IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 12, 2018, accepted January 11, 2019, date of publication January 23, 2019, date of current version February 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2894337

Differential Cryptanalysis of Round-Reduced SPECK Suitable for Internet of Things Devices

ASHUTOSH DHAR DWIVEDI[1,2], PAWEL MORAWIECKI[1], AND GAUTAM SRIVASTAVA[2,3]

[1]Institute of Computer Science, Polish Academy of Sciences, 01-248 Warsaw, Poland
[2]Department of Mathematics and Computer Science, Brandon University, Brandon, MB R7A 6A9, Canada
[3]Research Center for Interneural Computing, China Medical University, Taichung 40402, Taiwan

Corresponding author: Ashutosh Dhar Dwivedi (ashudhar7@gmail.com)

The work of A. D. Dwivedi and P. Morawiecki was supported by the Polish National Science Centre under Grant DEC-2014/15/B/ST6/05130.

ABSTRACT In this paper, we focus on differential cryptanalysis of a lightweight ARX cipher. These ciphers use three simple arithmetic operations, namely, modular addition, bitwise rotation, and exclusive-OR, and therefore, are designed very well to perform over the Internet-of-Things (IoT) devices. We choose a very well-known ARX cipher designed by the National Security Agency (NSA) of the United States of America in June 2013, named SPECK. SPECK was subjected to several years of detailed cryptanalytic analysis within NSA and has been subjected to academic analysis by researchers worldwide. SPECK is specially optimized for low-cost processors like those used in the IoT devices. We first find the differential paths for all the variants of SPECK, and based on that differential path, we attack the round-reduced variant of the cipher. Finding differential paths in ARX is one of the most difficult and time-consuming problems due to the huge state space. We use a nested-based heuristic technique to find a differential path which is inspired by the nested Monte Carlo search (NMCS) algorithm. NMCS was successfully applied before for different games: Morpion Solitaire, SameGame, and 16 × 16 Sudoku, but the use of such heuristic techniques in cryptography is entirely new and time-saving.

INDEX TERMS Differential path, ARX ciphers, nested Monte-Carlo search, IoT ciphers, differential cryptanalysis, SPECK.

I. 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 used for IoT devices. After mobile internet technologies and the World Wide Web, the time has come for Internet of Things (IoT). IoT consists of devices responsible for generating, processing and exchanging privacy-sensitive information. It has a broad range of applications including health management, smart homes, traffic, agriculture, weather monitoring just to name a few. IoT devices are lightweight and also have shallow energy footprints. This small amount of available energy is generally used to execute core application functionality and therefore supporting other challenges of security and privacy is quite challenging. Due to lightweight encryption methods, ARX ciphers are well suited for IoT devices.

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 is known for great performance both in software and hardware applications. Its design is similar to Threefish - the block cipher used in the hash function Skein [2]. 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 consists 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 clearly infeasible to calculate these tables. However, a partial difference distribution table (pDDT) containing just a few fractions of all differentials that has a probability greater than some fixed threshold is still a viable option. This becomes possible due to the fact that the probabilities of XOR (respectively ADD) differentials through the modular addition (respectively 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 Soduku. The heuristics called Nested Monte-Carlo Search works very well for these games as shown in [3]. 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 [4] and [5], 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) [6] to reduce the search space of our algorithm. In another work [5], we applied this advance method to ARX cipher LEA and found differential path for 13 rounds.

Besides the concept of pDDT our inspiration is drawn from the highways and country roads analogy proposed by Biryukov and Velichkov [6] and Biryukov *et al.* [8]. 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 (with respect to 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.

II. RELATED WORK

Biryukov and Velichkov [6] published a paper where they analyzed 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.* [7] 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.* [8] 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.

In the context of security of IoT devices data, we have seen some strong work recently by Wu *et al.* [10]–[13]. They have been able to focus on security of IoT and Big Data. Both of which are important factors going forward with Smart City design and implementation.

The SIMON and SPECK families of block ciphers were designed specifically to offer security on constrained devices, where simplicity of design is crucial [13]. The NSA developed the SPECK as an aid for securing applications in very constrained environments where AES may not be suitable, such as IoT [14]. Specifically, in [15], results brought some new insights into the question of how well lightweight ciphers like SPECK are suited to secure the Internet of things.

III. 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, the left half and right half.

ROUND FUNCTION

SPECK uses 3 basic operations on n-bit words 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:

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

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

TABLE 1. SPECK parameters.

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

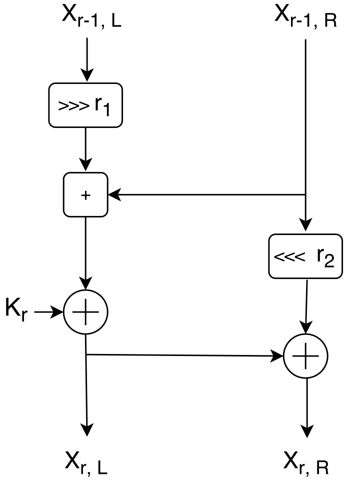

FIGURE 1. The round function of SPECK.

We can clearly visualize the round function of SPECK in Figure 1. 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.

IV. CALCULATING DIFFERENTIAL PROBABILITIES

Lipmaa and Moriai [17] 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 output difference c. Lipmaa and Moriai [17] 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 bit rotation are linear in GF(2), therefore for these two operations for every input difference there is only one valid output difference.

V. PARTIAL DIFFERENCE DISTRIBUTION TABLES (PDDT)

Partial difference distribution table (pDDT) proposed by Biryukov and Velichkov [6] is a table that contains all XOR differentials $(a, b \to c)$ whose differential probabilities (DP) are greater than or equal to a 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 the following proposition: The differential probability (DP) of XOR of addition modulo 2^n is monotonously decreasing with the word size of differences a, b, c.

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

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 $x_k = x[k-1 : 0]$. In our algorithm, we start from least-significant (LS) bit position $k = 0$ and recursively assign the values to $a[k], b[k]$ and $c[k]$. For each bit position $k : n > k > 0$ check if probability of partially constructed $(k+1) - bit$ differential is greater than the threshold. If yes, then move to 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. Initial value of k is 0 and $a_0, b_0, c_0 = \phi$.

In our nested algorithm. shown in Algorithm 1, we set the threshold value equal to 0.1 and therefore the size of our

Algorithm 1 Computation of a pDDT for XOR

```
Input: $n, p_{thres}, k, p_k, a_k, b_k, c_k$.
Output: pDDT $D$ : $(a, b, c) \in$ D : DP$(a, b \rightarrow c) \geq p_{thres}$.
function computepddt($n, p_{thres}, k, p_k, a_k, b_k, c_k$)
    if n==k then
        Add $(a, b, c) \longleftarrow (a_k, b_k, c_k)$ to D
    end if
    return
    for $x, y, z \in 0, 1$ do
        $a_{k+1} \longleftarrow x|a_k, b_{k+1} \longleftarrow y|b_k, c_{k+1} \longleftarrow z|c_k$
        $p_{k+1} = DP(a_{k+1}, b_{k+1} \rightarrow c_{k+1})$
        if $p_{k+1} \geq p_{thres}$ then
            computepddt($n, p_{thres}, k+1, p_{k+1}, a_{k+1},$
            $b_{k+1}, c_{k+1}$)
        end if
    end for
    return
end function
```

TABLE 2. Timings of pDDT for XOR on 32-bit words using algorithm 1.

Threshold Probability	Elements in pDDT	Time
0.1	3951388	1.23 min
0.07	3951388	2.29 min
0.06	167065948	44.36 min
0.01	$\geq$ 72589325174	$\geq$ 29 days.

algorithms search space is equal to 3951388 from Table 2. If we decrease the value of threshold the size of search space will increase depending on the threshold value and the differential path computational speed of our algorithm will decrease in equal proportion.

VI. 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. Coulom [19] proposed an application to game-tree search based on Monte Carlo method in 2007 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 in [3].

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 memorizes the move associated with the best score of the lower level searches. In other words, a nested move of level 1 makes a playout for every possible move and chooses to play the move of the best playout. A nested move of level 2 does the same thing except that it replaces the playout by a nested move of level one.

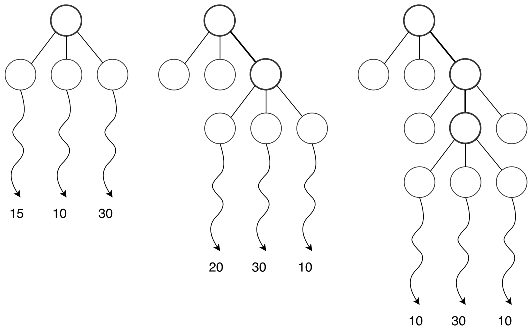

FIGURE 2. Nested Monte Carlo search.

During the first iteration the initial state (root) is selected, and for the selected state all legal moves are determined (Figure 2). Therefore at level 0 it plays the random game for all possible moves valid for selected state (root). Then it moves one step ahead to the next level with the 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 in this context. Our goal is to find the shortest path from one city to another city. We represent all possible paths as a tree, as shown in Figure 3. The root of the tree is considered as the starting point, and all leaves are end points reached by different paths (nodes). Each edge between nodes is associated with a number which represents the distance between the two nodes. Initially, we have two lists named *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 traveled. Both the lists are initially empty.

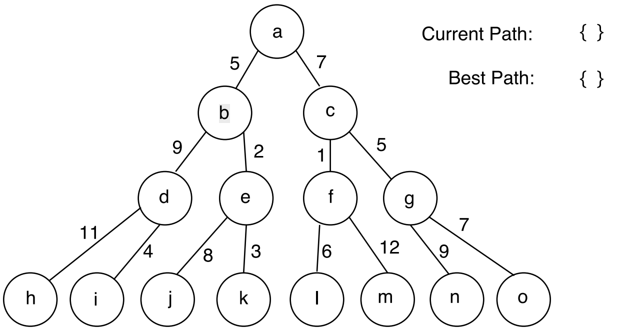

FIGURE 3. Different paths from the root (base node) to the destination (leaf nodes).

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

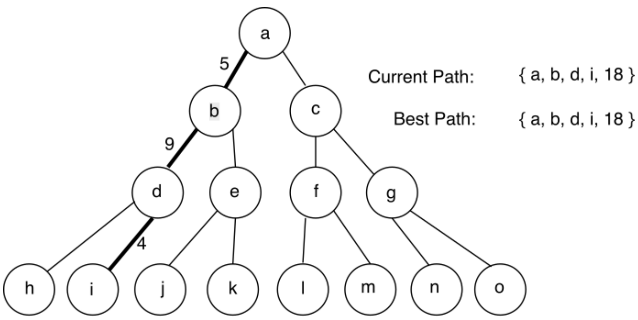
a
5
b
c
9
d
e
f
g
4
h
i
j
k
l
m
n
o
Current Path: { a, b, d, i, 18 }
Best Path: { a, b, d, i, 18 }

FIGURE 4. Random path from the base node to the leaf node.

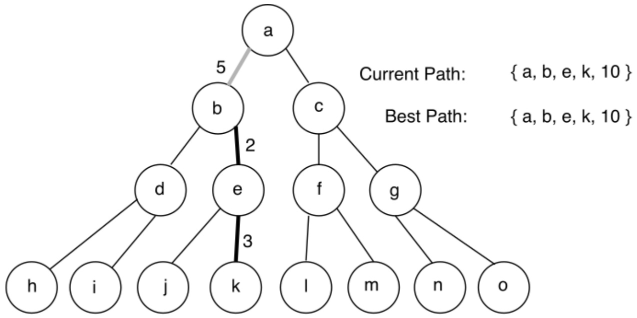
a
5
b
c
2
d
e
f
g
3
h
i
j
k
l
m
n
o
Current Path: { a, b, e, k, 10 }
Best Path: { a, b, e, k, 10 }

FIGURE 5. A random path from the *b* node to the leaf node.

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.

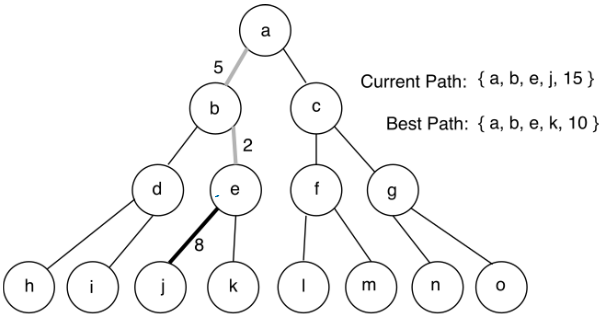
a
5
b
c
2
d
e
f
g
8
h
i
j
k
l
m
n
o
Current Path: { a, b, e, j, 15 }
Best Path: { a, b, e, k, 10 }

FIGURE 6. Random path from node *e* to leaf node.

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.

VII. FORMAL DESCRIPTION OF OUR ALGORITHM BASED ON NMCS

To formally describe the algorithm, let us first define two functions, which are the 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 2 A Basic Function to Generate a Random Path

```
function randomPath(node_position)
    while node_position $\neq$ leaf do
        go randomly to the next node
    end while
return path, cost
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 a better solution.)

Algorithm 3 The Recursive Function Nested

```
function Nested($node\_position$)
    while $node\_position \neq$ 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 $node\_position \neq$ leaf then
            Nested ($node\_position$)
        end if
    end while

end function
```

Algorithm 4 Iterative Calls to the Function Nested

```
best-score = 9999999, node_position = base node
while $i < number\_of\_iterations$ do
    Nested ($node\_position$)
    $i = i + 1$
end while
```

Algorithm 5 Function to Find Differential Path

```
function int FIND-BEST-PATH($st_0$, $st_1$, $rounds$)
    while not end of the rounds do
        if ($st_0$ and $st_1$) $\in$ pDDT then
            Add differential output and the weight to the
            path and weight list, respectively
        else
            $op = st_1 \oplus st_0$
            $wt = weight(st_0, st_1, op)$ (Calculate the
            weight using method described in section IV)
            Add differential output $op$ and the weight $wt$
            to the path and weight list, respectively
        end if
        SPECK Encryption operations
    end while
return $path$, $weight$
end function
```

The Nested function can be called iteratively in a loop until we meet our criterion as shown in Algorithm 3. 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. We can do this either with completely independent instances or with a small overhead to communicate best solutions between instances.

VIII. FINDING DIFFERENTIAL PATHS

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 use our heuristics algorithm to circumvent this limitation and to find the best differential trails. As described earlier, the algorithm takes 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 decide 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. We show the details in Algorithm VIII. 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

Algorithm 6 Finding Differential Paths in SPECK Through Nested Monte-Carlo Search

```
function int Nested($st_0$, $st_1$, $rounds$, $best\_weight$,
$weight\_above$)
    while not end of the rounds do
        $temp\_path\_list$, $temp\_weight$ = FIND-BEST-
        PATH($st_0$,$st_1$, $rounds$)
        if ($temp\_weight + weight\_above < best\_weight$)
then
            $best\_weight = temp\_weight + weight\_above$
            Update $best\_path\_list$ by $temp\_path\_list$
(from
            current round to end of the $round$)
            Update $weight\_list$ by $temp\_weight$ (from
            current round to end of the $round$)
        end if
        update $st_0$ and $st_1$ from $best\_path\_list$ with the
        decision for current $rounds$
        $weight\_above$ = weight from first round to cur-
rent
        speck round
        $rounds = rounds + 1$
    end while
return $best\_weight$
end function
```

Algorithm 7 Searching a Differential Path With NESTED

```
while $best\_weight > weight\_threshold$ do
    Take the $i^{th}$ indexed value of $st_0$, $st_1$ from pDDT list
     path, $best\_weight$ = Nested ($st_0$, $st_1$,)
    $rounds$, $best\_weight$, $weight\_above$
    $i = i + 1$
end while
```

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 different 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.

To calculate the differential path by our algorithm using the pDDT table, we use the main function in Algorithm 6. The calculated weight from round 1 to the current round is represented by *weight_above*. The two lists *weight_list*

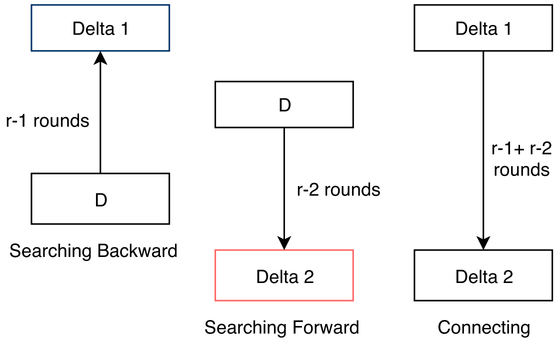

FIGURE 7. Algorithm applying on SPECK Cipher.

and *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 *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 rounds for which we are trying to find the lowest weight is represented by *srounds*. The first and second half block of SPECK cipher is represented by st_0 and st_1.

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

IX. OBTAINING LONG CHARACTERISTIC

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 to get a long characteristic of 14 rounds. This method also increases time efficiency and provides better results.

X. RESULTS

In this paper, we use our naive algorithm extended with the partial difference distribution table (pDDT) for finding the best differential trails in ARX cipher SPECK. We show 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 analyze 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 in Appendix I (Table 3,4) 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 $2^{-weight}$).

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 report the differential path in Appendix I (Table 6,7) 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.

XI. DIFFERENTIAL ATTACKS

Dinur [20] proposed an enumeration technique for key recovery in differential attacks against SPECK. Consider we have differential characteristic of SPECK2n/mn 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.

XII. 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 obtain 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 a 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.

APPENDIX I.

APPENDIX II.
DINUR's ENUMERATION TECHNIQUE FOR KEY RECOVERY ATTACK AGAINST SPECK

Generally for the key recovery of differential cryptanalysis, counting techniques are common. We extract partial

TABLE 3. Differential trails for SPECK32, SPECK48, SPECK64.

	SPECK32			SPECK48			SPECK64		
Round	Δ_L	Δ_R	$weight$	Δ_L	Δ_R	$weight$	Δ_L	Δ_R	$weight$
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 4. Differential trails for SPECK96, SPECK128.

	SPECK96			SPECK128		
Round	Δ_L	Δ_R	$weight$	Δ_L	Δ_R	$weight$
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

TABLE 5. Differential attack on SPECK.

Variants $2n/mn$	Rounds attacked/Total Rounds	Time	Data	Memory	Differential trail
$32/64$	$12/22$	2^{63}	2^{31}	2^{22}	8
$48/72$	$12/22$	2^{72}	2^{48}	2^{22}	9
$48/96$	$13/23$	2^{96}	2^{48}	2^{22}	9
$64/96$	$14/26$	2^{96}	2^{64}	2^{22}	11
$64/128$	$15/27$	2^{128}	2^{64}	2^{22}	11
$96/96$	$12/28$	2^{93}	2^{93}	2^{22}	10
$96/144$	$13/29$	2^{141}	2^{93}	2^{22}	10
$128/128$	$13/32$	2^{126}	2^{126}	2^{22}	11
$128/192$	$14/33$	2^{190}	2^{126}	2^{22}	11
$128/256$	$15/34$	2^{254}	2^{126}	2^{22}	11

key material from outer rounds of the cipher using statistical analysis. However in case of SPECK, Dinur [20] increased the number of rounds attacked with the application of enumaration techniques in the key recovery. The enumeration technique tries all suggestions for the full key proposed by a sub-cipher attack. To describe the attack on SPECK with enumeration technique, we consider the case when $m = 2$, master key contains 2 words. This attack can be extended to other cases when $m = 3$ or $m = 4$. Let us say, we have r-round differential $(\Delta x_0, \Delta y_0) \to (\Delta x_r, \Delta y_r)$ of the cipher with probability p. In such case we can proceed to attack as follow:

TABLE 6. Differential trails for SPECK32, SPECK48, SPECK64.

	SPECK32			SPECK48			SPECK64		
Round	Δ_L	Δ_R	$weight$	Δ_L	Δ_R	$weight$	Δ_L	Δ_R	$weight$
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 7. Differential trails for SPECK96, SPECK128.

	SPECK96			SPECK128		
Round	Δ_L	Δ_R	$weight$	Δ_L	Δ_R	$weight$
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

- Request the encryption of p^{-1} plaintext pairs P and $P' = P \oplus (\Delta x_0, \Delta y_0)$ and denote the corresponding ciphertexts by C and C', respectively.
- For the plaintext pairs P and P': Execute the 2 round attack (Section "The 2-Round Attack") using $(\Delta x_r, \Delta y_r)$, C and C' and get suggestions for k_{r+1} and k_r. For the suggested value of k_{r+1} and k_r, reverse the key schedule to obtain the master key. Test the master key using additional encryptions and if passes the test return the master key.

In the given attack, we need $2 \cdot p^{-1}$ plaintexts. The average time complexity is less than 2 encryptions in the key recovery of 2 round attack (Section "The 2-Round Attack") and therefore the total time complexity of the attack is $2 \cdot p^{-1}$. For $m = 3$ or $m = 4$, by guessing the last $m - 2$ round keys, we can recover $r + m$ rounds with complexity $2 \cdot p^{-1} \cdot 2^{(m-2)n}$ encryptions.

THE 2-ROUND ATTACK

In this section we present the details of 2 round attack on cipher. We use r-round differential path for this attack. Consider we have initial difference $(\Delta x_0, \Delta y_0)$ and final difference $(\Delta x_r, \Delta y_r)$. We take final differences $(\Delta x_r, \Delta y_r)$ as the input of 2-round differential attack. We are given actual values of (x_{r+2}, y_{r+2}) and $(x_{r+2} \oplus \Delta x_{r+2}, y_{r+2} \oplus \Delta y_{r+2})$. We try to enumerate possible round keys k_r and k_{r+1} so that partial decryption of 2 round of the pairs (x_{r+2}, y_{r+2}) and $(x_{r+2} \oplus \Delta x_{r+2}, y_{r+2} \oplus \Delta y_{r+2})$ is equal to $(\Delta x_r, \Delta y_r)$.

The notation we use in our analysis is given in Figure 8, where the XOR differential notation is given on the left, and the notation of the intermediate encryption values for (x_{r+2}, y_{r+2}) is given on the right.

All the XOR differences in the 2 round scheme can be easily determined. Since $\Delta x_{r+1} = \Delta y_{r+1} \oplus (\Delta y_r \lll \beta)$ and $\Delta y_{r+1} = (\Delta x_{r+2} \oplus \Delta y_{r+2}) \ggg \beta$ can be calculated using

TABLE 8. Differential Attack on SPECK.

Variants $2n/mn$	Rounds attacked/Total Rounds	Time	Data	Memory	Differential trail
$32/64$	$13/22$	2^{64}	2^{32}	2^{22}	9
$48/72$	$13/22$	2^{68}	2^{44}	2^{22}	10
$48/96$	$14/23$	2^{92}	2^{44}	2^{22}	10
$64/96$	$15/26$	2^{96}	2^{64}	2^{22}	12
$64/128$	$16/27$	2^{128}	2^{64}	2^{22}	12
$96/96$	$15/28$	2^{90}	2^{90}	2^{22}	13
$96/144$	$16/29$	2^{138}	2^{90}	2^{22}	13
$128/128$	$17/32$	2^{128}	2^{128}	2^{22}	15
$128/192$	$18/33$	2^{192}	2^{128}	2^{22}	15
$128/256$	$19/34$	2^{256}	2^{128}	2^{22}	15

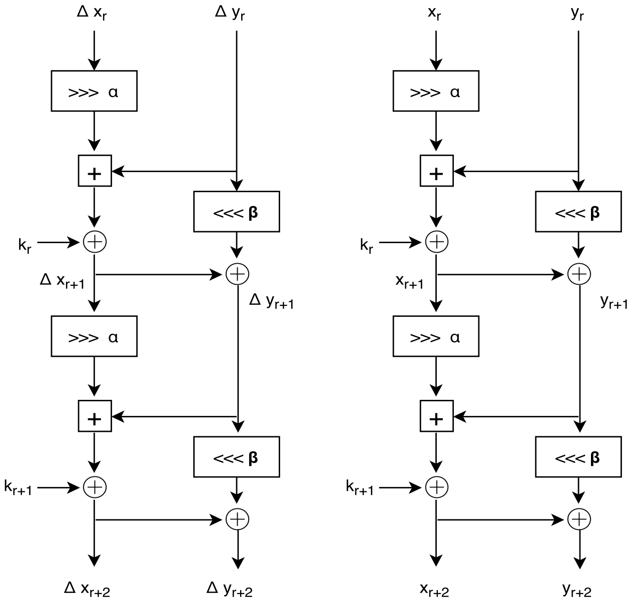

FIGURE 8. Two rounds of Speck.

known variables. The value of $y_{r+1} = (x_{r+2} \oplus y_{r+2}) \ggg \beta$ can be calculated using the known values of (x_{r+2}, y_{r+2}) whereas x_{r+1} and (x_r, y_r) remains unknown. Finding the values of k_{r+1} and k_r is equivalent to find the values of x_r and x_{r+1}, as $k_{r+1} = (y_{r+1} \boxplus (x_{r+1} \ggg \alpha)) \oplus x_{r+2}$ and as $y_r = (x_{r+1} \oplus y_{r+1}) \ggg \beta$, then $k_r = (y_r \boxplus (x_r \ggg \alpha)) \oplus x_{r+1}$ can be derived as well. Therefore we concentrate on finding the intermediate values of x_r and x_{r+1}.

The problem of solving differential equations of addition (DEA) of the form $(x \oplus \delta_1) \boxplus (y \oplus \delta_2) = (x \boxplus y) \oplus \delta_3$ (where $\delta_1, \delta_2, \delta_3$ are given and x, y are unknown variables) is a basic problem in the analysis of ARX cryptosystems, and was extensively studied in several papers. For SPECK we can also write the same equation and find values of x_r and x_{r+1}, we omit the right circular shift $\ggg$, and then we have two differential equations of addition:

$$(x_r \oplus \Delta x_r) \boxplus (y_r \oplus \Delta y_r) = (x_r \boxplus y_r) \oplus \Delta x_{r+1}$$
$$(x_{r+1} \oplus \Delta x_{r+1}) \boxplus (y_{r+1} \oplus \Delta y_{r+1}) = (x_{r+1} \boxplus y_{r+1}) \oplus \Delta x_{r+2}$$

where all differences are known, and in the second equation y_{r+1} and $y_{r+1} \oplus \Delta y_{r+1}$.

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 Cryptol. ePrint Arch., Santa Barbara, CA, USA, Tech. Rep. 404, 2013.

[2] N. Ferguson *et al.*, "The skein hash function family," in *Proc. NIST SHA-3 Competition (Round 2)*, 2009, p. 3.

[3] T. Cazenave, "Nested Monte-Carlo search," in *Proc. 21st Int. Joint Conf. Artif. Intell. (IJCAI)*, Pasadena, CA, USA, Jul. 2009, pp. 456–461.

[4] A. D. Dwivedi, P. Morawiecki, and S. Wójtowicz, "Finding differential paths in ARX ciphers through nested Monte-Carlo search," *Int. J. Electron. Telecommun.*, vol. 64, no. 2, pp. 147–150, 2018.

[5] A. D. Dwivedi and G. Srivastava, "Differential cryptanalysis of round-reduced LEA," *IEEE Access*, vol. 6, pp. 79105–79113, 2018.

[6] A. Biryukov and V. Velichkov, "Automatic search for differential trails in ARX ciphers," in *Proc. Cryptographer's Track RSA Conf. (CT-RSA)*, in Lecture Notes in Computer Science, San Francisco, CA, USA, vol. 8366, J. Benaloh, Ed. Springer, Feb. 2014, pp. 227–250.

[7] A. Biryukov, V. Velichkov, and Y. Le Corre, "Automatic search for the best trails in ARX: Application to block cipher SPECK," in *Proc. Int. Conf. Fast Softw. Encryption*. Berlin, Germany: Springer, 2016, pp. 289–310.

[8] L. Song, Z. Huang, and Q. Yang, "Automatic differential analysis of ARX block ciphers with application to SPECK and LEA," in *Proc. ACISP*, in Lecture Notes in Computer Science, vol. 9723. Springer, 2016, pp. 379–394.

[9] J. Wu, S. Luo, S. Wang, and H. Wang, "NLES: A novel lifetime extension scheme for safety-critical cyber-physical systems using SDN and NFV," *IEEE Internet Things J.*, to be published.

[10] J. Wu, M. Dong, K. Ota, J. Li, and Z. Guan, "Big data analysis-based secure cluster management for optimized control plane in software-defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 1, pp. 27–38, Mar. 2018.

[11] J. Wu, M. Dong, K. Ota, J. Li, and Z. Guan, "FCSS: Fog computing based content-aware filtering for security services in information centric social networks," *IEEE Trans. Emerg. Topics Comput.*, to be published.

[12] J. Wu, K. Ota, M. Dong, and C. Li, "A hierarchical security framework for defending against sophisticated attacks on wireless sensor networks in smart cities," *IEEE Access*, vol. 4, pp. 416–424, 2016.

[13] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The SIMON and SPECK lightweight block ciphers," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.

[14] F. Abed, E. List, S. Lucks, and J. Wenzel, "Differential cryptanalysis of round-reduced SIMON and SPECK," in *Proc. Int. Workshop Fast Softw. Encryption*. London, U.K.: Springer, 2014, pp. 525–545.

[15] D. Dinu, Y. Le Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, "Triathlon of lightweight block ciphers for the Internet of Things," *J. Cryptograph. Eng.*, pp. 1–20, 2015.

[16] H. Lipmaa and S. Moriai, "Efficient algorithms for computing differential properties of addition," in *Proc. FSE*, in Lecture Notes in Computer Science, vol. 2355. Springer, 2001, pp. 336–350.

[17] R. Coulom, "Computing 'Elo ratings' of move patterns in the game of go," *ICGA J.*, vol. 30, no. 4, pp. 198–208, 2007.

[18] I. Dinur, "Improved differential cryptanalysis of round-reduced SPECK," in *Selected Areas in Cryptography* (Lecture Notes in Computer Science), vol. 8781. Springer, 2014, pp. 147–164.

ASHUTOSH DHAR DWIVEDI received the B.Sc.(Maths) degree from the Ewing Christian College (an autonomous college of the University of Allahabad), Allahabad, India, and the M.C.A. degree from the Amity School of Computer Sciences, Noida, India, in 2013. He is having a rich experience of industry and academia of around six years. He was an Intern (under his master's project) with the prestigious organization "Centre for Railway Information Systems, New Delhi," governed by the Ministry of Railways, India. He was with organizations related to software development projects for two years. In 2015, he moved to Poland and started career in cryptography research. He is currently a Cryptography Researcher with the Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland, where he focuses on symmetric-key cryptography. He is also a Visiting Researcher with the Department of Mathematics and Computer Science, Brandon University, Brandon, Canada, where he is doing research focusing on blockchains, bitcoin, and game theory. His primary research interests include symmetric-key cryptography, cryptanalysis of block ciphers, blockchains, and bitcoin. He has been serving as a Reviewer for few international journals and conferences in the area of cryptography, including the IEEE Access.

PAWEL MORAWIECKI is currently an Associate Professor with the Institute of Computer Science, Polish Academy of Sciences. He is one of the designers of cipher ICEPOLE: high-speed, hardware-oriented authenticated encryption. He has several publications in top venues, such as Eurocrypt, FSE, and CHES. He is currently very interested in the intersection between machine learning and security. His main field of expertise is cryptography and cryptanalysis.

GAUTAM SRIVASTAVA received the B.Sc. degree from Briar Cliff University, USA, in 2004, and the M.Sc. and Ph.D. degrees from the University of Victoria, Victoria, BC, Canada, in 2006 and 2011, respectively. He then taught for three years with the Department of Computer Science, University of Victoria, where he was regarded as one of the top undergraduate professors in the computer science course instruction. In 2014, he joined a tenure-track position with Brandon University, Brandon, MB, Canada, where he was promoted to an Associate Professor, in 2018, and is currently active in various professional and scholarly activities. He is active in research in the field of data mining and big data. In his seven-year academic career, he has published a total of 33 papers in high-impact conferences in many countries and in high-status journals and has also delivered invited guest lectures on big data at many Taiwanese universities. He is an Editor of several international scientific research journals. He currently has active research projects with other academics in Taiwan, Singapore, Canada, Czech Republic, Poland, and USA. He is constantly looking for collaboration opportunities with foreign professors and students. Dr. Srivastava received the Best Oral Presenter Award from the FSDM 2017 which was held at National Dong Hwa University, Shoufeng (Hualien County), Taiwan, in 2017.