8}	2^{-12}	$\geq 2^{-17}$	$\geq 2^{-20}$	$\geq 2^{-25}$		
	64	1	1	2^{-1}	2^{-3}	2^{-6}	$\geq 2^{-10}$	$\geq 2^{-14}$	$\geq 2^{-17}$	$\geq 2^{-19}$	$\geq 2^{-21}$	$\geq 2^{-25}$	$\geq 2^{-31}$
	96	1	1	2^{-1}	2^{-3}	$\geq 2^{-6}$	$\geq 2^{-11}$						
	128	1	1	2^{-1}	2^{-3}	$\geq 2^{-6}$	$\geq 2^{-11}$						

ear approximation of modulo addition. Section 3 describes the search for linear trails on SPECK and the cryptanalytic results. Section 4 provides the alternative implementation of Wallén's algorithm. Finally, Sect. 5 draws conclusions.

2 Preliminaries

2.1 Notions

a_i	the i-th least-significant bit of word $\boldsymbol{a}$, i.e. $\boldsymbol{a} = (a_{n-1}, \cdots, a_0)$
$\mathbf{1}$	the word $(1, \ldots, 1)$
null	a special word of length zero, i.e. ( )
$\Vert$	the concatenation operation
$\boldsymbol{a} \cdot \boldsymbol{b}$	the inner product of $\boldsymbol{a}, \boldsymbol{b}$
$\boldsymbol{ab}$	the bitwise AND of $\boldsymbol{a}, \boldsymbol{b}$
$\underset{n}{\boxplus}$	the addition modulo 2^n and n is omitted if it is clear from the context
$\underset{n}{\boxminus}$	the subtraction modulo 2^n and n is omitted if it is clear from the context
$\Pr_D(\boldsymbol{y})$	the probability to be $\boldsymbol{y}$ given the probability distribution function D

2.2 Description of SPECK

SPECK is a family of block ciphers containing five variants according to the block size which can be further divided into ten variants regarding the key size. Each variant has two constants ς, τ depending on the block size, i.e. $\varsigma = 7, \tau = 2$ for SPECK-32 and $\varsigma = 8, \tau = 3$ otherwise. The i-th round function (Fig. 1) is defined by

$$\boldsymbol{x}[i+1] \leftarrow ((\boldsymbol{x}[i] \ggg \varsigma) \boxplus \boldsymbol{y}[i]) \oplus \boldsymbol{k}[i]$$
$$\boldsymbol{y}[i+1] \leftarrow (\boldsymbol{y}[i] \lll \tau) \oplus \boldsymbol{x}[i+1]$$

where $\boldsymbol{x}[i]$ and $\boldsymbol{y}[i]$ denote the left and right block of the input respectively, and $\boldsymbol{k}[i]$ is the round-key. The key schedule algorithm is omitted since it is irrelevant to the search, but it should be noted that the master key can be recovered with $^{2\times\text{key length}}/_{\text{block size}}$ successive round-keys. For more details, please refer to [1].

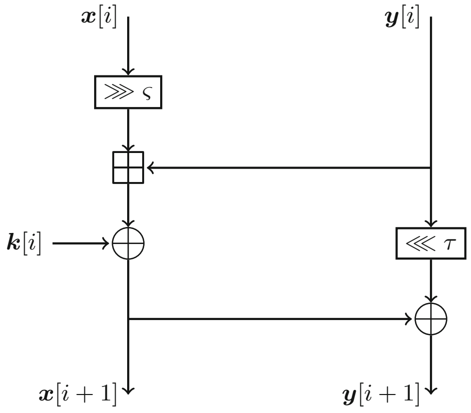


Fig. 1. The round function of SPECK

2.3 Automatic Search Framework

The following is an introduction to the general branch-and-bound search framework proposed by Matsui at EUROCRYPT 1994 [8] in the language of linear cryptanalysis.

To find the best correlation of r successive rounds $B[r]$, the framework performs a recursive search from the knowledge of shorter rounds $B[1], \dots, B[r-1]$ and an initial estimate $\hat{B}[r]$ such that $|\hat{B}[r]| < |B[r]|$. In the search phase, an s-round trail is kept only if

$$|B[r-s]\prod_{i=1}^{s} c[i]| > |\hat{B}[r]|, 1 \leq s \leq r$$

where $c[i]$ denotes the correlation of the i-th round and $B[0]$ is defined to be 1. $\hat{B}[r]$ is updated once the correlation of a r-round trail is better than $\hat{B}[r]$. Therefore, $B[r] = \hat{B}[r]$ when the search completes. Algorithm 1 is an overview where *Get_Mask* is a cipher dependent function to extend linear trails.

2.4 Linear Approximation of Modulo Addition

In this subsection, we briefly introduce Wallén's results on linear approximations of addition modulo 2^n in [10,11].

Definition 1 (Correlation). *Let $\boldsymbol{u}$ be the output mask of the modulo addition and $\boldsymbol{v}, \boldsymbol{w}$ be the input masks. Then the correlation is defined by*

$$c(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \triangleq 2\Pr(\boldsymbol{u} \cdot (\boldsymbol{Z}_1 \boxplus \boldsymbol{Z}_2) \oplus \boldsymbol{v} \cdot \boldsymbol{Z}_1 \oplus \boldsymbol{w} \cdot \boldsymbol{Z}_2 = 0) - 1$$

where $\boldsymbol{Z}_1, \boldsymbol{Z}_2$ are independent uniform distributed random variables.

Algorithm 1. Matsui Search for The Best Linear Trail

```
1: function SEARCH($B$, $T = \{\}$)    ▷ $T = \{T[1], \ldots, T[s]\}$ denotes the linear trail
2:     $r \leftarrow$ SIZEOF($B$) $- 1, s \leftarrow$ SIZEOF($T$)
3:     if $s = r$ then
4:         $\hat{B}[r] \leftarrow \prod_{i=1}^{r} c[i]$
5:     else
6:         for $T'$ in GET_MASK($T$) do    ▷ Extend $T$ to $(s+1)$-round linear trails
7:             if $|B[r-(s+1)] \prod_{i=1}^{s+1} c'[i]| > |\hat{B}[r]|$ then    ▷ $c'[i]$ is the correlation of $T'[i]$
8:                 SEARCH($B$, $T'$)
9:             else
10:                return ▷ Pruning, supposing that $T'$s are enumerated in decreasing order
11:            end if
12:        end for
13:    end if
14: end function
```

The Enumeration Algorithm.

Theorem 1. [9, 11] *Let* $S^0(0,0) \triangleq \{null\}$, $S^0(n,k) = S^1(n,k) \triangleq \emptyset$ *when* $k < 0$ *or* $k \geq n > 0$, *and*

$$S^0(n,k) \triangleq \left(S^0(n-1,k) \parallel \{0\}\right) \cup \left(S^1(n-1,k-1) \parallel \{1,2,4,7\}\right) \tag{1}$$

$$S^1(n,k) \triangleq \left(S^0(n-1,k) \parallel \{7\}\right) \cup \left(S^1(n-1,k-1) \parallel \{0,3,5,6\}\right) \tag{2}$$

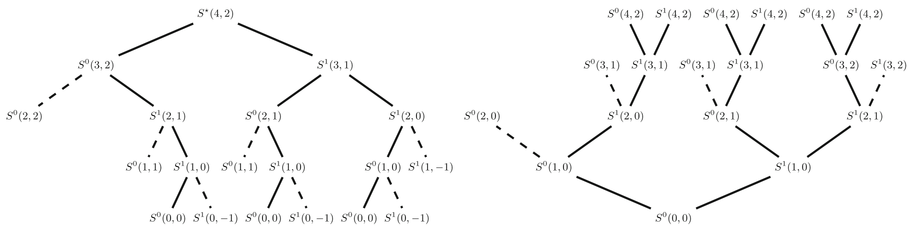
otherwise, where $S^\star \parallel \Omega \triangleq \{\boldsymbol{a} \parallel \boldsymbol{b} \mid \boldsymbol{a} \in S^\star, \boldsymbol{b} \in \Omega\}$. *Then*

$$S(n,k) \triangleq \big\{(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) \mid 4u_i + 2v_i + w_i = s_i, i = 0, \ldots, n-1, \\ \boldsymbol{s} \in S^0(n,k) \cup S^1(n,k)\big\}$$

is the set of all masks such that $c(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) = \pm 2^{-k}$.

Example 1. $S^0(n,0) = \{(0\cdots0)\}, S^1(n,0) = \{(0\cdots07)\}$, thus $S(n,0) = \{((0\cdots 0),(0\cdots0),(0\cdots0)),((0\cdots01),(0\cdots01),(0\cdots01))\}$ is the set of all masks such that $c(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) = \pm 1$.

As was pointed out by Wallén, the LAT of addition modulo 2^n can be enumerated using $O(n)$ space via Theorem 1. A trivial implementation, called the *top-down* method in this paper, can be deduced as shown in Fig. 2(a) and Appendix A.1. However, it is inefficient in the sense that the same subtree will be generated for multiple times. Another possible implementation is the *bottom-up* method which is shown in Fig. 2(b) and Appendix A.2, i.e. starting from $S^0(0,0)$ and then computing $S(1,0)$ etc. While it also generates duplicate subtrees, surprisingly it is faster than the top-down method. (See Fig. 5 for the comparison)



(a) The Top-down Method (b) The Bottom-up Method

Fig. 2. The computational process of $S(4,2)$

Common Prefix Mask vs. Correlation. This subsection serves for the alternative implementation of the above algorithm and may be skipped safely to understand the search.

Definition 2 (CPM). *Let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^n$. If $n = 2$, the common prefix mask of $\boldsymbol{a}, \boldsymbol{b}$ is defined by*

$$cpm_2(\boldsymbol{a}, \boldsymbol{b}) = a_1$$

If $n > 2$, the common prefix mask of $\boldsymbol{a}, \boldsymbol{b}$ is defined by

$$cpm_n(\boldsymbol{a}, \boldsymbol{b}) = a_{n-1} \parallel cpm_{n-1}\left((a_{n-2} \oplus a_{n-1} \cdot b_{n-2}) \parallel \boldsymbol{a}', 1 \parallel \boldsymbol{b}'\right)$$

where $\boldsymbol{a}' = (a_{n-3}, \dots, a_0)$ and $\boldsymbol{b}' = (b_{n-3}, \dots, b_0)$.[1]

Lemma 1. *[10] Let $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{F}_2^n$ be defined as in Definition 1, $\boldsymbol{\phi} = \boldsymbol{v} \oplus \boldsymbol{u}, \boldsymbol{\varphi} = \boldsymbol{w} \oplus \boldsymbol{u}$ be the input masks of the carry function, $\boldsymbol{\gamma} = \boldsymbol{v} \oplus \boldsymbol{w}$ and $\boldsymbol{\delta} = cpm_{n+1}(0 \parallel \boldsymbol{u}, (0 \parallel \boldsymbol{\gamma}) \oplus \mathbf{1})$. Then*

$$c(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) = \begin{cases} (-1)^{wt(\boldsymbol{\delta}\boldsymbol{\phi}\boldsymbol{\varphi})} 2^{-wt(\boldsymbol{\delta})}, & \text{if } \boldsymbol{\phi} = \boldsymbol{\phi}\boldsymbol{\delta} \text{ and } \boldsymbol{\varphi} = \boldsymbol{\varphi}\boldsymbol{\delta} \\ 0, & \text{otherwise} \end{cases}$$

where wt is the hamming weight.

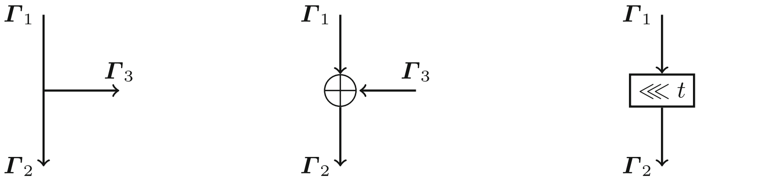
Example 2. Suppose $\boldsymbol{u} = (1100), \boldsymbol{v} = \boldsymbol{w} = (1000)$, then $\boldsymbol{\phi} = \boldsymbol{\varphi} = (0100)$, $\boldsymbol{\gamma} = (0000)$ and $\boldsymbol{\delta} = (0100)$. Thus, $c(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) = -2^{-1}$.

3 Linear Results on SPECK

3.1 Details of the Search

In this section, we will concentrate on the design of *Get_Mask* which will be used to extend linear trails by Algorithm 1. Firstly, we recall the linear properties of branch, bitwise XOR and bitwise rotation.

[1] This definition is the method proposed by Wallén to calculate the CPM.



(a) Branch (b) Bitwise XOR (c) Bitwise Rotation

Fig. 3. The spread of linear masks

Property 1. Let $\boldsymbol{\Gamma}_1, \boldsymbol{\Gamma}_2, \boldsymbol{\Gamma}_3$ be linear masks defined by Fig. 3(a), then the correlation is nonzero if and only if $\boldsymbol{\Gamma}_1 \oplus \boldsymbol{\Gamma}_2 \oplus \boldsymbol{\Gamma}_3 = \mathbf{0}$.

Property 2. Let $\boldsymbol{\Gamma}_1, \boldsymbol{\Gamma}_2, \boldsymbol{\Gamma}_3$ be linear masks defined by Fig. 3(b) then the correlation is nonzero if and only if $\boldsymbol{\Gamma}_1 = \boldsymbol{\Gamma}_2 = \boldsymbol{\Gamma}_3$.

Property 3. Let $\boldsymbol{\Gamma}_1, \boldsymbol{\Gamma}_2$ be linear masks defined by Fig. 3(c) then the correlation is nonzero if and only if $\boldsymbol{\Gamma}_2 = \boldsymbol{\Gamma}_1 \lll t$.

Let the linear masks of the i-th round be defined in Fig. 4. Accordingly,

$$\begin{aligned}\boldsymbol{u}[i] &= \boldsymbol{X}[i+1] \oplus \boldsymbol{Y}[i+1]\\ \boldsymbol{v}[i] &= \boldsymbol{X}[i] \ggg \varsigma\\ \boldsymbol{w}[i] &= \boldsymbol{Y}[i] \oplus (\boldsymbol{Y}[i+1] \ggg \tau)\end{aligned}$$

Thereupon,

$$\begin{aligned}\boldsymbol{u}[r] &= \boldsymbol{X}[r+1] \oplus \boldsymbol{Y}[r+1]\\ \boldsymbol{u}[r-1] &= (\boldsymbol{v}[r] \lll \varsigma) \oplus \boldsymbol{w}[r] \oplus (\boldsymbol{Y}[r+1] \ggg \tau)\end{aligned}$$

and

$$\begin{aligned}\boldsymbol{u}[i] = (\boldsymbol{v}[i+1] \lll \varsigma) \oplus \boldsymbol{w}[i+1] \oplus&\\ ((\boldsymbol{u}[i+1] \oplus (\boldsymbol{v}[i+2] \lll \varsigma)) \ggg \tau), 1 \leq i \leq r-2&\end{aligned}$$

If we enumerate $\boldsymbol{X}[r+1]$ and $\boldsymbol{Y}[r+1]$ directly, the complexity is at least 2^{2n} and it is a waste of efforts on masks with insignificant correlations at the initial stage. Since $\boldsymbol{X}[r+1], \boldsymbol{Y}[r+1]$ are uniquely determined by $\boldsymbol{u}[r], \boldsymbol{v}[r], \boldsymbol{w}[r], \boldsymbol{u}[r-1]$, it is equivalently and more efficiently to enumerate $\boldsymbol{u}[r], \boldsymbol{v}[r], \boldsymbol{w}[r], \boldsymbol{u}[r-1]$ using the Wallén's algorithm. On the other hand, when $1 \leq i \leq r-2$, $\boldsymbol{u}[i]$ can be deduced from the two following rounds. As a result, we have presented a method to extend linear trails by appending one round to the front and Algorithm 2 is the corresponding implementation of *Get_Mask*.

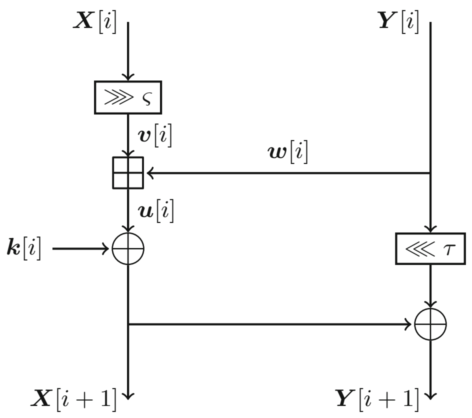


Fig. 4. Masks of the i-th round

Algorithm 2. The Implementation of *Get_Mask*

```
1: function GENERATE()
2:     for $k$ from 0 to $n-1$ do
3:         BU_GENERATE($n, 000, k, \textit{not used}, \textit{not used}, \textit{not used}$)    ▷ bottom-up
   generation of $S(n,k)$ (see Appendix A.2) and 000 means totally free
4:     end for
5: end function

6: function GENERATE($\boldsymbol{u}$)
7:     for $k$ from 0 to $n-1$ do
8:         BU_GENERATE($n, 100, k, \boldsymbol{u}, \textit{not used}, \textit{not used}$)  ▷ bottom-up generation of
   $S(n,k)$ (see Appendix A.2) and 100 means $\boldsymbol{u}$ is supplied and fixed
9:     end for
10: end function

11: function GET_MASK($T$)    ▷ $T = \{T[1], \ldots, T[s]\}$ and $T[r+1-i] = \{\boldsymbol{u}[i], \boldsymbol{v}[i],$
    $\boldsymbol{w}[i]\}$
12:     $s \leftarrow$ SIZEOF($T$), $curr \leftarrow r - s, last \leftarrow curr + 1$
13:     if $s < 2$ then
14:         for $tuple$ in GENERATE() do
15:             if $\boldsymbol{u}[r], \boldsymbol{u}[r-1]$ don't equal $\mathbf{0}$ simultaneously then
16:                 $T[s+1] \leftarrow tuple$ and yield $T$
17:             end if
18:         end for
19:     else
20:         $\boldsymbol{u}[curr] \leftarrow (\boldsymbol{v}[last] \lll \varsigma) \oplus \boldsymbol{w}[last] \oplus ((\boldsymbol{u}[last] \oplus (\boldsymbol{v}[last+1] \lll \varsigma)) \ggg \tau)$
21:         for $tuple$ in GENERATE($\boldsymbol{u}[curr]$) do
22:             $T[s+1] \leftarrow tuple$ and yield $T$
23:         end for
24:     end if
25: end function
```

3.2 Search Results

The automatic search is applied to variants of all block sizes and the best correlations are presented in Tables 1 and 2. Since the quotient of the best correlations of successive rounds is quite regular, $\hat{B}[r]$ is set to $2^{-3}B[r-1]$ for most of the cases. However, not all searches can finish in a reasonable time period due to the huge size of the search space, even with a tight threshold (e.g. $\hat{B}[r] \approx B[r]$). Thus, the "$\geq$" in the Table 2 denotes the best correlation that has been found in this case, i.e. a lower bound.

3.3 Linear Distinguishers

A Linear Distinguisher identifies the nonuniformity of a cipher and generally converts to a hypothesis testing problem using statistical tools. In this and subsequent sections, we make the common assumption that the correlation of a linear approximation can be estimated by the correlation of a significant linear trail. Moreover, the data complexity to distinguish two probability distributions D and D_0 is estimated by $C\left(D, D_0\right)^{-1}$ (see [5] for example) with the capacity

$$C\left(D, D_0\right) \triangleq \textstyle\sum_{\boldsymbol{y} \in \mathcal{Y}} \left(\Pr_D(\boldsymbol{y}) - \Pr_{D_0}(\boldsymbol{y})\right)^2 / \Pr_{D_0}(\boldsymbol{y})$$

Since

$$\begin{aligned}\boldsymbol{x}[i] \cdot \boldsymbol{X}[i] \oplus \boldsymbol{y}[i] \cdot \boldsymbol{Y}[i] = (((\boldsymbol{x}[i-1] \ggg \varsigma) \boxplus \boldsymbol{y}[i-1]) \oplus \boldsymbol{k}[i-1]) \cdot \\ (\boldsymbol{X}[i] \oplus \boldsymbol{Y}[i]) \oplus \boldsymbol{y}[i-1] \cdot (\boldsymbol{Y}[i] \ggg \tau)\end{aligned}$$

and $\boldsymbol{k}[i-1] \cdot (\boldsymbol{X}[i] \oplus \boldsymbol{Y}[i])$ is constant, the *absolute value* of the correlation of

$$\boldsymbol{x}[2] \cdot \boldsymbol{X}[2] \oplus \boldsymbol{y}[2] \cdot \boldsymbol{Y}[2] \oplus \boldsymbol{x}[2+r] \cdot \boldsymbol{X}[2+r] \oplus \boldsymbol{y}[2+r] \cdot \boldsymbol{Y}[2+r]$$

can be calculated from $\boldsymbol{x}[1], \boldsymbol{y}[1], \boldsymbol{x}[2+r], \boldsymbol{y}[2+r]$ without $\boldsymbol{k}[1]$. In other words, a r-round linear trail can be transformed into a $(r+1)$-round linear distinguisher by appending one round to the front. Thus, we immediately obtain the results in Table 3.

Table 3. Linear distinguishers against the SPECK family

Block length	Trail length	Correlation	Rounds	Data
32	9	2^{-14}	10	2^{28}
48	9	2^{-20}	10	2^{40}
64	11	2^{-25}	12	2^{50}
64	12	2^{-31}	13	2^{62}
96	6	2^{-11}	7	2^{22}
128	6	2^{-11}	7	2^{22}

3.4 Key Recovery Attacks

For key recovery attacks, we adopt the χ^2 extension of Matsui's Algorithm 2 which was presented by Hermelin et al. in [6] and does not require the distribution of the linear approximation for the correct key.

Let $h(\boldsymbol{a})$ denote one plus the position of the most-significant one of $\boldsymbol{a}$ and $\boldsymbol{a}^{\boxplus} \triangleq 2^{h(\boldsymbol{a})} - 1$. Because

$$\boldsymbol{a} \underset{n}{\boxplus} \boldsymbol{b} = \boldsymbol{c} \Rightarrow \boldsymbol{a} \underset{h(\boldsymbol{a})}{\boxplus} \boldsymbol{b} = \boldsymbol{c}\boldsymbol{a}^{\boxplus} \Rightarrow \boldsymbol{a}\boldsymbol{a}^{\boxplus} = \boldsymbol{c}\boldsymbol{a}^{\boxplus} \underset{h(\boldsymbol{a})}{\boxminus} \boldsymbol{b}$$
$$\boldsymbol{y}[i] = (\boldsymbol{x}[i+1] \oplus \boldsymbol{y}[i+1]) \ggg \tau$$

guessing $\boldsymbol{k}[i]\boldsymbol{v}[i]^{\boxplus}$ is enough to calculate

$$\begin{aligned}\boldsymbol{x}[i] \cdot \boldsymbol{X}[i] \oplus \boldsymbol{y}[i] \cdot \boldsymbol{Y}[i] &= (\boldsymbol{x}[i] \ggg \varsigma) \cdot \boldsymbol{v}[i] \oplus \boldsymbol{y}[i] \cdot \boldsymbol{Y}[i] \\ &= \left((\boldsymbol{x}[i] \ggg \varsigma)\boldsymbol{v}[i]^{\boxplus}\right) \cdot \boldsymbol{v}[i] \oplus \boldsymbol{y}[i] \cdot \boldsymbol{Y}[i] \\ &= \left(\left(\boldsymbol{x}[i+1]\boldsymbol{v}[i]^{\boxplus} \oplus \boldsymbol{k}[i]\boldsymbol{v}[i]^{\boxplus}\right) \underset{h(\boldsymbol{v}[i])}{\boxminus} \boldsymbol{y}[i]\right) \cdot \boldsymbol{v}[i] \oplus \boldsymbol{y}[i] \cdot \\ &\quad \boldsymbol{Y}[i]\end{aligned}$$

from $\boldsymbol{x}[i+1], \boldsymbol{y}[i+1]$. Therefore, if m rounds are appended to the back of a r-round distinguisher, then only

$$\begin{aligned}h(\boldsymbol{v}[r+1]) + (m-1)n &= h((\boldsymbol{u}[r] \oplus \boldsymbol{Y}[r+1]) \ggg \varsigma) + (m-1)n \\ &= h((\boldsymbol{u}[r] \oplus ((\boldsymbol{w}[r] \oplus \boldsymbol{Y}[r]) \lll \tau)) \ggg \varsigma) + (m-1)n \\ &= h((\boldsymbol{u}[r] \oplus ((\boldsymbol{w}[r] \oplus (\boldsymbol{u}[r-1] \oplus (\boldsymbol{v}[r] \lll \varsigma))) \lll \tau)) \ggg \\ &\quad \varsigma) + (m-1)n\end{aligned}$$

bits of key need to be guessed, i.e. $\boldsymbol{k}[r+1]\boldsymbol{v}[r+1]^{\boxplus}, \boldsymbol{k}[r+2], \ldots, \boldsymbol{k}[r+m]$. Consequently, we have Table 4 where

$$\text{Time} = \text{Data} \times 2^{\text{guessed bits}} + 2^{\text{key length}} \times \beta$$
$$\text{Average Time} = {}^{\text{Time}}/_{1-\alpha}$$

and α, β are missing detection and false alarm probabilities respectively. Moreover, the results may be improved by trails of smaller $h(\boldsymbol{v}[r+1])$ or vectorial linear approximations. But it seems unable to be improved by the similar technique of [4] since the size of the equation derived from a sub-cipher is one bit instead of $2n$ bits in the case of 1-dimensional linear cryptanalysis.

4 Another Implementation of Wallén's Algorithm

In this section, we present another implementation of Wallén's algorithm, called the *CPM* method, and compare the performance of different implementations. Firstly, a set of necessary conditions for correlations to be non-zero needs to be proved.

Table 4. Key recovery attacks on the SPECK family

Block/ key length	Trail length (this paper/ [4])	Rounds (this paper/ [4]/Total)	Guessed bits	α	β	Data (this paper/ [4])	Time	Average time (this paper/ [4])
32/ 64	9/ 10	12/ 14/ 22	$13 + 16$	2^{-1}	2^{-6}	$2^{30.8668}/2^{31}$	$2^{60.2164}$	$2^{61.2164}/2^{63}$
48/ 72	9/ 11	11/ 14/ 22	24	2^{-2}	2^{-7}	$2^{43.727}/2^{41}$	$2^{67.93}$	$2^{68.345}/2^{65}$
48/ 96	9/ 11	12/ 15/ 23	$24 + 24$	2^{-2}	2^{-7}	$2^{43.727}/2^{41}$	$2^{91.93}$	$2^{92.345}/2^{89}$
64/ 96	11/ 15	13/ 18/ 26	31	2^{-2}	2^{-14}	$2^{54.6279}/2^{61}$	$2^{85.7401}$	$2^{86.1551}/2^{93}$
64/ 96	12/ 15	14/ 18/ 26	31	2^{-1}	2^{-2}	$2^{62.7302}/2^{61}$	$2^{94.8714}$	$2^{95.8714}/2^{93}$
64/ 128	11/ 15	14/ 19/ 27	$31 + 32$	2^{-2}	2^{-14}	$2^{54.8029}/2^{61}$	$2^{117.74}$	$2^{118.155}/2^{125}$
64/ 128	12/ 15	15/ 19/ 27	$31 + 32$	2^{-1}	2^{-2}	$2^{62.7302}/2^{61}$	$2^{126.871}$	$2^{127.871}/2^{125}$
96/ 96	6/ 14	8/ 16/ 28	47	2^{-3}	2^{-26}	$2^{27.6463}/2^{85}$	$2^{74.7028}$	$2^{74.8954}/2^{85}$
96/ 144	6/ 14	9/ 17/ 29	$47 + 48$	2^{-3}	2^{-26}	$2^{27.6463}/2^{85}$	$2^{122.703}$	$2^{122.895}/2^{133}$
128/ 128	6/ 15	8/ 17/ 32	63	2^{-5}	2^{-36}	$2^{28.2959}/2^{113}$	$2^{92.6905}$	$2^{92.7363}/2^{113}$
128/ 192	6/ 15	9/ 18/ 33	$63 + 64$	2^{-5}	2^{-36}	$2^{28.2959}/2^{113}$	$2^{156.69}$	$2^{156.736}/2^{177}$
128/ 256	6/ 15	7/ 19/ 34	$63+2\times64$	2^{-5}	2^{-36}	$2^{28.2959}/2^{113}$	$2^{220.69}$	$2^{220.736}/2^{241}$

Lemma 2. *Let $\boldsymbol{u}, \boldsymbol{\gamma}, \boldsymbol{\delta} \in \mathbb{F}_2^n$. Then*

$$\boldsymbol{\delta} = cpm_{n+1}\left(0 \parallel \boldsymbol{u}, (0 \parallel \boldsymbol{\gamma}) \oplus \mathbf{1}\right) \Longleftrightarrow \boldsymbol{\delta} = \left(\boldsymbol{u} \oplus (\boldsymbol{\gamma} \oplus \mathbf{1})\, \boldsymbol{\delta}\right) \gg 1$$

Proof. "$\Longrightarrow$". From Definition 2, it is clear that $\delta_{n-1} = 0$ and $\delta_i = u_{i+1} \oplus (\gamma_{i+1} \oplus 1)\delta_{i+1}, i = 0, \ldots, n-2$.

"$\Longleftarrow$". *Suppose* $\boldsymbol{\delta}' = cpm_{n+1}(0 \parallel \boldsymbol{u}, (0 \parallel \boldsymbol{\gamma}) \oplus \mathbf{1})$, then $\delta'_i = u_{i+1} \oplus (\gamma_{i+1} \oplus 1)\delta'_{i+1}, i = 0, \ldots, n-2$. Thus $\delta_i \oplus \delta'_i = (\gamma_{i+1} \oplus 1)(\delta_{i+1} \oplus \delta'_{i+1}), i = 0, \ldots, n-2$. Finally, $\delta_{n-2} = \delta'_{n-2}, \ldots, \delta_0 = \delta'_0$ following from $\delta_{n-1} = \delta'_{n-1} = 0$. □

Theorem 2. *Let $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{\phi}, \boldsymbol{\varphi}, \boldsymbol{\delta} \in \mathbb{F}_2^n$ and $\boldsymbol{\phi} = \boldsymbol{v} \oplus \boldsymbol{u}, \boldsymbol{\varphi} = \boldsymbol{w} \oplus \boldsymbol{u}, \boldsymbol{\gamma} = \boldsymbol{v} \oplus \boldsymbol{w}$. Then*

$$\boldsymbol{\delta} = cpm_{n+1}(0 \parallel \boldsymbol{u}, (0 \parallel \boldsymbol{\gamma}) \oplus \mathbf{1}), c(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \neq 0$$

if and only if

$$\boldsymbol{\phi} = \boldsymbol{\phi}\boldsymbol{\delta} \tag{3}$$
$$\boldsymbol{\varphi} = \boldsymbol{\varphi}\boldsymbol{\delta} \tag{4}$$
$$\boldsymbol{\gamma} \gg 1 = ((\boldsymbol{u} \oplus \boldsymbol{\delta}) \gg 1) \oplus \boldsymbol{\delta} \tag{5}$$
$$\mathbf{0} = ((\boldsymbol{u} \gg 1) \oplus \boldsymbol{\delta})\, ((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1) \tag{6}$$
$$\mathbf{0} = ((\boldsymbol{v} \gg 1) \oplus \boldsymbol{\delta})\, ((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1) \tag{7}$$
$$\mathbf{0} = ((\boldsymbol{w} \gg 1) \oplus \boldsymbol{\delta})\, ((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1) \tag{8}$$

Proof. Proof of the *only-if-part*. Since $c(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \neq 0$, (3) and (4) follow from Lemma 1 directly. According to Lemma 2, $\boldsymbol{\delta} = (\boldsymbol{u} \oplus (\boldsymbol{\gamma} \oplus \mathbf{1})\boldsymbol{\delta}) \gg 1$. Hence,

$$\begin{aligned}((\boldsymbol{u} \oplus \boldsymbol{\delta}) \gg 1) \oplus \boldsymbol{\delta} &= ((\boldsymbol{u} \oplus \boldsymbol{\delta}) \oplus (\boldsymbol{u} \oplus (\boldsymbol{\gamma} \oplus \mathbf{1})\,\boldsymbol{\delta})) \gg 1 = (\boldsymbol{\gamma}\boldsymbol{\delta}) \gg 1 \\ &= ((\boldsymbol{\phi} \oplus \boldsymbol{\varphi})\,\boldsymbol{\delta}) \gg 1 = (\boldsymbol{\phi} \oplus \boldsymbol{\varphi}) \gg 1 = \boldsymbol{\gamma} \gg 1\end{aligned}$$

Accordingly,

$$\begin{aligned}\mathbf{0} &= \boldsymbol{\gamma}\,(\boldsymbol{\delta} \oplus \mathbf{1}) = (\boldsymbol{\gamma}\,(\boldsymbol{\delta} \oplus \mathbf{1})) \gg 1 = (\boldsymbol{\gamma} \gg 1)\,((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1) \\ &= (((\boldsymbol{u} \oplus \boldsymbol{\delta}) \gg 1) \oplus \boldsymbol{\delta})\,((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1) \\ &= ((\boldsymbol{u} \gg 1) \oplus \boldsymbol{\delta})\,((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1) \oplus ((\boldsymbol{\delta}(\boldsymbol{\delta} \oplus \mathbf{1})) \gg 1) \\ &= ((\boldsymbol{u} \gg 1) \oplus \boldsymbol{\delta})\,((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1)\end{aligned}$$

(3) implies $(\boldsymbol{\phi}\,(\boldsymbol{\delta} \oplus \mathbf{1})) \gg 1 = \mathbf{0}$, thus

$$\begin{aligned}&((\boldsymbol{v} \gg 1) \oplus \boldsymbol{\delta})\,((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1) = \\ &\qquad (\boldsymbol{\phi}\,(\boldsymbol{\delta} \oplus \mathbf{1})) \gg 1 \oplus ((\boldsymbol{u} \gg 1) \oplus \boldsymbol{\delta})\,((\boldsymbol{\delta} \oplus \mathbf{1}) \gg 1) = \mathbf{0}\end{aligned}$$

(8) holds similarly.

Proof of the *if-part*. From (5),

$$\begin{aligned}((\boldsymbol{u} \oplus \boldsymbol{\delta}) \gg 1) \oplus \boldsymbol{\delta} &= \boldsymbol{\gamma} \gg 1 = (\boldsymbol{\gamma}\boldsymbol{\delta}) \gg 1 = (\boldsymbol{u} \oplus \boldsymbol{\gamma}\boldsymbol{\delta} \oplus \boldsymbol{\delta} \oplus \boldsymbol{\delta} \oplus \boldsymbol{u}) \gg 1 \\ &= ((\boldsymbol{u} \oplus (\boldsymbol{\gamma} \oplus \mathbf{1})\,\boldsymbol{\delta}) \gg 1) \oplus ((\boldsymbol{\delta} \oplus \boldsymbol{u}) \gg 1)\end{aligned}$$

Therefore,

$$\boldsymbol{\delta} = (\boldsymbol{u} \oplus (\boldsymbol{\gamma} \oplus \mathbf{1})\,\boldsymbol{\delta}) \gg 1 = cpm_{n+1}\,(0 \parallel \boldsymbol{u}, (0 \parallel \boldsymbol{\gamma}) \oplus \mathbf{1})$$

and the conclusion is derived from Lemma 1. □

We next discusses details of the CPM method under different scenarios.
Case 1: $\boldsymbol{u}$ is known and fixed. Therefore, $\boldsymbol{\delta}$ should satisfy (6) and δ_i is determined by δ_{i+1} for $0 \leq i < n-1$, i.e.

$$\delta_i = \begin{cases} 0, 1 & \text{if } \delta_{i+1} = 1 \\ u_{i+1} & \text{otherwise} \end{cases}$$

Recall that $\delta_{n-1} = 0$, thus $\boldsymbol{\delta}$ can be resolved bit by bit from left to right. But it should be noted that $\boldsymbol{\delta}$ needs to be enumerated in the order of hamming weight according to Lemma 1. We adopt a deque (i.e. a data structure supporting push and pop in both front and back directions) for this purpose, and $\boldsymbol{\delta}$ is pushed to the front whenever $\delta_{i-1} = 0$ and is pushed to the back otherwise. Details are presented in Algorithm 3.

Given $\boldsymbol{u}$ and $\boldsymbol{\delta}$, the approximation is determined by two of $\boldsymbol{v}, \boldsymbol{w}$ and $\boldsymbol{\gamma}$. Obviously, $\boldsymbol{\gamma}$ can be obtained from (5) except γ_0. Thus, the input masks are

Algorithm 3. Generate $\boldsymbol{\delta}$ given $\boldsymbol{u}$

```
1: function CPM_GENERATE_DELTA($\boldsymbol{u}$)
2:     $deque \leftarrow \{(\mathbf{0}, n-1)\}$
3:     while $deque$ is not empty do
4:         $(\boldsymbol{\delta}, i) \leftarrow$ POP_FRONT($deque$)
5:         if $i \neq 0$ then
6:             if $\delta_i = 1$ then
7:                 PUSH_FRONT($deque, (\boldsymbol{\delta}, i-1)$)
8:                 PUSH_BACK($deque, (\boldsymbol{\delta} \oplus (1 \ll (i-1)), (i-1))$)
9:             else if $u_i = 1$ then
10:                PUSH_BACK($deque, (\boldsymbol{\delta} \oplus (1 \ll (i-1)), (i-1))$)
11:            else
12:                PUSH_FRONT($deque, (\boldsymbol{\delta}, i-1)$)
13:            end if
14:        else
15:            yield $\boldsymbol{\delta}$
16:        end if
17:    end while
18: end function
```

known once $\boldsymbol{v}$ or $\boldsymbol{w}$ is generated. Without loss of generality, we choose to generate $\boldsymbol{v}$ and then calculate $\boldsymbol{w}$ as $\boldsymbol{w} = \boldsymbol{v} \oplus \boldsymbol{\gamma}$. According to (3),

$$v_i = \phi_i \oplus u_i = \begin{cases} 0, 1 & \text{if } \delta_i = 1 \\ u_i & \text{otherwise} \end{cases}$$

for $0 \leq i < n$. Hence, the bits of $\boldsymbol{v}$ where $\boldsymbol{\delta}$ equals one need to be traversed to generate all valid masks. As far as we know, the most efficient method to generate all tuples is the Gray code strategy [7] which flips one bit only in each iteration as shown in Appendix B. Also, this step may be customized for special purpose, e.g. generating the tuples by hamming weight. See Algorithm 4 for details.

Case 2: $\boldsymbol{v}$ or $\boldsymbol{w}$ is known and fixed. Suppose $\boldsymbol{v}$ is known, then $\boldsymbol{\delta}$ should satisfy (7). Thus, $\boldsymbol{\delta}$ can be generated using the procedure *CPM_Generate_Delta* with the parameter $\boldsymbol{v}$ and thereupon $\boldsymbol{u}$ can be determined by (6), i.e.

$$u_i = \begin{cases} 0, 1 & \text{if } \delta_i = 1 \\ \delta_{i-1} & \text{otherwise} \end{cases}$$

for $1 \leq i \leq n-1$. Since $\phi_0\delta_0 = \delta_0 = \phi_0 = v_0 \oplus u_0$ according to (3), $u_0 = v_0$ if $\delta_0 = 0$ and $u_0 \in \{0, 1\}$ otherwise. Finally, $\boldsymbol{\gamma}$ and $\boldsymbol{w}$ are determined by (5) and (4) as in Case 1.

Case 3: $\boldsymbol{u}, \boldsymbol{v}$ or $\boldsymbol{u}, \boldsymbol{w}$ are known and fixed. Suppose $\boldsymbol{u}, \boldsymbol{v}$ are known, so $\boldsymbol{\phi} = \boldsymbol{v} \oplus \boldsymbol{u}$ is known as well. And $\boldsymbol{\delta}$ should satisfy (3), (6) and (7). Notice that the conditions may be incompatible and result in zero correlation. Indeed, since $\delta_{n-1} = 0$, $\boldsymbol{\delta}$

Algorithm 4. The case that $\boldsymbol{u}$ is known and fixed

```
1: function CPM_GENERATE_MASK($\boldsymbol{u}$, $\boldsymbol{\delta}$)
2:     $\boldsymbol{\gamma} \leftarrow ((\boldsymbol{u} \oplus (\boldsymbol{\delta} \lll 1)) \oplus \boldsymbol{\delta})\boldsymbol{\delta}$     ▷ $\gamma_0 \in \{0,1\}$ if $\delta_0 = 1$ and $\gamma_0 \in \{0\}$ otherwise
3:     $ones \leftarrow \{0 \le i < n : \delta_i = 1\}$
4:     for $\boldsymbol{v}$ in GRAY_VISIT($\boldsymbol{\delta} \oplus \boldsymbol{u}, ones$) do
5:         $\boldsymbol{w} \leftarrow \boldsymbol{v} \oplus \boldsymbol{\gamma}$
6:         yield $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
7:         if $\delta_0 = 1$ then
8:             yield $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \oplus 1)$     ▷ Equivalent to flipping $\gamma_0$
9:         end if
10:    end for
11: end function

12: function GENERATE'($\boldsymbol{u}$)
13:     for $\boldsymbol{\delta}$ in CPM_GENERATE_DELTA($\boldsymbol{u}$) do
14:         CPM_GENERATE_MASK($\boldsymbol{u}$, $\boldsymbol{\delta}$)
15:     end for
16: end function
```

exists only if $\phi_{n-1} = 0$. By (3) and (6), we have

$$\delta_i = \begin{cases} 0,1 & \text{if } \delta_{i+1} = 1 \text{ and } \phi_i = 0 \\ 1 & \text{if } \delta_{i+1} = 1 \text{ and } \phi_i = 1 \\ 1 & \text{if } \delta_{i+1} = 0 \text{ and } u_{i+1} = 1 \\ 0 & \text{if } \delta_{i+1} = 0 \text{ and } u_{i+1} = \phi_i = 0 \\ \perp & \text{otherwise} \end{cases}$$

for $0 \le i < n-1$ where $\perp$ means no solution. Consequently, $\boldsymbol{\delta}$ can be solved using procedure similar to *CPM_Generate_Delta*. At last, $\boldsymbol{\gamma}$ is resolved by (5) and $\boldsymbol{w} = \boldsymbol{v} \oplus \boldsymbol{\gamma}$.
Case 4: $\boldsymbol{v}, \boldsymbol{w}$ are known. Thus, $\boldsymbol{\gamma}$ is fixed and $\boldsymbol{\delta}$ should satisfy $\boldsymbol{\gamma\delta} = \boldsymbol{\gamma}$, (7) and (8). Similar to Case 3, $\boldsymbol{\delta}$ exists only if $\gamma_{n-1} = 0$, and

$$\delta_i = \begin{cases} 0,1 & \text{if } \delta_{i+1} = 1 \text{ and } \gamma_i = 0 \\ 1 & \text{if } \delta_{i+1} = 1 \text{ and } \gamma_i = 1 \\ 1 & \text{if } \delta_{i+1} = 0 \text{ and } v_{i+1} = 1 \\ 0 & \text{if } \delta_{i+1} = 0 \text{ and } v_{i+1} = \gamma_i = 0 \\ \perp & \text{otherwise} \end{cases}$$

for $0 \le i < n-1$. Then, $\boldsymbol{u}$ is calculated by (5) except that u_0 needs to satisfy

$$(u_0 \oplus v_0)\delta_0 = u_0 \oplus v_0$$
$$(u_0 \oplus v_0)\delta_0 = u_0 \oplus v_0$$

Since $\gamma_0 = v_0 \oplus w_0 = 1 \Rightarrow \delta_0 = 1$, then $u_0 \in \{0,1\}$ if $\delta_0 = 1$ and $u_0 = v_0 = w_0$ otherwise.

Case 5: All Masks are Free. In this case, $\boldsymbol{\delta}$ is generated first according to its hamming weight to ensure the order of approximations. Then, $\boldsymbol{u}$ is obtained as in Case 2 without the constraint on u_0. At last, the procedure *CPM_Generate_Mask* takes over. Refer to Algorithm 5 for details.

Algorithm 5. The case that all masks are free

```
1: function Generate'()
2:     for $k$ from 0 to $n-1$ do
3:         for $\boldsymbol{\delta}$ of weight $k$ do
4:             $ones \leftarrow \{0 < i < n : \delta_i = 1\} \cup \{0\}$
5:             for $\boldsymbol{u}$ in Gray_Visit($\boldsymbol{\delta} \ll 1, ones$) do
6:                 CPM_Generate_Mask($\boldsymbol{u}$, $\boldsymbol{\delta}$)
7:             end for
8:         end for
9:     end for
10: end function
```

Obviously, the CPM method is not as elegant as the top-down/bottom-up method, but surprisingly it is faster for $n \geq 11$ according to Fig. 5 (note that the labels on y-axis increase *exponentially*). We believe better direct techniques to instantiate Theorem 1 exists, but *Generate′* is the most effective implementation we can think of at present and is used to replace *Generate* in Algorithm 2.

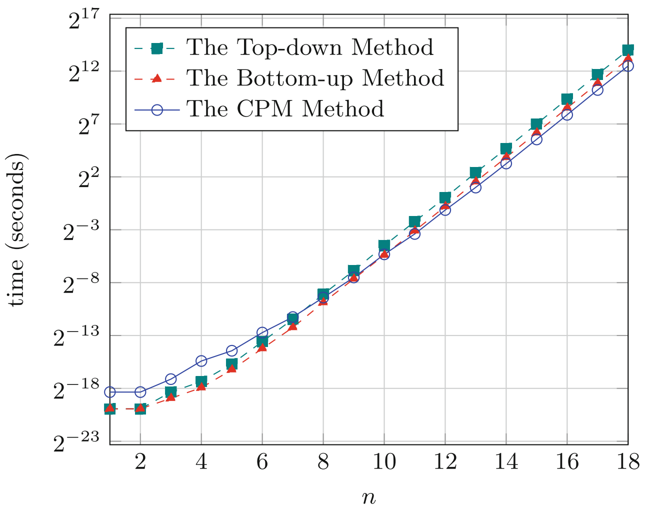


Fig. 5. The performance of generating $\bigcup_{k=0}^{n-1} S(n,k)$ Platform: 32-bit Win7 with Visual C++ 2015 CTP optimized by /Ox

5 Conclusions

In this paper, we presented a search for linear trails on the SPECK family via Wallén's enumeration algorithm and Matsui's branch-and-bound framework. The best correlation of full rounds of SPECK-32 was found as well as reduced rounds of other variants. According to the best 10-round correlation of SPECK-32 which is 2^{-17}, SPECK-32 is immune to the 1-dimensional linear cryptanalysis. We further proposed the first linear distinguishers and key recovery attacks on the SPECK family which do not threaten the security of SPECK. Finally, a CPM implementation of the Wallén's algorithm was presented which seems faster than the straightforward instantiations, i.e. the top-down and the bottom-up approaches.

Additional future work items include applying the threshold search [2] on SPECK, mounting vectorial linear cryptanalyses and implementing the search on other ARX ciphers.

A Straightforward Implementations of Wallén's Algorithm

The *mode* argument indicates whether $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ are fixed and used hereafter.

A.1 The Top-Down Method

```
1:  function TDV($mode, t_{i+1}, t_i, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$)
2:      if $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, t_{i+1}, t_i$ and $mode$ are compatible then
3:          modify the $i$-th bit of $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ and yield $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
4:      end if
5:  end function

6:  function TDD_GENERATE($N, mode, t, n, rk, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$)
7:      if $n = 0$ then
8:          if $t = S0$ then                                        ▷ $S^0(0,0)$
9:              yield $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
10:         end if
11:         return
12:     end if
13:     $i \leftarrow N - n$
14:     if $n \neq rk$ or $rk = 0$ then
15:         if $n = N$ or $t = S0$ then                 ▷ $S^0(n-1, rk) \leftarrow S^0(n, rk)$
16:             for $\boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$ in TDV($mode, S0, S0, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$) do
17:                 TDD_GENERATE($N, mode, S0, n-1, rk, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$)
18:             end for
19:         end if
20:         if $n = N$ or $t = S1$ then                 ▷ $S^0(n-1, rk) \leftarrow S^1(n, rk)$
21:             for $\boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$ in TDV($mode, S0, S1, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$) do
```

```
                TDD_GENERATE($N, mode, S0, n-1, rk, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$)
            end for
        end if
    end if
    if $rk \neq 0$ then
        if $n = N$ or $t = S0$ then        ▷ $S^1(n-1, rk-1) \leftarrow S^0(n, rk)$
            for $\boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$ in TDV($mode, S1, S0, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$) do
                TDD_GENERATE($N, mode, S1, n-1, rk-1, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$)
            end for
        end if
        if $n = N$ or $t = S1$ then        ▷ $S^1(n-1, rk-1) \leftarrow S^1(n, rk)$
            for $\boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$ in TDV($mode, S1, S1, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$) do
                TDD_GENERATE($N, mode, S1, n-1, rk-1, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$)
            end for
        end if
    end if
end function

function TD_GENERATE($n, mode, k, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$)        ▷ generate $S(n, k)$
    TDD_GENERATE($n, mode, not\_used, n, k, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$)        ▷ start from $S(n, k)$
end function
```

A.2 The Bottom-Up Method

```
function BUV($mode, t_i, t_{i-1}, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$)
    if $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, t_i, t_{i-1}$ and $mode$ are compatible then
        modify the $i$-th bit of $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ and yield ($\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$)
    end if
end function

function BUD_GENERATE($N, mode, t, n, rk, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$)
    if $n = N$ then        ▷ $S(N, k)$
        yield ($\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$) and return
    end if
    $i \leftarrow N - 1 - n$
    if $t = S0$ then
        if $i = 0$ or $rk < i$ then        ▷ $S^0(n, k-rk) \rightarrow S^0(n+1, k-rk)$
            for $\boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$ in BUV($mode, S0, S0, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$) do
                BUD_GENERATE($N, mode, S0, n+1, rk, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$)
            end for
        end if
        if $i = 0$ or $rk \neq 0$ then        ▷ $S^0(n, k-rk) \rightarrow S^1(n+1, k-rk)$
            for $\boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$ in BUV($mode, S0, S1, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$) do
                BUD_GENERATE($N, mode, S1, n+1, rk, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$)
            end for
        end if
```

```
22:     else
23:         if $i = 0$ or $rk \le i$ then                ▷ $S^1(n, k - rk) \to S^0(n + 1, k - rk + 1)$
24:             for $\boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$ in BUV$(mode, S1, S0, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ do
25:                 BUD_Generate$(N, mode, S0, n + 1, rk - 1, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}')$
26:             end for
27:         end if
28:         if $i = 0$ or $rk \ne 1$ then                ▷ $S^1(n, k - rk) \to S^1(n + 1, k - rk + 1)$
29:             for $\boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}'$ in BUV$(mode, S1, S1, i, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ do
30:                 BUD_Generate$(N, mode, S1, n + 1, rk - 1, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{w}')$
31:             end for
32:         end if
33:     end if
34: end function

35: function BU_Generate$(n, mode, k, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$                ▷ generate $S(n, k)$
36:     BUD_Generate$(n, mode, S0, 0, k, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$        ▷ start from $S^0(0, k - k)$
37: end function
```

B The Gray_Visit Procedure

```
 1: function Gray_Visit$(\boldsymbol{a}, set)$
 2:     $s \leftarrow$ sizeof$(set), buf \leftarrow \{1, 2, \ldots, s + 1\}$
 3:     while true do
 4:         yield $\boldsymbol{a}$
 5:         $j \leftarrow buf[1], buf[1] \leftarrow 1$
 6:         if $j = s + 1$ then
 7:             return
 8:         end if
 9:         $i \leftarrow j + 1, buf[j] \leftarrow buf[i], buf[i] \leftarrow i$
10:         flip $a[set[j]]$
11:     end while
12: end function
```

References

1. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013). http://eprint.iacr.org/
2. Biryukov, A., Velichkov, V.: Automatic search for differential trails in ARX ciphers. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 227–250. Springer, Heidelberg (2014). http://dx.doi.org/10.1007/978-3-319-04852-9_12
3. Cho, J.Y., Hermelin, M.: Improved linear cryptanalysis of SOSEMANUK. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 101–117. Springer, Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-14423-3_8

4. Dinur, I.: Improved differential cryptanalysis of round-reduced SPECK. Cryptology ePrint Archive, Report 2014/320 (2014). http://eprint.iacr.org/. Accepted by SAC 2014
5. Hermelin, M.: Multidimensional Linear Cryptanalysis. Ph.D. thesis, Aalto University School of Science and Technology, Faculty of Information and Natural Sciences, Department of Information and Computer Science (2003). http://lib.tkk.fi/Diss/2010/isbn9789526031903/isbn9789526031903.pdf
6. Hermelin, M., Cho, J.Y., Nyberg, K.: Multidimensional extension of matsui's algorithm 2. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 209–227. Springer, Heidelberg (2009). http://dx.doi.org/10.1007/978-3-642-03317-9_13
7. Knuth, D.: The Art of Computer Programming: Generating All Tuples and Permutations. Addison-Wesley Series in Computer Science and Information Proceedings, vol. 4. Addison Wesley Publishing Company Incorporated, Upper Saddle River (2005)
8. Matsui, M.: On correlation between the order of S-Boxes and the strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 366–375. Springer, Heidelberg (1995). http://dx.doi.org/10.1007/BFb0053451
9. Nyberg, K., Wallén, J.: Improved linear distinguishers for SNOW 2.0. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 144–162. Springer, Heidelberg (2006). http://dx.doi.org/10.1007/11799313_10
10. Wallén, J.: Linear approximations of addition modulo 2^n. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 261–273. Springer, Heidelberg (2003). http://dx.doi.org/10.1007/978-3-540-39887-5_20
11. Wallén, J.: On the differential and linear properties of addition. Master's thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory for Theoretical Computer Science (2003). http://www.tcs.hut.fi/Publications/bibdb/HUT-TCS-A84.pdf