Abstract

In this paper, we propose a new algorithm inspired by Nested to find a 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 use the concept of partial difference distribution table (pDDT) along with the algorithm. The algorithm itself is applied on reduced-round variants of the SPECK block cipher family. In our previous paper, we applied a naive algorithm with a large search space of values and presented the result only for one block size variant of SPECK. In this new approach, we provide the results within a simpler framework and within a very short period of time 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 a large number of rounds, we divide long characteristics into short ones, by easily constructing a large characteristic from two short ones. Instead of starting from the first round, we start from the middle and run the experiments forwards as well as in the reverse direction. Using this method, we were able to improve our previous 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 algorithms which use three simple arithmetic operations: namely modular addition, bitwise rotation and exclusive-OR. In both industry and academia, ARX cipher has gained a lot more interest and attention in the last few years. By using combined linear (XOR, bit shift, bit rotation) and non-linear (modular addition) operations and iterating them for many rounds, ARX algorithms have become more resistance against differential and linear cryptanalysis. ARX lacks a look-up table, associated with S-box based algorithms and therefore has an increased resistance against side-channel attacks. Due to the simplicity of operations, ARX algorithms exhibit excellent performance, especially for software platforms.

In our analysis, we focus on SPECK [1]. SPECK is a secure, flexible and lightweight block cipher designed by researchers from the National Security Agency (NSA) of the United States of America (USA) in June 2013. It has great performance both in software and hardware applications. 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 consists 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 with regards to ARX, for a 32-bit word it is infeasible to calculate these tables. Although a partial difference distribution table (pDDT) containing few fractions of all differentials that has a probability greater than a fixed threshold is still
possible. This becomes 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 becomes a problem since a huge state space exists and there is no clear and obvious way to take the next step. This kind of problem exists in different areas, but our inspiration comes from single-player games such as Morpion solitaire, SameGame and Soduko. The heuristics called Nested Monte-Carlo Search works very well for these games [4]. We can treat a search for good differential paths also as a single-player game and argue that this approach could be a base for more sophisticated heuristics. However, our modified algorithm depends on the technical complexity of this problem, but it is also strongly inspired by Nested Monte-Carlo Search.

In our previous paper [6], we applied a naive approach algorithm to all variants of SPECK and found good results only for one variant with the smallest state size in SPECK32. For bigger variants, our algorithm was demanding to reduce the search space to enhance the random decision process and therefore we used the partial difference distribution table (pDDT) [2] to reduce the search space of our algorithm.

Besides the concept of pDDT our inspiration is drawn from the highways and country roads analogy proposed by Biryukov et al. [2] [3]. 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 roadmap, 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, our algorithm first tries to find a probability above the threshold probability and if such a probability does not exist, then it uses the low probability values. Using this concept, the algorithm does not take a completely random decision in iterations and hence improves the random decision process by using a much smaller search space.

2 Related Cryptanalysis

Biryukov et al.[2] published a paper where they analysed ARX cipher SPECK and by introducing the concept of partial difference distribution table (pDDT) they extend Matsui’s algorithm, originally proposed for DES-like ciphers, to the class of ARX ciphers. They found differential trails of 9, 10 and 13 rounds for 3 variant SPECK32, SPECK48 and SPECK64, respectively.

Biryukov et al. [3] again presented a paper where they propose the adaptation of Matsui’s 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 which 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. [9] 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 Feistel-like structure in which each block is divided into 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 the block size in bits. Each block size is divided into 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, ⊕,
- addition modulo $2^n$, ⊞
- 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:

$$X_{r,L} = ((X_{r-1,L} \gg r_1) \oplus X_{r-1,R}) \oplus k_r$$  \hspace{1cm} (1)

$$X_{r,R} = ((X_{r-1,R} \ll r_2) \oplus X_{r,L})$$  \hspace{1cm} (2)

Fig. 1: The round function of SPECK

Different key sizes have been used by several instances of the SPECK family and the total number of rounds depends on the 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 Table 1.

4 Calculating Differential Probabilities

In [8], Moriai and Lipmaa studied the differential properties of addition. Let $xdp^+(a, b \rightarrow c)$ 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 $(a, b \rightarrow c)$ is valid if and only if:

$$eq(a \ll 1, b \ll 1, c \ll 1) \land (a \oplus b \oplus c \oplus (b \ll 1)) = 0$$  \hspace{1cm} (3)

where

$$eq(p, q, r) := (\neg p \oplus q) \land (\neg p \oplus r)$$  \hspace{1cm} (4)

For every valid differential $(a, b \rightarrow c)$, we define the weight $w(a, b \rightarrow c)$ of the differential as follows:
<table>
<thead>
<tr>
<th>Variant</th>
<th>Block Size(2n)</th>
<th>Word Size(n)</th>
<th>Key Size</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECK32</td>
<td>32</td>
<td>16</td>
<td>64</td>
<td>22</td>
</tr>
<tr>
<td>SPECK48</td>
<td>48</td>
<td>24</td>
<td>72</td>
<td>22</td>
</tr>
<tr>
<td>SPECK64</td>
<td>64</td>
<td>32</td>
<td>96</td>
<td>29</td>
</tr>
<tr>
<td>SPECK96</td>
<td>96</td>
<td>48</td>
<td>144</td>
<td>29</td>
</tr>
<tr>
<td>SPECK128</td>
<td>128</td>
<td>64</td>
<td>256</td>
<td>34</td>
</tr>
</tbody>
</table>

Table 1: SPECK Parameters

\[ w(a, b \rightarrow c) = -\log_2(xdp^+(a, b \rightarrow c)) \]  \hspace{1cm} (5)

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

\[ w(a, b \rightarrow c) := h(-eq(a, b \rightarrow c)), \]  \hspace{1cm} (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. \[2\] is a table that contains all XOR differentials \((a, b \rightarrow c)\) whose differential probabilities (DP) are greater than or equal to pre-defined threshold \( p_{thres} \).

\[(a, b, c) \in pDDT \iff DP(a, b \rightarrow c) \geq p_{thres} \]  \hspace{1cm} (7)

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

\[ p_n \leq \ldots \leq p_k \leq p_{k-1} \leq \ldots \leq p_1 \leq p_0 \]  \hspace{1cm} (8)

where \( p_k = DP(a_k, b_k \rightarrow c_k) \), \( n \geq k \geq 1, p_0 = 1 \) 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 \( k : n \geq k > 0 \) 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 \((a_k, b_k \rightarrow c_k)\) 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: \textbf{Input:} \( n, p_{thres}, k, p_k, a_k, b_k, c_k \).
2: \textbf{Output:} \( pDDT \ D : (a, b, c) \in D : DP(a, b \rightarrow c) \geq p_{thres} \).
3: \textbf{function} \text{computeDDT} (n, p_{thres}, k, p_k, a_k, b_k, c_k) \textbf{end function}

4: \textbf{if} \ n=k \textbf{then}
5: \quad \text{Add } (a, b, c) \leftarrow (a_k, b_k, c_k) \text{ to } D
6: \textbf{end if}
7: \textbf{return} \ D
8: \textbf{for} \ x, y, z \in 0, 1 \textbf{ do}
9: \quad a_{k+1} \leftarrow x|a_k, b_{k+1} \leftarrow y|b_k, c_{k+1} \leftarrow z|c_k
10: \quad p_{k+1} = DP(a_{k+1}, b_{k+1} \rightarrow c_{k+1})
11: \quad \textbf{if} \ p_{k+1} \geq p_{thres} \textbf{ then}
12: \quad \text{computeDDT}(n, p_{thres}, k+1, p_{k+1}, a_{k+1}, b_{k+1}, c_{k+1})
13: \textbf{end if}
14: \textbf{end for}
15: \textbf{end function}

<table>
<thead>
<tr>
<th>Threshold Probability</th>
<th>Elements in pDDT</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>3951388</td>
<td>1.23 min</td>
</tr>
<tr>
<td>0.07</td>
<td>3951388</td>
<td>2.29 min</td>
</tr>
<tr>
<td>0.06</td>
<td>167065948</td>
<td>44.36 min</td>
</tr>
<tr>
<td>0.01</td>
<td>( \geq 72589325174 )</td>
<td>( \geq 29 \text{ days} ).</td>
</tr>
</tbody>
</table>

6 Nested Monte Carlo Search

Our algorithm is inspired by Nested Monte Carlo Search (NMCS) algorithms. The Monte Carlo method is a heuristic based random sampling method. Remi Coulom proposed an application to game-tree search based on Monte Carlo method 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 game, a variant called Nested Monte Carlo Search has been proposed [4].

Let us take a tree-like structure to understand the Nested Monte-Carlo Search algorithm. At each step, the NMCS algorithm tries all possible moves and memorises the move associated with the best score of the lower level searches. In other words, 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.

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 the random game for all possible moves valid for selected state (root). Then it moves one step ahead to the 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.

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

Initially, the algorithm takes a random move from the base node to the leaf node and saves the path in the Current Path list. Let us say that the random path selected by the 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).

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).
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*.

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

In this kind of problem, we often face the exploration versus exploitation dilemma when searching for a new path. In our algorithm, by letting it investigate entirely 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.
7 Formal description of Algorithm

To formally describe the algorithm, let us first define two functions, which are the main building blocks of the algorithm. The first function :math:`\text{RandomPath}(\text{node\_position})` is the function, which for a given node walks a random path in the search tree until it reaches the leaf node. The function :math:`\text{RandomPath}` returns a list of nodes (from the base node to the leaf) and the cost corresponding to the path.

**Algorithm 2** A basic function to generate a random path

1: function `randomPath(node_position)`
2:     while `node_position` ≠ `leaf` do
3:         go randomly to the next node
4:     end while
5:     return path, cost
6: end function

The second function :math:`\text{Nested}(\text{node\_position})` is a recursive function, which calls itself on every level of the tree search until it reaches the leaf node. The pseudo-code of the function is given in Algorithm 2. In the given pseudo-code we use two global variables, which keep a list of nodes in the best path (:math:`\text{best\_path}`) and its corresponding cost (:math:`\text{best\_cost}`). Initially, :math:`\text{best\_path}` is empty and :math:`\text{best\_cost}` is initialized with some big value. (Here we assume that a lower cost means a better solution.)

**Algorithm 3** The recursive function Nested

1: function `Nested(node_position)`
2:     while `node_position` ≠ `leaf` do
3:         \[ `path`, `cost` = \text{RandomPath}(\text{node\_position}) \]
4:         if `(cost < `best\_cost`) then
5:             `best\_cost` = `cost`
6:             `best\_path` = `path`
7:         end if
8:         update `node_position`
9:         by going a level below in `best\_path`
10:     if `node_position` ≠ `leaf` then
11:         Nested `(node\_position)`
12:     end if
13: end while
14: end function

The Nested function can be called iteratively in a loop until we meet our criterion. The criterion could be, for example, a number of iterations, time limit or the maximum cost of the best path. The algorithm could also be easily run in parallel. Either with completely independent instances or with a small overhead to communicate best solutions between instances.
Algorithm 4 Iterative calls to the function Nested

1: best-score = 9999999, node_position = base node
2: while i < number_of_iterations do
3:    Nested (node_position)
4:    i = i + 1
5: end while

8 Finding differential paths in SPECK

In SPECK cipher the only source of non-linearity is the modular addition, and its complete differential properties (differential distribution tables) are infeasible to calculate. Therefore, we have used our heuristics algorithm to circumvent this limitation and to find the best differential trails. As described earlier, the algorithm took a random decision from the search space. For the larger variant of SPECK, this random property of the algorithm is not enough to produce good results. Therefore, we decided to reduce the search space of the algorithm by introducing a partial difference distribution table (pDDT). This table is used in our algorithm and instead of taking random inputs for SPECK, we take the initial inputs from pDDT table, which contains valid differentials above the threshold value. Each time SPECK starts the next round, the algorithm initially checks the values in the pDDT table. If it does not find such a value in the pDDT set it simply calculates 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 a node where we need to take a decision of required output (valid differential), and the weight of a valid differential is treated as a score. Our aim is to find a differential path for a given number of rounds with lower weight.

The basic FIND-BEST-PATH function runs the cipher for a given number of rounds. The function checks the differential values in the pDDT table having a probability greater than some threshold value. In case the algorithm does not find such a value in the table then it calculates a valid differential output by XOR-ing the two inputs, which gives the highest probability with given inputs (best possible path for given differences). We have not mentioned the SPECK encryption operations in the algorithm for simplicity, and 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 5 Function to find differential path

1: function INT FIND-BEST-PATH($st_0$, $st_1$, rounds)
2:     while not end of the rounds do
3:         if $(st_0$ and $st_1) \in$ pDDT then
4:             Add differential output and the weight to the path and weight list, respectively
5:         else
6:             $op = st_1 \oplus st_0$
7:             $wt = weight(st_0, st_1, op)$ (Calculate the weight using method described in section)
8:             Add differential output $op$ and the weight $wt$ to the path and weight list, respectively
9:         end if
10:     SPECK Encryption operations
11:     end while
12: return path, weight
13: end function
To calculate the differential path by our algorithm using the pDDT table, we are using the main function in Algorithm 6. The calculated weight from round 1 to the current round is represented by \( \text{weight}_{\text{above}} \). The two lists \( \text{weight\_list} \) and \( \text{best\_path\_list} \) saves the weight and list of the path for each decision from one round onwards. Both lists are initially empty, and the value of \( \text{weight\_above} \) and \( \text{best\_weight} \) given to algorithm is 0 and 9999 respectively. Every time the \( \text{weight\_list} \) and \( \text{best\_path\_list} \) is updated with the newly found sequence, and the best move is played. The total number of rounds for which we are trying to find the lowest weight is represented by \( \text{rounds} \). The first and second half block of SPECK cipher is represented by \( st_0 \) and \( st_1 \).

Algorithm 6 Finding differential paths in SPECK through Nested Monte-Carlo Search

1: \textbf{function} \( \text{int} \ \text{Nested}(st_0, st_1, \text{rounds}, \text{best\_weight}, \text{weight\_above}) \)
2: \textbf{while} \( \text{round} <= \text{rounds} \) \textbf{do}
3: \hspace{1em} \( \text{temp\_path\_list, temp\_weight} = \text{FIND-BEST-PATH}(st_0, st_1, \text{rounds}) \)
4: \hspace{1em} \textbf{if} (\( \text{temp\_weight} + \text{weight\_above} < \text{best\_weight} \)) \textbf{then}
5: \hspace{2em} \text{best\_weight} = \text{temp\_weight} + \text{weight\_above}
6: \hspace{2em} \text{update \text{best\_path\_list} by \text{temp\_path\_list} (from current round to end of the \text{round})}
7: \hspace{2em} \text{update \text{weight\_list} by \text{temp\_weight} (from current round to end of the \text{round})}
8: \hspace{1em} \textbf{end if}
9: \hspace{1em} \text{update \( st_0 \) and \( st_1 \) from \text{best\_path\_list} with the decision for current \text{rounds}}
10: \hspace{1em} \text{weight\_above} = \text{weight from first round to current speck round}
11: \hspace{1em} \text{round} = \text{round} + 1
12: \textbf{end while}
13: \textbf{return} \text{best\_weight}
14: \textbf{end function}

We can now call NESTED in a loop until a criterion is met (e.g. best weight threshold).

Algorithm 7 Searching a differential path with NESTED

1: \textbf{while} \( \text{best\_weight} > \text{weight\_threshold} \) \textbf{do}
2: \hspace{1em} Take the \( \text{i}^{th} \) indexed value of \( st_0, st_1 \) from pDDT list
3: \hspace{1em} path, \( \text{best\_weight} = \text{Nested}(st_0, st_1, \text{rounds, best\_weight, weight\_above}) \)
4: \hspace{1em} \( i = i + 1 \)
5: \textbf{end while}

9 Obtaining a long characteristic from two short ones

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 shorter ones. In this method, we start our algorithm from the middle of the rounds in two directions, forward and backwards. In this experiment, we apply internal difference inputs from in 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 in the forward direction and 7 rounds in backwards (reverse) direction. Once results from both are achieved, we combine them together to get a long characteristic of 14 rounds. This method also increases time efficiency and provides better results.
10 Differential Attack

Dinur proposed an enumeration technique [5] for key recovery in differential attacks against SPECK. Consider we have differential characteristic of SPECK2n/m with r number of rounds that has probability $p > 2 \cdot 2^{-2n}$. The technique can be used to recover $(r + m)$ rounds. We first attack $(r + m)$ rounds with the value $m = 2$. The number of plaintexts required to recover the key will be $2 \cdot p^{-1}$ with an average time complexity of $2 \cdot p^{-1} \cdot 2^{(m-2)n}$ encryptions. Then we can extend the attack to the remaining instances, with $m = 3$ and $m = 4$.

Let us take one variant SPECK32, we have 8 rounds differential characteristics with probability $2^{-30}$. Combined with Dinur’s enumeration technique for key recovery, given differential characteristics can be used to attack 12-round SPECK32 with $2 \cdot 2^{30} = 2 \cdot 2^{31}$ plaintexts and $2 \cdot 2^{30} \cdot 2^{32} = 2^{63}$ encryptions.

11 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 makes 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 experiments 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 $\Delta_L$ and $\Delta_R$, respectively. Differences are encoded as hexadecimal numbers (Probability for a given weight is $2^{-\text{weight}}$).
### Table 3: Differential trails for SPECK32, SPECK48, SPECK64

<table>
<thead>
<tr>
<th>Round</th>
<th>ΔL</th>
<th>ΔR</th>
<th>weight</th>
<th>ΔL</th>
<th>ΔR</th>
<th>weight</th>
<th>ΔL</th>
<th>ΔR</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0014</td>
<td>0800</td>
<td>2</td>
<td>1000</td>
<td>82</td>
<td>120000 3</td>
<td>0800</td>
<td>0000</td>
<td>000000 1</td>
</tr>
<tr>
<td>2</td>
<td>2000</td>
<td>0000</td>
<td>1</td>
<td>9010</td>
<td>0000</td>
<td>10000 3</td>
<td>0000</td>
<td>0800</td>
<td>000000 2</td>
</tr>
<tr>
<td>3</td>
<td>0040</td>
<td>0040</td>
<td>1</td>
<td>0080</td>
<td>1000</td>
<td>00010 3</td>
<td>0008</td>
<td>0800</td>
<td>0048000 4</td>
</tr>
<tr>
<td>4</td>
<td>8040</td>
<td>8140</td>
<td>2</td>
<td>1000</td>
<td>9000</td>
<td>10000 3</td>
<td>0048</td>
<td>0000</td>
<td>02084008 6</td>
</tr>
<tr>
<td>5</td>
<td>0040</td>
<td>0542</td>
<td>4</td>
<td>8010</td>
<td>0010</td>
<td>0090 5</td>
<td>0a08</td>
<td>0808</td>
<td>1a4a0848 9</td>
</tr>
<tr>
<td>6</td>
<td>8542</td>
<td>904a</td>
<td>6</td>
<td>1090</td>
<td>0010</td>
<td>10400 5</td>
<td>1240</td>
<td>0000</td>
<td>00104200 5</td>
</tr>
<tr>
<td>7</td>
<td>1540</td>
<td>546a</td>
<td>7</td>
<td>9004</td>
<td>9000</td>
<td>10 8  5</td>
<td>8002</td>
<td>0200</td>
<td>8081206 5</td>
</tr>
<tr>
<td>8</td>
<td>d440</td>
<td>85e9</td>
<td>7</td>
<td>8034</td>
<td>9400</td>
<td>051014 8</td>
<td>8000</td>
<td>0000</td>
<td>84008030 5</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>9190</td>
<td>0010</td>
<td>8010 9</td>
<td>8080</td>
<td>0020</td>
<td>80801280 5</td>
</tr>
</tbody>
</table>

### Table 4: Differential trails for SPECK96, SPECK128

<table>
<thead>
<tr>
<th>Round</th>
<th>ΔL</th>
<th>ΔR</th>
<th>weight</th>
<th>ΔL</th>
<th>ΔR</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>0000</td>
<td>00000000000</td>
<td>00000000000</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8000000000000000</td>
<td>0000000000000000</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8080000000000000</td>
<td>0800000000000000</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8000800000000000</td>
<td>8400800000000000</td>
<td>06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8080800000000000</td>
<td>a0848080000000000</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>80400000124</td>
<td>842004008801</td>
<td>09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>a0a000008880</td>
<td>81a00204c88c</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>010080004c804</td>
<td>0c0180228c60</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>008000a288a8</td>
<td>680c81b6eb9a8</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>c00481364920</td>
<td>80608c811463</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>88080200008280082</td>
<td>80c81a0201682804</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the second part, we also perform the experiment starting from the middle round and run our tool in both directions, reverse as well as forward. Using this method, we improved our results and reported the differential path for up to 9, 10, 12, 13 and 15 rounds of SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128 respectively. For variants with larger block size, say 96 or 128, we achieved better results.
### Table 5: Differential trails for SPECK32, SPECK48, SPECK64

<table>
<thead>
<tr>
<th>Round</th>
<th>$\Delta_L$</th>
<th>$\Delta_R$</th>
<th>weight</th>
<th>$\Delta_L$</th>
<th>$\Delta_R$</th>
<th>weight</th>
<th>$\Delta_L$</th>
<th>$\Delta_R$</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14ac 5209 7</td>
<td>020888</td>
<td>5209</td>
<td>020888</td>
<td>5209</td>
<td>020888</td>
<td>5a4a0848</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0a20 4205</td>
<td>5209</td>
<td>4205</td>
<td>0a20 4205</td>
<td>5209</td>
<td>4205</td>
<td>0a20 4205</td>
<td>5209</td>
<td>4205</td>
</tr>
<tr>
<td>3</td>
<td>0211 0a04</td>
<td>008202</td>
<td>008202</td>
<td>020012</td>
<td>4</td>
<td>020012</td>
<td>008a00</td>
<td>0481a021</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>2800 0010</td>
<td>000090</td>
<td>000090</td>
<td>100000</td>
<td>1</td>
<td>100000</td>
<td>0489008</td>
<td>02084018</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>0040 0000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
</tr>
<tr>
<td>6</td>
<td>8000 8000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
<td>008000</td>
<td>0820200</td>
<td>00001202</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>8100 8102</td>
<td>008080</td>
<td>048080</td>
<td>048080</td>
<td>048080</td>
<td>048080</td>
<td>9248000</td>
<td>40104200</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>8000 8404</td>
<td>084000</td>
<td>084000</td>
<td>084000</td>
<td>084000</td>
<td>084000</td>
<td>084000</td>
<td>084000</td>
<td>084000</td>
</tr>
<tr>
<td>9</td>
<td>850a 9520</td>
<td>a00808</td>
<td>a2085</td>
<td>a00808</td>
<td>a2085</td>
<td>a00808</td>
<td>80000000</td>
<td>80000000</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>248085</td>
<td>058408a</td>
<td>8</td>
<td>248085</td>
<td>058408a</td>
<td>8</td>
<td>80000000</td>
<td>80000000</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>80800000</td>
<td>80800000</td>
<td>12</td>
<td>80800000</td>
<td>80800000</td>
<td>12</td>
<td>80800000</td>
<td>80800000</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>80000004</td>
<td>84008020</td>
<td>5</td>
<td>80000004</td>
<td>84008020</td>
<td>5</td>
<td>80000004</td>
<td>84008020</td>
<td>5</td>
</tr>
</tbody>
</table>

**weight** 31 43 63

### Table 6: Differential trails for SPECK96, SPECK128

<table>
<thead>
<tr>
<th>Round</th>
<th>$\Delta_L$</th>
<th>$\Delta_R$</th>
<th>weight</th>
<th>$\Delta_L$</th>
<th>$\Delta_R$</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a22a20200800 013223206808 14</td>
<td>0096492440040124 042014304000c01 18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>019009004800 08011000308040 10</td>
<td>2020820a20200800 0120201203206808 14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0800800a0800 4800800124a08 10</td>
<td>010009009004800 0801000010030840 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>400000924000 00400000104206</td>
<td>08000000800a0808 4800000000124a08 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>000000008202 02000000001204</td>
<td>40000000924000 00400000104206</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>000000000000 00000000000000</td>
<td>0000000000000000 00000000000000012 04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>800000000000 00000000000000</td>
<td>0000000000000000 000000000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>800000000000 00800000000000</td>
<td>0000000000000000 000000000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>800000000000 00000000000000</td>
<td>0000000000000000 000000000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>048000800000 208400880000 06</td>
<td>0808000000000000 048000000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>20800800000 24a0848080101</td>
<td>0480008000000000 208408000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>24804000008 018420040088 09</td>
<td>2080800800000000 24a084808000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>8a0a000088 8c81a02004c8 12</td>
<td>248040000080001 0184200400800008</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>8a0a000088 8c81a02004c8 12</td>
<td>0a0a000000080008 0c81a02004808048</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>04810080048000c8 608c018020840288 17</td>
<td>04810080048000c8 608c018020840288 17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**weight** 89 127

### 12 Conclusion

By applying our algorithm based on Nested and by reducing the search space using the partial difference distribution table (pDDT) to all five instances of block cipher SPECK, we obtained better results for all variants. Another method we attempted was starting from the middle and working in both directions. This method produced good results for bigger state sizes. By changing the threshold, we can increase or decrease the size of pDDT table. For a bigger threshold value, pDDT size is small and speed of experiment is fast because of smaller search space. However, the tradeoff is that we may miss a few values which are necessary to make the good differential path. On the other hand, for
smaller threshold values, pDDT table is large, and the resulting experiment speed is slow because of the bigger search space. That being said, the larger search space might include the values which are necessary to make a good differential path.

Acknowledgement

The project was financed by Polish National Science Centre, project DEC-2014/15/B/ST6/05130.

References

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.