Differential cryptanalysis in ARX ciphers, Application to SPECK

Ashutosh Dhar Dwivedi, Paweł Morawiecki

Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract

We propose a new algorithm inspired by Nested to find differential path in ARX ciphers. To enhance the decision process of our algorithm and to reduce the search space of our heuristic nested tool, we used the concept of partial difference distribution table (pDDT) along with the algorithm. The algorithm is applied on reduced round variants of SPECK block cipher family. In our previous paper we applied naive algorithm with a large search space of values and presented the result only for one block size variant of SPECK. Our new approach in this paper provide the results within a simpler framework and within a very short period of time (few minutes) for all bigger block size variants of SPECK. More specifically, we report the differential path for up to 8, 9, 11, 10 and 11 rounds of SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128, respectively. To construct a differential characteristics for large number of rounds, we divide long characteristics into short ones, say constructing a large characteristics from two short characteristics. Instead of starting from first round we start from the middle and run experiment in forward as well as reverse direction. Using this method we improved our results and report the differential path for up to 9, 10, 12, 13 and 15 rounds of SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128, respectively.

Keywords: Differential path, Nested Monte-Carlo Search, ARX ciphers, SPECK Cipher, Differential Cryptanalysis

1 Introduction

ARX (Addition/Rotation/XOR) is a class of cryptographic algorithm which uses three simple arithmetic operations: modular addition, bitwise rotation and exclusive-OR. In both industry and academia, ARX cipher has gained more interest and attention in last few years. By using combined linear (XOR, bit shift, bit rotation) and non-linear (modular addition) operation and iterating them for many rounds, ARX algorithms become more resistance against differential and linear cryptanalysis. ARX has a lack of look-up table, associated with S-box based algorithms and therefore increase the resistance against side channel attacks. Due to simplicity of operations, ARX algorithms exhibit excellent performance, especially for software platforms.

In our analysis we focus on SPECK [2]. SPECK is a secure, flexible and light weight block cipher designed by researchers from National Security Agency (NSA) of the USA in June 2013. It has great performance both on software and hardware. Its design is similar to Threefish- the block cipher used in the hash function Skein [7]. SPECK is a pure ARX cipher with a Feistel-like structure in which both branches are modified at every round. SPECK consist of 5 variants SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128 with block sizes 32, 48, 64, 96 and 128 bits, respectively.

The cryptanalysis of ARX design is more difficult. Since a typical S-box consist of 4 or 8-bit words, the differential or linear properties can be evaluated by computing its difference distribution table (DDT) or linear approximation table (LAT) respectively. But in case of ARX, for a 32-bit word it is clearly infeasible to calculate these tables. Although a partial difference distribution table (pDDT) containing few fraction of all differentials that has a probability greater than a fixed threshold is still possible. This is possible due to the fact that the probabilities of XOR (resp. ADD) differentials through the modular addition (resp. XOR) operation are monotonously decreasing with the bit size of the word.

In this paper we propose a method for finding good differential paths in ARX ciphers. Finding a differential trail is a kind of problem where we face huge state space and there is no clear and obvious way how we should make a next step. This kind of problem are also available in many different areas but we were inspired with single-player games such as Morpion solitaire, SameGame and Sudoku. The heuristics called Nested Monte-Carlo Search works very well for these games [6]. We can treat a search for good differential paths also as a single-player game and we argue that this approach could be a base for more sophisticated heuristics. However we modified algorithm depends on the technical complexity of our problem but it is somehow inspired by Nested Monte-Carlo Search.

In our previous paper [1] we already applied naive approach of algorithm to all variant of SPECK but we found good result only for one variant with smallest state size SPECK32. For bigger variants, our algorithm was demanding to reduce the search space to enhance the random decision process and therefore we used the the partial difference distribution table (pDDT) [3] to reduce the search space of our algorithm.

Beside the concept of pDDT we were inspired by the highways and country roads analogy proposed by Biryukov et al [3] [9]. We relate the problem of finding high probability differential trails in a cipher to the problem of finding fast routes between two cities on a road map, then differentials that have high probability (w.r.t. a fixed threshold) can be thought of as highways and conversely differentials with low probability can be viewed as slow roads or country roads. Therefore in our algorithm firstly tries to find a probability which has probability above the threshold probability and if such probability does not exist then it uses the low probability values. Using this concept algorithm do not take completely random decision in iteration and hence it improves the random decision process by using smaller search space.

2 Related Cryptanalysis

Biryukov et al.[4] published a paper where they analyzed ARX cipher SPECK and by introducing the concept of partial difference distribution table (pDDT) they extend Matsuis algorithm, originally proposed for DES-like ciphers, to the class of ARX ciphers. They found differential trail of 9, 10 and 13 rounds for 3 variant SPECK32, SPECK48 and SPECK64, respectively.

Biryukov et al. [5] again presented a paper in FSE 2016 where they propose the adaptation of Matsuis algorithm for finding the best differential and linear trails to the class of ARX ciphers. It was based on a branch-and-bound search strategy, does not use any heuristics and returns optimal results. They report the probabilities of the best differential trails for up to 10, 9, 8, 7 and 7 rounds of SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128, respectively.

Song et al. [10] presented a paper where they develop Mouha et al.'s framework for finding differential characteristics by adding a new method to construct long characteristics from short ones. They report the probabilities of the best differential trails of SPECK for up to 10, 11, 15, 17, and 20 rounds of SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128, respectively.

3 Description of SPECK

SPECK is a family of lightweight block ciphers with the Fiestel-like structure in which each block is divided in two branches and both branches are modified at every round. It has 5 variants, SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128, where a number in the name denotes a block size in bits. Each block size is divided in two parts, left half and right half.

3.1 Round Function

SPECK uses 3 basic operations on n-bit word for each round:

– bitwise XOR, $\oplus$,
– addition modulo 2^n, $\boxplus$
– left and right circular shifts by r_2 and r_1 bits, respectively.

Left half n-bit word is denoted by $X_{r-1,L}$ and right half n-bit word is denoted by $X_{r-1,R}$ to the r-th round and n-bit round key applied in the r-th round is denoted by k_r. $X_{r,L}$ and $X_{r,R}$ denotes output words from round r which are computed as follows:

$$X_{r,L} = ((X_{r-1,L} \ggg r_1) \boxplus X_{r-1,R}) \oplus k_r \tag{1}$$

$$X_{r,R} = ((X_{r-1,R} \lll r_2) \oplus X_{r,L}) \tag{2}$$

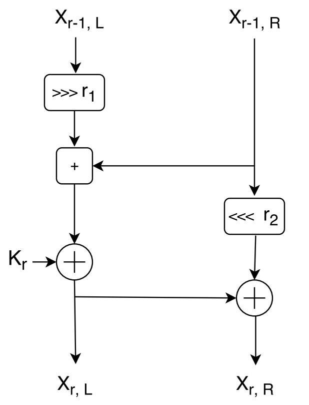

Fig. 1: The round function of SPECK

Different key size have been used by several instances of SPECK family and the total number of rounds depends on key size. The value of rotation constant r_1 and r_2 are specified as: $r_1 = 7$, $r_2 = 2$ for SPECK32 and $r_1 = 8$, $r_2 = 3$ for all other variants. Parameters for all variants represented in the Table 1.

Variant	Block Size(2n)	Word Size(n)	Key Size	Rounds
SPECK32	32	16	64	22
SPECK48	48	24	72	22
			96	23
SPECK64	64	32	96	26
			144	29
SPECK96	96	48	96	28
			144	29
SPECK128	128	64	128	32
			192	33
			256	34

Table 1: SPECK Parameters

4 Calculating Differential Probabilities

In [8], Moriai and Lipmaa studied the differential properties of addition. Let $xdp^+(a, b \to c)$ be the XOR-differential probability of addition modulo 2^n, with input differences a and b and the output difference c. Moriai and Lipmaa proved that the differential $(a, b \to c)$ is valid if and only if:

$$eq(a \ll 1, b \ll 1, c \ll 1) \wedge (a \oplus b \oplus c \oplus (b \ll 1)) = 0 \tag{3}$$

where

$$eq(p, q, r) := (\neg p \oplus q) \wedge (\neg p \oplus r) \tag{4}$$

For every valid differential $(a, b \to c)$, we define the weight $w(a, b \to c)$ of the differential as follows:

$$w(a, b \to c) = -\log_2(xdp^+(a, b \to c)) \tag{5}$$

The weight of a valid differential can then be calculated as:

$$w(a, b \to c) := h(\neg eq(a, b \to c)), \tag{6}$$

where $h(x)$ denotes the number of non-zero bits in x, not counting $x[n-1]$.

A differential characteristic defines not only the input and output differences, but also the internal differences after every round of the iterated cipher. In our analysis, we follow a common assumption that the probability of a valid differential characteristic is equal to the multiplication of the probabilities of each addition operation. The XOR operation and the bit rotation are linear in GF(2), therefore for these two operations for every input difference there is only one valid output difference.

5 Partial Difference Distribution Tables (pDDT)

The Partial difference distribution table (pDDT) proposed by Biryukov et al. [3] is a table that contains all XOR differentials $(a, b \to c)$ whose differential probabilities (DP) are greater than or equal to pre-defined threshold p_{thres}.

$$(a, b, c) \in pDDT \Leftrightarrow DP(a, b \to c) \geq p_{thres} \tag{7}$$

To compute pDDT efficiently we will use following proposition: The differential probability of XOR of addition modulo 2^n is monotonously decreasing with the word size.

$$p_n \leq .... \leq p_k \leq p_{k-1} \leq .... \leq p_1 \leq p_0 \tag{8}$$

where $p_k = DP(a_k, b_k \to c_k), n \geq k \geq 1, p_0 = 1$ and x_k denotes the k LSB's of the difference x that is, $x_k = x[k-1:0]$. The algorithm is defined in a recursive fashion. For each bit position $k : n > k > 0$ check if probability of partially constructed $(k+1) - bit$ differential is greater than threshold p_{thres}. If yes, then move to the next bit, otherwise go back and assign different values to $a[k], b[k]$ and $c[k]$. Repeat the process until $k = n$ and once $k = n$ add $(a_k, b_k \to c_k)$ to the pDDT. The initial value of k is 0 and $a_0, b_0, c_0 = \phi$.

Algorithm 1 Computation of a pDDT for XOR

```
1: Input: $n, p_{thres}, k, p_k, a_k, b_k, c_k$.
2: Output: pDDT $D : (a, b, c) \in$ D : DP$(a, b \rightarrow c) \geq p_{thres}$.
3: function COMPUTEPDDT$(n, p_{thres}, k, p_k, a_k, b_k, c_k)$
4:     if n==k then
5:         Add $(a, b, c) \longleftarrow (a_k, b_k, c_k)$ to D
6:     end if
7:     return D
8:     for $x, y, z \in 0, 1$ do
9:         $a_{k+1} \longleftarrow x|a_k, b_{k+1} \longleftarrow y|b_k, c_{k+1} \longleftarrow z|c_k$
10:        $p_{k+1} = DP(a_{k+1}, b_{k+1} \rightarrow c_{k+1})$
11:        if $p_{k+1} \geq p_{thres}$ then
12:            computepddt$(n, p_{thres}, k + 1, p_{k+1}, a_{k+1}, b_{k+1}, c_{k+1})$
13:        end if
14:    end for
15: end function
```

6 Nested Monte Carlo Search

Our algorithm is inspired by Nested Monte Carlo Search algorithms. The Monte Carlo method is a heuristic based random sampling method. An application to game-tree search based on Monte Carlo method was proposed by Remi Coulom in 2008 named as Monte Carlo Tree Search (MCTS). This algorithm was useful to games where it is hard to formulate an evaluation function, such as the game of Go. Later for a single player games, a variant called Nested Monte Carlo Search has been proposed [6].

Lets take a tree like structure to understand the Nested Monte-Carlo Search algorithm. At each step the NMCS algorithm tries all possible moves and memorizes the move associated to the best score of the lower level searches, that is, a nested of level 1 makes a playout for every possible move and choose to play the move of the best playout. A nested of level 2 does the same thing except that is replaces the playout by a nested of level one.

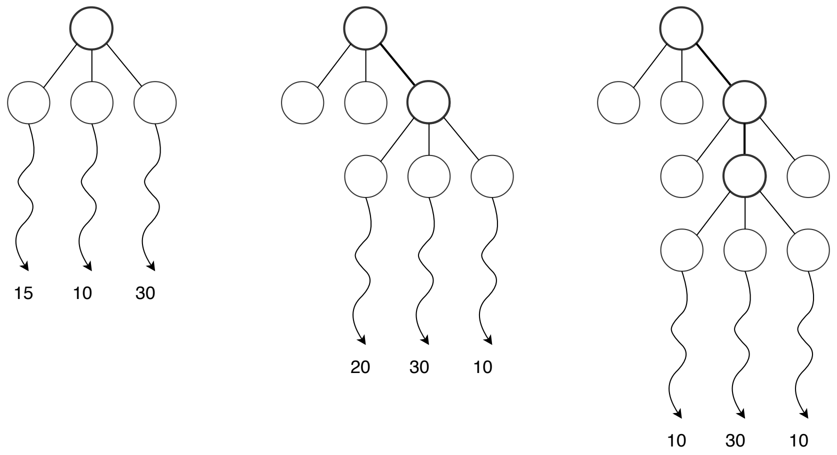

Fig. 2: Nested Monte Carlo Search.

During the first iteration the initial state (root) is selected and for the selected state all legal actions are determined (Figure 2). Therefore at level 0 it play random game for all possible moves valid for selected state (root). Then it moves one step ahead to next level with greatest associated score.

Therefore we changed the original NMCS algorithm to eliminate this problem for our cipher and presented a new algorithm based on NMCS with an example in the next paragraph. Instead of trying all possible moves, we try only one random move.

Problem of finding differential path in a cipher with high probability could be treated as the problem of finding fast routes between two cities on the road map. Let us understand the algorithm with this example. Our goal is to find shortest path from a city to other city. We represent all possible paths as a tree. Root of the tree is considered as starting point while all the leaves are ending point reached by different paths (nodes). Each edge between nodes is associated with a number which represent distance between two nodes. Initially we have two lists named as *BestPath* and*CurrentPath* represent the best available path from previous search and random path which is under investigation, respectively. The last element in both the list represents total distance traveled. Both the lists are initially empty.

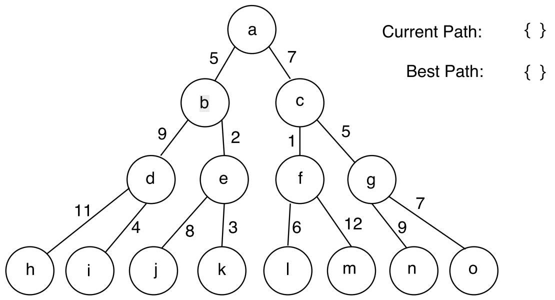

Fig. 3: Different paths from the root (base node) to the destination (leaf nodes)

Initially algorithm takes a random moves from base node to leaf node and save the path in Current Path list. Lets say random path selected by algorithm is $\{a, b, d, i\}$ with distance score 18. Since initially there was no better path available (*BestPath* is empty), then we save the current path and its distance as *BestPath* (See Figure 4).

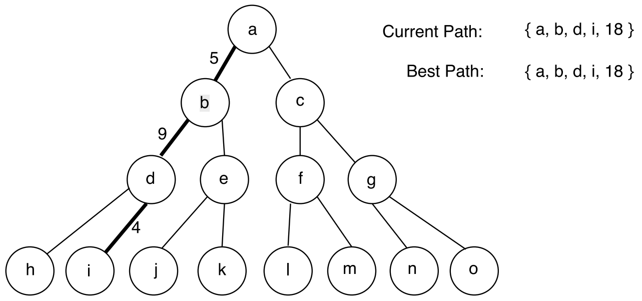
Current Path: { a, b, d, i, 18 }
Best Path: { a, b, d, i, 18 }
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
5
9
4

Fig. 4: Random path from the base node to the leaf node.

Again we move one level down in *BestPath* and start a new random move from the node. Therefore in our example we will start from node b, and we found a new random path $\{b, e, k\}$. The new path score (including the distance above b) is 10, which is better than the previous best path score. Therefore we update *BestPath* by *CurrentPath* a, b, e, k and update the score also (See Figure 5).

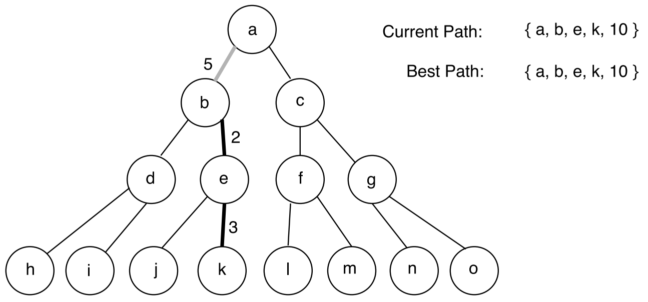
Current Path: { a, b, e, k, 10 }
Best Path: { a, b, e, k, 10 }
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
5
2
3

Fig. 5: A random path from the b node to the leaf node.

Again in the *BestPath* we go one step down and repeat the same process. We play a random move from e and find the new path is $\{e, j\}$(See Figure 6.) The score for *CurrentPath* is 15, which is not better than the previous best path. Hence, we do not update *BestPath*.

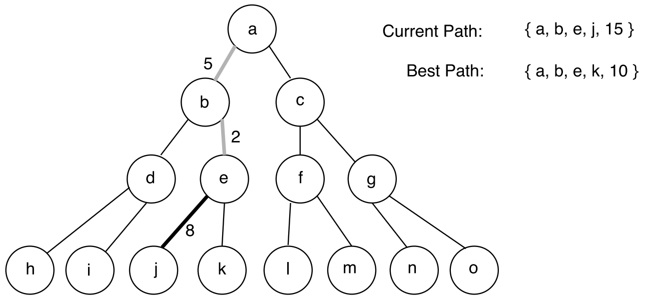
Current Path: { a, b, e, j, 15 }
Best Path: { a, b, e, k, 10 }
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
5
2
8

Fig. 6: Random path from node e to leaf node.

Once we reach the leaf node we repeat the whole process again from the base node. Yet this time *BestPath* would not be empty, as there would be some result from the previous search.

In this kind of problems like in our example, we often face the exploration vs. exploitation dilemma when searching for a new path. In our algorithm by letting investigate a completely new paths (starting randomly from the base node), the algorithm 'cares' about exploration. On the other hand, by investigating *BestPath* on the subsequent levels of the tree, we exploit *BestPath* and hope to improve it.

Finding differential paths in SPECK

In SPECK cipher the only source of non-linearity is modular addition and its complete differential properties (differential distribution tables) are infeasible to calculate, therefore we used our heuristics algorithm to circumvent this limitation and to find the best differential trails. As we described above, the algorithm took a random decision from the search space. For larger variant of SPECK this random property of algorithm is not enough to produce good results and therefore we decided to reduce the search space of algorithm by introducing partial difference distribution table. We used this table in our algorithm and instead of taking random inputs for SPECK we are taking the initial inputs from pDDT table, which contains valid differentials above the threshold value. Each time when SPECK start the next round, the algorithm initially check the values in pDDT table. If it does not find such value in the pDDT set it simply calculate a valid differential output for given inputs, without any threshold condition. In our experiment with SPECK cipher, modular addition for each round is treated as node where we need to take decision of required output (valid differential) and the weight of a valid differential is treated as score. Our aim is to find a differential path for a given number of round with lower weight.

The basic FIND-BEST-PATH function just run the cipher for a given number of round. Function check the differential values in pDDT table having probability greater than some threshold value. In case if algorithm could not find such value in the table then it calculate a valid differential output by XOR-ing the two inputs, which gives highest probability with given inputs (best possible path for given differences). We have not mentioned SPECK encryption operations in the algorithm for simplicity, it is trivial that after each round of encryption st_0 and st_1 changes its value and every time we check these two values in the pDDT table list.

Algorithm 2 Function to find differential path

```
 1: function INT FIND-BEST-PATH($st_0$, $st_1$, $speck\_round$)
 2:     while not end of the round do
 3:         if ($st_0$ and $st_1$) $\in$ pDDT then
 4:             Add differential output and the weight to the path and weight
 5:             list, respectively
 6:         else
 7:             $op = st_1 \oplus st_0$
 8:             $wt = weight(st_0, st_1, op)$ (Calculate the weight using method described
 9:             in section 4)
10:             Add differential output $op$ and the weight $wt$ to the path and weight
11:             list, respectively
12:         end if
13:         SPECK Encryption operations
14:     end while
15: return $path, weight$
16: end function
```

To calculate the differential path by our algorithm using pDDT table we are using main function in Algorithm 3. The calculated weight from round 1 to current round is represented by *weight_above*. The two lists *weight_list* and *best_path_list* saves the weight and list of path for each decision from one round to other. Both list are empty initially and the value of *weight_above* and *best_weight* given to algorithm is 0 and 9999 respectively. Every time the *weight_list* and *best_path_list* is updated with the newly found sequence and the best move is played. The total number of round for which we are trying to find lowest weight is represented by *speck_round*. The first and second half block of SPECK cipher is represented by st_0 and st_1.

Algorithm 3 Finding differential paths in SPECK through Nested Monte-Carlo Search

```
1: function INT NESTED($st_0$, $st_1$, $speck\_round$, $best\_weight$, $weight\_above$)
2:     while $round <= speck\_round$ do
3:         $temp\_path\_list$, $temp\_weight$ = FIND-BEST-PATH($st_0$, $st_1$, $speck\_round$)
4:         if ($temp\_weight + weight\_above < best\_weight$) then
5:             $best\_weight = temp\_weight + weight\_above$
6:             update $best\_path\_list$ by $temp\_path\_list$ (from current round to end of the
7:             $round$)
8:             update $weight\_list$ by $temp\_weight$ (from current round to end of the
9:             $round$)
10:        end if
11:        update $st_0$ and $st_1$ from $best\_path\_list$ with the decision for current $speck\_round$
12:        $weight\_above$ = weight from first round to current speck round
13:        $round = round + 1$
14:    end while
15: return $best\_weight$
16: end function
```

The algorithm can be made anytime with iterative calls:

Algorithm 4 Function for iterative calls

```
1: $best\_weight = 9999$, $weight\_above = 0$, $speck\_round = 9$
2: while $best\_weight > 31$ do
3:     Take the $i^{th}$ indexed value of $st_0$, $st_1$ from pDDT list
4:      path, $best\_weight$ = Nested ($st_0, st_1$, $speck\_round, best\_weight, weight\_above$)
5:     $i = i + 1$
6: end while
7: PRINT $best\_weight$, $path$
```

7 Obtaining a long characteristic from two short ones

It is a well known fact that it is easier to find a short characteristic (for a small number of rounds) instead of a long characteristic. Therefore, we use the start-in-the-middle approach to find a long characteristic from two short ones. In this method, we start our algorithm from the middle of the rounds in two directions, forward toward the end and at the same time backwards towards the beginning. In this experiment we apply internal difference inputs from the middle of the given number of rounds. For example, if we want to find a path for 14 rounds then we pass inputs to our algorithm and let it run for 7 rounds forwards and 7 rounds backwards (reverse). Once we acquire the result for both directions, we combine them together to get a long characteristic for the 14 rounds (see Figure 7). This method also saves time by not needing to search long characteristics and in the end provides us with a better result.

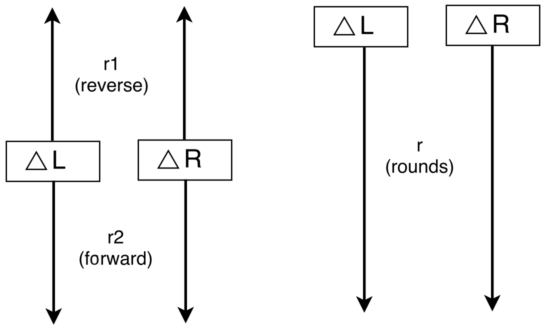

Fig. 7: Algorithm applying on SPECK Cipher

8 Results

In this paper we used our naive algorithm extended with the partial difference distribution table (pDDT) for finding the best differential trails in ARX cipher SPECK. We showed the practical application of the new method on round reduced variants of block cipher from the SPECK family. For the 32-bit state of the cipher, it only make sense to analyse the differential paths with probability higher than 2^{-32}. It is because a path with lower probability would not lead to any meaningful attack, which would be faster than exhaustive search in the 32-bit state. Similarly for SPECK48, SPECK64, SPECK96 and SPECK128 probability should be higher than 2^{-48}, 2^{-64}, 2^{-96} and 2^{-128} respectively. We run the experiment for long characteristics starting from the first round. We report the differential path for up to 8, 9, 11, 10 and 11 rounds of SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128 respectively. In the table left and right part of the state are denoted by Δ_L and Δ_R, respectively. Differences are encoded as hexadecimal numbers (Probability for a given weight is $2^{-weight}$).

Table 2: Differential trails for SPECK32, SPECK48, SPECK64

	SPECK32			SPECK48			SPECK64		
Round	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$
1	0014	0800	2	100082	120000	3	08000000	00000000	1
2	2000	0000	1	901000	001000	3	00080000	00080000	2
3	0040	0040	1	008010	000010	3	00080800	00480800	4
4	8040	8140	2	100090	100010	3	00480008	02084008	6
5	0040	0542	4	801010	001090	5	0a080808	1a4a0848	9
6	8542	904a	6	109080	101400	5	12400040	c0104200	5
7	1540	546a	7	900490	10a490	8	80020200	80801206	5
8	d440	85e9	7	803494	051014	8	80001004	84008030	5
9				919020	b91080	9	80808020	a08481a4	8
10							80040124	84200c01	7
11							a0a00800	81a0680c	9
$\Sigma_r p_r$			-30			-47			-63

Table 3: Differential trails for SPECK96, SPECK128

	SPECK96			SPECK128		
Round	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$
1	000000000080	000000000000	00	0000000000000060	0000000000000000	02
2	800000000000	800000000000	01	2000000000000000	2000000000000000	02
3	808000000000	808000000004	03	2020000000000000	2020000000000001	04
4	800080000004	840080000020	05	2000200000000001	2100200000000008	06
5	808080800020	a08480800124	09	2020202000000008	2821202000000049	10
6	800400008124	842004008801	09	2001000020000049	6108010020000200	10
7	a0a000008880	81a02004c88c	12	2828000020200200	2068080120201203	14
8	01008004c804	0c0180228c60	14	2040200120003201	230060082100a218	18
9	080080a288a8	680c81b6eba8	21	222020282020a22a	3a2320692825b2eb	27
10	c00481364920	80608c811463	18	1001004900059249	c118030041280510	17
11				8808020008280082	80c81a0201682804	15
$\Sigma_r p_r$			-92			-125

In the second part we also perform the experiment starting from the middle of round and run our tool in both direction, reverse as well as forward. Using this method we improved our results and report the differential path for up to 9, 10, 12, 13 and 15 rounds of SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128 respectively. For variant with larger block size say 96 or 128, we got better results compared to previous results.

Table 4: Differential trails for SPECK32, SPECK48, SPECK64

	SPECK32			SPECK48			SPECK64		
Round	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$
1	14ac	5209	7	020888	5a4208	7	02080888	1a4a0848	9
2	0a20	4205	5	d24000	005042	6	92480040	40184200	8
3	0211	0a04	4	008202	020012	4	008a0a00	0481a021	8
4	2800	0010	2	000090	100000	1	00489008	02084018	8
5	0040	0000	0	800000	000000	1	0a080888	1a4a0848	9
6	8000	8000	1	008000	008000	2	92400040	40104200	6
7	8100	8102	2	008080	048080	3	00820200	00001202	4
8	8000	840a	4	848000	a08400	4	00009000	00000010	2
9	850a	9520	6	a00080	a42085	7	00000080	00000000	0
10				248085	0584a8	8	80000000	80000000	1
11							80800000	80800004	3
12							80008004	84008020	5
$\Sigma_r p_r$			-31			-43			-63

Table 5: Differential trails for SPECK96, SPECK128

	SPECK96			SPECK128		
Round	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$
1	a22a20200800	013223206808	14	0096492440040124	0420144304600c01	18
2	019009004800	080110030840	10	2020820a20200800	0120201203206808	14
3	0800800a0808	480800124a08	10	0100009009004800	0801000010030840	10
4	400000924000	004000001042	06	08000000800a0808	4808000000124a08	10
5	000000008202	020000000012	04	4000000000924000	0040000000001042	06
6	000000000090	100000000000	01	0000000000008202	0200000000000012	04
7	800000000000	000000000000	01	0000000000000090	1000000000000000	01
8	800000000000	008000000000	02	8000000000000000	0000000000000000	01
9	008080000000	048080000000	04	0080000000000000	0080000000000000	02
10	048000800000	208400800000	06	0080800000000000	0480800000000000	04
11	208080808000	24a084808001	10	0480008000000000	2084008000000000	06
12	248004000081	018420040088	09	2080808080000000	24a0848080000001	10
13	80a0a0000088	8c81a02004c8	12	2480040000800001	0184200400800008	10
14				00a0a00000808008	0c81a02004808048	14
15				04810080048000c8	608c018020840288	17
$\Sigma_r p_r$			-89			-127

Conclusion

By applying our algorithm based on Nested by reducing the search space using the partial difference distribution table (pDDT) to all five instances of block cipher SPECK with 32, 48, 64, 96 and 128 bit block sizes respectively we obtained result for all variant. Another method we tried is starting from the middle provide more good result for bigger state sizes. By changing threshold we can increase or decrease size of pDDT table. For a bigger threshold value pDDT size is small and speed of experiment is fast because of smaller search space but we can miss few values which are necessary to make good differential path, on the other hand for smaller threshold value pDDT table is large and experiment speed is slow because of bigger search space but it might include the values which are necessary to make the good differential path.

Acknowledgement

Project was financed by Polish National Science Centre, project DEC-2013/09/D/ST6/03918.

References

1. Paweł Morawiecki Ashutosh Dhar Dwivedi and Sebastian Wójtowicz. Finding differential paths in arx ciphers through nested monte-carlo search. *International Journal of electronics and telecommunications*, 64(2):147–150, 2018.
2. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.
3. Alex Biryukov and Vesselin Velichkov. Automatic search for differential trails in ARX ciphers. In Josh Benaloh, editor, *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, volume 8366 of *Lecture Notes in Computer Science*, pages 227–250. Springer, 2014.

4. Alex Biryukov and Vesselin Velichkov. Automatic search for differential trails in arx ciphers. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, pages 227–250, Cham, 2014. Springer International Publishing.
5. Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic search for the best trails in ARX: application to block cipher speck. In Peyrin [9], pages 289–310.
6. Tristan Cazenave. Nested monte-carlo search. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 456–461, 2009.
7. N. Ferguson, B. Schneier S. Lucks, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family. Submission to the NIST SHA-3 Competition (Round 2), 2009.
8. Mitsuru Matsui, editor. *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*. Springer, 2002.
9. Thomas Peyrin, editor. *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*. Springer, 2016.
10. Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In Joseph K. Liu and Ron Steinfeld, editors, *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II*, volume 9723 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2016.