Finding Differential Paths in ARX Ciphers through Nested Monte-Carlo Search

Ashutosh Dhar Dwivedi, Paweł Morawiecki, Sebastian Wójtowicz
Institute of Computer Science, Polish Academy of Sciences, Poland

***Abstract*—We propose the adaptation of Nested Monte-Carlo Search algorithm for finding differential trails in the class of ARX ciphers. The practical application of the algorithm is demonstrated on round-reduced variants of block ciphers from the SPECK family. More specifically, we report the best differential trails,up to 9 rounds, for SPECK32.**

***Keywords* —ARX ciphers, SPECK Cipher, Nested Monte-Carlo Search, Differential Cryptanalysis**

I. Introduction

OVER the last few years the ARX ciphers have gained more attention and interest both in industry and academia. ARX stands for Addition/Rotation/XOR and it refers to a class of algorithms, which use only three basic operations: modular addition, bitwise rotation and eXclusive-OR. There are a few reasons why ARX designs are getting momentum. First, a lack of look-up tables, associated with S-box based designs, increases the resilience against side-channel attacks. Second, ARX algorithms exhibit excellent performance, especially for software platforms. Last but not least, a compact description of such algorithms is particularly appealing for all the environments where memory constraints are really harsh.

In our analysis we focus on a recent design called SPECK [1]. It is a family of block ciphers proposed by researchers from the National Security Agency (NSA) of the USA in June 2013. SPECK aspires to be a flexible, secure, and analysable lightweight block cipher. Its design bears strong similarity to Threefish — the block cipher used in the hash function Skein [2]. SPECK has been designed to provide excellent performance both in software and hardware but have been optimized for performance on micro-controllers. SPECK is a pure ARX cipher with a Feistel-like structure in which both branches are modified at every round. There are five variants of the algorithm with the block size ranges from 32 to 128 bits.

Undoubtedly, ARX designs such as SPECK have many advantages, yet their rigorous cryptanalysis is more difficult. For S-box based algorithms (such as AES), it is relatively easy to evaluate differential properties of the underlying S-box and the whole algorithm. However, in ARX designs the only source of non-linearity is modular addition and its complete differential properties (differential distribution tables) are infeasible to calculate, even for 32-bit word size. Therefore, we need some clever heuristics to circumvent this limitation. Recently, this challenge has been addressed in [3], [4].

In our paper we propose a framework for finding good differential paths in ARX ciphers. Finding good (with relatively high probability) differential paths is a kind of problem, where we face a huge state space and there is no clear and obvious way how we should make a next 'step'. Certainly, we find this kind of a problem in many different areas and we were inspired how it is solved in single-player games such as Sudoku, SameGame, and Morpion solitaire. It turns out that the heuristics called Nested Monte-Carlo Search works very well for these games [5]. It is a randomized heuristic, where a search is organized hierarchically. Our point is that 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.

Description of Speck

SPECK is a family of lightweight block ciphers. The designers specify five variants SPECK32, SPECK48, SPECK64, SPECK96 and SPECK128, where a number in the name denotes a block size in bits. SPECK is the Fiestel-like structure in which each block is divided in two branches and both branches are modified at every round.

Round function: For each round, SPECK encryption uses 3 operations on n-bit words:

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

Let $X_{r-1,L}$ and $X_{r-1,R}$ denote the left and right n-bit input words to the r-th round and let k_r denotes the n-bit round key applied in the r-th round. Then, $X_{r,L}$ and $X_{r,R}$ denotes output words from round r, which are computed as follows:

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

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

Every instance of the SPECK family supports several key sizes and a 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. A summary of parameters (block size, key size, rounds) of all variants are mentioned in the below table.

Differential cryptanalysis in ARX ciphers

Differential cryptanalysis is one of the most powerful technique to analyse the security of symmetric-key cryptographic

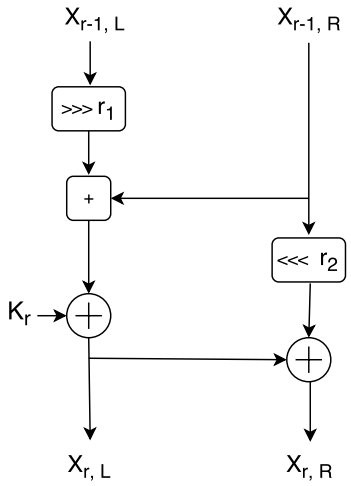

Fig. 1: The round function of SPECK

TABLE I: Speck parameters

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

primitives. ARX cipher designers have previously attempted to argue the security against differential cryptanalysis by using various techniques to search for differential characteristics of high-probability and explaining that such characteristics could not be found for sufficient number of rounds .

In case of AES-like algorithms, where we typically use 8- or 4-bit S-box, differential properties can be easily computed by creating the difference distribution table (DDT) for the S-box. But ARX algorithms use modular addition as a source of non-linearity rather than S-boxes, and typical word size in ARX ciphers is 32- or 64-bit. Constructing the DDT for n-bit words modular addition would require $2^{3n} \times 4$ bytes of memory and would clearly be infeasible for a word size of 32-bit. Therefore, we need some clever heuristics to circumvent this limitation.

Calculating Differential Probabilities: In [6], Moriai and Lipmaa studied the differential properties of addition. Let $xdp^+(\alpha, \beta \rightarrow \gamma)$ be the XOR-differential probability of addition modulo 2^n, with input differences α and β and output difference γ. Moriai and Lipmaa proved that the differential $(\alpha, \beta \rightarrow \gamma)$ is valid if and only if:

$$eq(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \wedge (\alpha \oplus \beta \oplus \gamma \oplus (\beta \ll 1)) = 0 \quad (3)$$

where

$$eq(x, y, z) := (\neg x \oplus y) \wedge (\neg x \oplus z) \quad (4)$$

For every valid differential $(\alpha, \beta \rightarrow \gamma)$, we define the weight $w(\alpha, \beta \rightarrow \gamma)$ of the differential as follows:

$$w(\alpha, \beta \rightarrow \gamma) = -\log_2(xdp^+(\alpha, \beta \rightarrow \gamma)) \quad (5)$$

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

$$w(\alpha, \beta \rightarrow \gamma) := h * (\neg eq(\alpha, \beta \rightarrow \gamma)), \quad (6)$$

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 bit rotation are linear in GF(2), therefore for these two operations for every input difference there is only one valid output difference.

In the paper we always refer to the XOR differences but our analysis could be easily extended to differences defined as the addition operation.

NESTED MONTE CARLO SEARCH

The Monte Carlo method — the heuristic based on random sampling — dates back to the 1940s. In 2008, Remi Coulom proposed what is now known as Monte Carlo Tree Search (MCTS), that is the application of the Monte Carlo method to game-tree search. The algorithm is particularly useful for games where it is hard to formulate an evaluation function, such as the game of Go. A very recent success of AlphaGo is partly due to the efficient MCTS algorithm (combined with a deep neural network) [7]. For single-player games, a variant called the Nested Monte-Carlo Search has been proposed [5].

Before we give a more formal description of Nested Monte-Carlo Search, let us explain this algorithm with a simple example. Our task is to find (possibly) shortest way from one city to another. We represent all possible paths as a tree, where a root is our starting point and leaves are ending points reached by different paths. Each edge between intermediate nodes is associated with a number, which is simply a distance between two nodes. Two lists $CurrentPath$ and $BestPath$ represent the random path currently under investigation and the best available path from previous searches, respectively. The last element in both list shows a total distance travelled. Initially both the lists are empty.

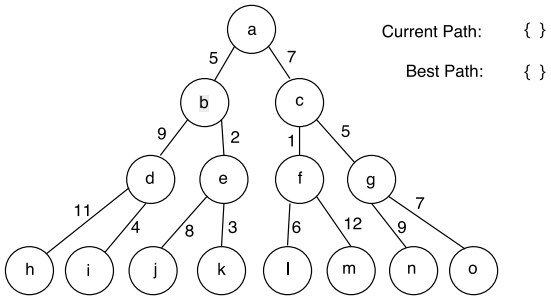

Fig. 2: Different paths from the root (base node) to the destination (leaf nodes)

Nested Monte-Carlo Search uses random playouts. Let us take random moves from the base node to the leaf node and save the path in Current Path list. Our random path is $\{a, b, d, i\}$ which has a distance score 18. Since there has not been a better path ($BestPath$ is empty), then we save the current path and its distance as $BestPath$, as shown in Figure 3.

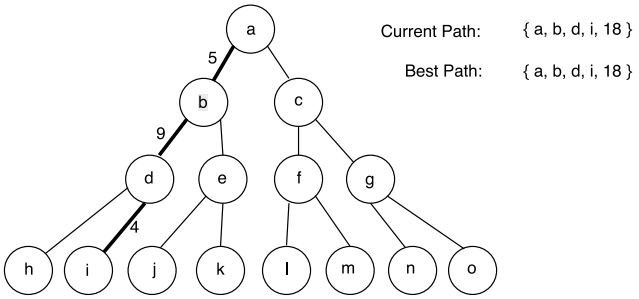

Fig. 3: Random path from the base node to the leaf node.

Next, we go one level down in $BestPath$ and start a random walk from the new node. In our example, starting from the node b, we randomly find a new path $\{b, e, k\}$. The score for the new path (including the distance above b) is 10, which is better than the previous best path score. So we update $BestPath$ by $CurrentPath$ $\{a, b, e, k\}$ and update the score also. (See Figure 4.)

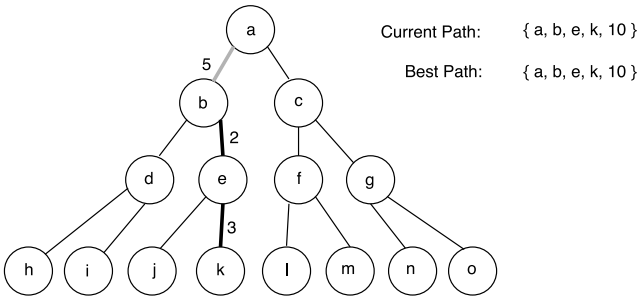

Fig. 4: A random path from the b node to the leaf node.

Then, we again go one step down in $BestPath$ and repeat the process. This time we play a random move from e and find that new path is $\{e, j\}$. (See Figure 5.) The score for $CurrentPath$ is 15, which is not better than the previous best path score. Thus we do not update $BestPath$.

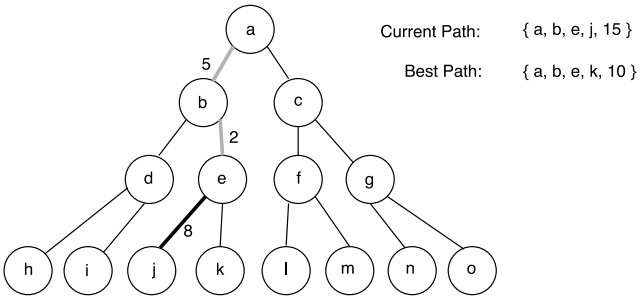

Fig. 5: Random path from node e to leaf node.

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 Nested Monte Carlo Search 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.

Formal description of Nested Monte Carlo Search

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

Algorithm 1 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 $Nested(node\_position)$ 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 ($best\_path$) and its corresponding cost ($best\_cost$). Initially, $best\_path$ is empty and $best\_cost$ is initialized with some big value. (Here we assume that a lower cost means better solution.)

Algorithm 2 The recursive function Nested

```
function NESTED(node_position)
    while node_position ≠ leaf do

        path, cost = RandomPath(node_position)
        if (cost < best_cost) then
            best_cost = cost
            best_path = path
        end if

        update node_position
        by going a level below in best_path

        if node_position ≠ leaf then
            Nested (node_position)
        end if
    end while

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 Nested Monte Carlo Search could be also easily run in parallel. Either with completely independent instances or with a small overhead to communicate best solutions between instances.

Algorithm 3 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
```

Finding Differential Paths through Nested Monte-Carlo Search

Finding differential paths in cryptographic algorithms could be seen as a single-player game. We start from some input difference and at each round of a cipher a decision is to be taken. The decision here means what input-output transition to choose through the non-linear part of the round. Specifically, in ARX primitives, it is a transition through the modular addition — a source of non-linearity in these algorithms. Typically, for given input differences there are many possible output differences. (Exact formulas which transitions are valid were given in the subsection 'Calculating Differential Probabilities'.) Each transition through the modular addition is probabilistic. Transition with a very low probability has a very high cost and vice-versa. The total cost of a path is calculated by multiplying probabilities associated with all transitions through modular addition. The aim of the game is to find a differential path for a given number of rounds with possibly highest probability.

Results for SPECK32: We adapted the Nested Monte-Carlo Search algorithm for finding differential trails in SPECK. We particularly focus on the variant with 32-bit state, namely SPECK32. 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. The best path we found for SPECK32 covers 9 rounds (out of 22) with probability 2^{-31}. It matches the state-of-the-art results reported in [3], yet within simpler framework. In Table II we show a full 9-round differential path. Left and right part of the state are denoted by Δ_L and Δ_R, respectively. Differences are encoded as hexadecimal numbers. The last column shows weights for each round. (Probability for a given weight is 2^{weight}.)

TABLE II: 9-round differential trails for SPECK32

Round	Δ_L	Δ_R	$\log_2 p$
0	A60	4205	-0
1	211	A04	-5
2	2800	10	-4
3	40	0	-2
4	8000	8000	-0
5	8100	8102	-1
6	8000	840A	-2
7	850A	9520	-4
8	802A	D4A8	-6
9	A8	520B	-7
$\Sigma_r p_r$			-31

Conclusion and Future Work

By applying Nested Monte-Carlo Search on SPECK32 we found the same result as the one obtained by Biryukov and Velichkov in [3]. This makes our approach promising as it has potential to provide state-of-the-art results but within a simpler framework. To apply the method to ciphers with a bigger state, we need to enhance a random decision process and reduce the search space by some other heuristics. This constitutes our next research goal.

Acknowledgement

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

References

[1] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," *IACR Cryptology ePrint Archive*, vol. 2013, p. 404, 2013.

[2] N. Ferguson, B. S. 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.

[3] A. Biryukov and V. Velichkov, "Automatic search for differential trails in ARX ciphers," in *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, 2014, pp. 227–250.

[4] A. Biryukov, V. Velichkov, and Y. L. Corre, "Automatic search for the best trails in ARX: application to block cipher speck," in *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, 2016, pp. 289–310.

[5] T. 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*, 2009, pp. 456–461.

[6] M. Matsui, Ed., *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2355. Springer, 2002. [Online]. Available: https://doi.org/10.1007/3-540-45473-X

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.