Differential Cryptanalysis in ARX Ciphers with specific applications to LEA

Ashutosh Dhar Dwivedi[1,2], Gautam Srivastava[2,3]

[1] Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
[2] Department of Mathematics and Computer Science, Brandon University, Brandon, Manitoba, Canada
[3] Research Center for Interneural Computing, China Medical University, Taichung, Taiwan, Republic of China

Keywords: Differential characteristics, Nested Monte-Carlo Search, ARX ciphers, LEA Cipher, Block Cipher

Abstract

In this paper, we focus on differential cryptanalysis dedicated to a particular class of cryptographic algorithms, namely ARX ciphers. We propose a new algorithm inspired by the Nested Monte-Carlo Search algorithm to find a differential path in ARX ciphers. We apply our algorithm to round reduced variant of the block cipher LEA. We use the concept of a partial difference distribution table (pDDT) in our algorithm to reduce the search space. This methodology reduced the search space of the algorithm by using only those differentials whose probabilities are greater than or equal to a pre-defined threshold. Using this concept, we removed many differentials which are not valid or whose probabilities are very low. By doing this, we decreased the time of finding a differential path by our nested algorithm due to a smaller search space. This partial difference distribution table also made our nested algorithm suitable for bigger block size ARX ciphers. Finding long differential characteristics is one of the hardest problems where we have seen other algorithms take many hours or days to find differential characteristics in ARX ciphers. Our algorithm finds the differential characteristics in just a few minutes with a very simple framework. We report the differential path for up to 9 rounds in LEA. To construct differential characteristics for a large number of rounds, we divide long characteristics into short ones, by constructing a large characteristic from two short characteristics. Instead of starting from the first round, we start from the middle and run experiments in forward as well as in the reverse direction. Using this method, we improved our results and reported the differential path for up to 12 rounds. Overall, the best property of our algorithm is that it has the potential to provide state-of-the-art results but within a simpler framework as well as less time. Our algorithm is also very interesting for the future aspect of research, as it could be applied to other ARX ciphers with a very easy going framework.

1 Introduction

ARX which stands for Addition/Rotation/XOR, is a class of symmetric-key algorithms designed using only the following simple operations: modular addition, bitwise rotation and exclusive-OR. In academia and industry alike, ARX has gained an enormous amount of interest because of its small size and simple operations. By using combined linear and non-linear operations by XOR, bit shift, bit rotation and modular addition (iterating them for many rounds), ARX has become very resistant against both linear and differential cryptanalysis. ARX has no look-up table like S-box based algorithms, and therefore these ciphers are very secure against well-known side channel attacks.

On one hand, the cryptanalysis of typical S-box based algorithms is easy because it consists of four or eight-bit words. On the other hand, the linear or differential cryptanalysis of ARX ciphers is much more difficult. By calculating the linear approximation table (LAT) or differential distribution table (DDT) it is easy to evaluate linear and differential properties of S-box. But to calculate such tables for ARX even for a 32-bit word is not feasible. However, a partial difference distribution table (pDDT) having just a few fractions of all the differentials is possible. This table contains only such differentials which have a probability greater than some fixed threshold. This is 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 our analysis, we focus on the LEA cipher introduced in [11]. LEA has 128-bit block size and 128, 192, or 256-bit key size. It provides encryption on general purpose processors with high-speed software. LEA is faster compared to AES on ARM, AMD, Coldforms and Intel platforms. LEA consists of only three ARX operations (modular Addition, bitwise Rotation, and bitwise XOR) which operate on 32-bit words. Those operations are fast and very well supported in many 32-bit or 64-bit platforms.

In this paper, we propose an algorithm based on the nested method to find good differential characteristics in ARX ciphers. We face a huge state space problem in finding a good differential path and there is no obvious way to proceed to the next step. These kinds of problems are present in many different fields, but we are inspired by single-player games such as Morpion solitaire, SameGame and Sudoku. We are inspired by the heuristics of the Nested Monte-Carlo Search which works very well for these games are was shown in [7]. We also create a nested algorithm based on the technical complexity of our problem.

In our previous work [9], we applied a naive algorithm and succeeded to find differential characteristics only for smaller state size in SPECK. For other variants, because of the bigger state size and due to large state space the algorithm given was not able to produce results even after applying for a long period of time. Constructing a complete Difference Distribution Table (DDT) for n-bit words, the modular addition would require $2^{3n} \times 4$ bytes of memory and also not feasible to create such a big table. Therefore, we used the concept of partial difference distribution table (pDDT) [5] in this paper where we used those differentials which are valid and have a probability greater than some specified threshold.

Similarly, we were inspired by the concept of highways and country roads analogy proposed by Biryukov et al. in two related papers [5][6]. The problem of finding a differential path in a cipher is the same as finding fast routes between two cities. Differentials that have high probability can be treated the same as highways, and low probability differentials can be treated the same as slow roads. In the same paper the author also introduced the concept of partial difference distribution table (pDDT). Therefore in our algorithm, we try to find those differentials which have high probability, and if such differentials do not exist, we, in turn, use low probability values. Therefore, we find no need to take completely random decisions for our advanced nested algorithm. This improves the decision process of the algorithm.

2 Related Cryptanalysis

Along the lines of ARX ciphers, there have been many works. Some examples of ARX ciphers are:

- block cipher SPECK [2]
- block cipher LEA [11]
- block cipher Chaskey [14]
- stream cipher Salsa20 [3]
- SHA-3 finalist Skein [10]
- SHA-3 finalist Blake [1]

Differential cryptanalysis was introduced first by Biham and Shamir in their pivotal work [4]. For block ciphers, it is used to analyze how input differences lead to output differences. If for example, certain input/output differences happen in a non-random way, it can be used to build a distinguisher or even help recover keys.

To consider the security of iterated block ciphers against differential cryptanalysis, Lai et al. were the pioneers in using the theory of Markov ciphers and made a distinction between a differential and a differential characteristic [12]. A differential is a difference propagation from an input difference to an output difference. On the other hand, a differential characteristic specifies not only the input/output difference but also all the internal differences after each round. For a Markov cipher, the probability of a differential characteristic is the multiplication of difference transition probabilities of each round, and the probability of a differential is equal to the sum of the probabilities of all differential characteristics which correspond to the differential.

In direct relation to our work here, Song et al. [16] presented a paper where they developed Mouha et al.'s framework from [14] for finding differential characteristics by adding a new method to construct long characteristics from short ones. They reported the probabilities of the best differential trails of LEA for up to 13 rounds with total weight -132 and probability $2^{-123.79}$.

Hong et al. [11], the designers of LEA did differential cryptanalysis of LEA cipher and reported the differential path up to 11 rounds with the probability 2^{-98}. They also did differential-linear, impossible differential and linear cryptanalysis and presented paths for 14, 10 and 11 rounds.

3 Description of LEA

LEA given first in [11] provides a high-speed encryption with software and general purpose processors. It has a block size of 128 bits with key sizes of 128 bit, 192 bit and 256 bits.

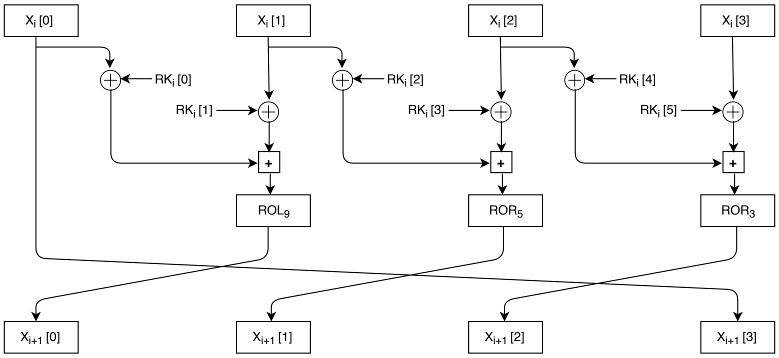


Fig. 1: The round function of LEA.

LEA plaintexts are divided into four 32 bit words $(x_0^0, x_1^0, x_2^0, x_3^0)$ and after encryption they produce ciphertexts$(x_0^r, x_1^r, x_2^r, x_3^r)$, where r represents rounds and the value of r is 24 for LEA-128. The Round function for round i, $0 \leq i < r$ is defined by:

$$x_0^{i+1} \leftarrow ((x_0^i \oplus rk_0^i) \boxplus (x_1^i \oplus rk_1^i) \lll 9 \tag{1}$$

$$x_1^{i+1} \leftarrow ((x_1^i \oplus rk_2^i) \boxplus (x_1^i \oplus rk_3^i) \lll 5 \tag{2}$$

$$x_2^{i+1} \leftarrow ((x_2^i \oplus rk_4^i) \boxplus (x_1^i \oplus rk_5^i) \lll 3 \tag{3}$$

$$x_3^{i+1} \leftarrow x_0^i \tag{4}$$

where $rk^i = (rk_0^i, rk_1^i, rk_2^i, rk_3^i, rk_4^i, rk_5^i)$ is the round key generated by the key schedule.

4 Calculating Differential Probabilities

In [13], Moriai and Lipmaa studied the differential properties of addition. Let $xdp^+(a, b \to c)$ be the XOR-differential probability of addition modulo 2^n, with input differences a and b and output difference c. Moriai and Lipmaa proved that the differential $(a, b \to c)$ is valid if and only if:

$$eq(a \ll 1, b \ll 1, c \ll 1) \wedge (a \oplus b \oplus c \oplus (b \ll 1)) = 0 \tag{5}$$

where

$$eq(p, q, r) := (\neg p \oplus q) \wedge (\neg p \oplus r) \tag{6}$$

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

$$w(a, b \to c) = -\log_2(xdp^+(a, b \to c)) \tag{7}$$

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

$$w(a, b \to c) := h^*(\neg eq(a, b \to c)), \tag{8}$$

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.

5 Partial Difference Distribution Tables (pDDT)

Partial difference distribution table (pDDT) proposed by Biryukov et al. [5] 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}.

$$(a, b, c) \in pDDT \Leftrightarrow DP(a, b \to c) \geq p_{thres} \tag{9}$$

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.

$$p_n \leq .... \leq p_k \leq p_{k-1} \leq .... \leq p_1 \leq p_0 \tag{10}$$

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 $a_0, b_0, c_0 = \phi$.

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$(a, b \to c) \geq p_{thres}$.
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
        $a_{k+1} \longleftarrow x|a_k, b_{k+1} \longleftarrow y|b_k, c_{k+1} \longleftarrow z|c_k$
        $p_{k+1} = DP(a_{k+1}, b_{k+1} \to c_{k+1})$
        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 1: Timings on the computation 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.

In our nested algorithm, we set the threshold value equal to 0.1 and therefore the size of our algorithms search space is equal to 3951388 from Table 1. 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.

6 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) [15]. For single-player games, a variant called the Nested Monte-Carlo Search has been proposed [7].

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 to the best score of the lower level searches, that is, a nested of level 1 makes a playout for every possible move and choose to play the move of the best playout. A nested of level 2 does the same thing except that is replaces the playout by a nested of level one.

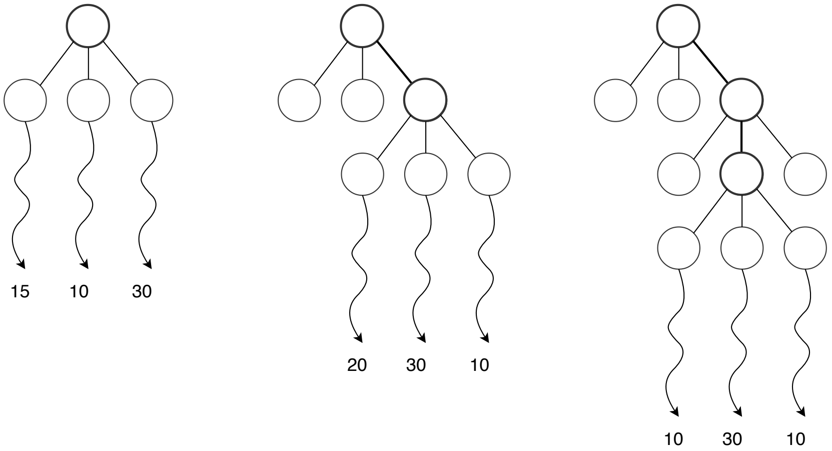


Fig. 2: Nested Monte Carlo Search.

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

Searching for a differential path with high probability is also a type of problem where we face huge state space. Typically, there is not a clear and obvious way to proceed in such kind of problems, and we hoped NMCS would be helpful for our problem. However, the NMCS algorithm tries all possible moves at each level, but in our problem of finding the differential path, it is not possible due to the large size of blocks.

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.

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 lists shows a total distance travelled. Initially, both the lists are empty.

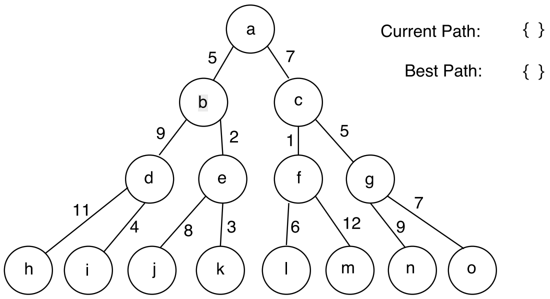


Fig. 3: Different paths from the root (base node) to the destination (leaf nodes)

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 the 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 4.

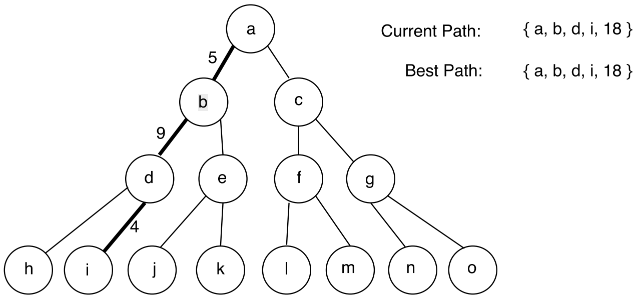


Fig. 4: Random path from the base node to the leaf node.

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 5.)

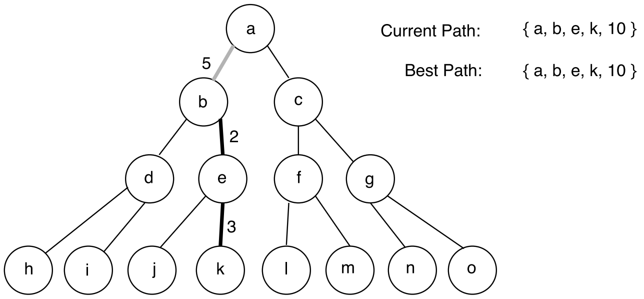


Fig. 5: A random path from the b node to the leaf node.

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 6.) The score for $CurrentPath$ is 15, which is not better than the previous best path score. Thus we do not update $BestPath$.

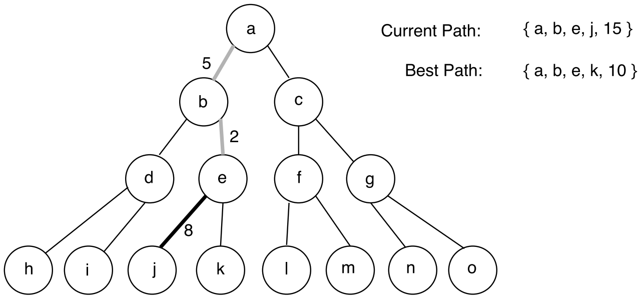


Fig. 6: Random path from node e to leaf node.

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 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 an 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 our Algorithm based on NMCS

We will define two functions, *RandomPath*(*node_position*) and *Nested*(*node_position*) to describe our algorithm. These two functions are the main building blocks of the algorithm. The first function *RandomPath*(*node_position*) walks a random path in the search tree from a given node, and it walks until it reaches the leaf node. A list of nodes (from the base node to the leaf) and the cost corresponding to the path are returned by function *RandomPath*.

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 is *Nested*(*node_position*) which is a recursive function. *Nested*(*node_position*) calls itself on every level of the tree search until it reaches the leaf node. **Algorithm 2** represents the pseudo-code of the function. We use two global variables in the given pseudo-code which are responsible for keeping a list of nodes in the best path, namely (*best_path*) and the cost (*best_cost*) corresponding to the given best path. Initially, *best_cost* is initialized with a large value and *best_path* is empty. We assume here 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
```

Until we meet our criterion, the Nested function can be called iteratively in a loop. The criterion itself is variable. It could be for example a number of iterations, a given time limit or even the maximum cost of the best path. The Nested Search could be also easily be 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

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

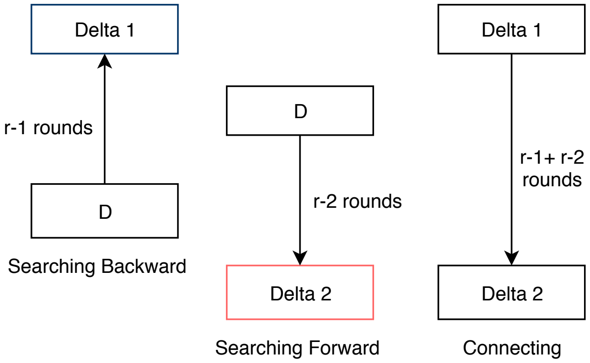
8 Finding Differential Paths

Finding differential characteristics in algorithms could be treated as a single player game. We start with input difference, and after each round of cipher, a decision is mandatory. Here, the decision means selection of input-output transition through the non-linear part of the round. In ARX cipher, modular addition is the source of non-linearity and therefore transition through this modular addition represents the decision part in the algorithm. Generally, for an input difference, there might be many valid output differences. We explained this in Section 4. Each transition through the modular addition is probabilistic. Transition with a very low probability has a very high cost and vice-versa. By multiplying the probabilities associated with all transitions through modular addition, we calculate the total cost of the path. To increase the speed of our nested algorithm and to produce better results for larger block size, we reduced the search space by using partial difference distribution table called pDDT. We note here that creating a complete difference distribution table is infeasible because of the large block size in ARX ciphers. This table only contains valid and high differential transitions. Therefore, our nested algorithm does not waste time in taking random values for the transitions which are not valid. For a 32 -bit block instead of taking transition values from the 2^{32} items (many transitions are not valid) it will only take those transitions (from pDDT table) which are valid and has some probability greater than a pre-defined threshold probability. We aim to find the differential path for a given number of rounds with the highest probability.

9 Obtaining Long Characteristic

It is a well known fact that it is easier to find a short characteristic (for a small number of rounds) instead of a long characteristic. Therefore, we use the start-in-the-middle approach to find a long characteristic from two short ones. In this method, we start our algorithm from the middle of the rounds in two directions, forward toward the end and at the same time backwards towards the beginning.

Fig. 7: Obtaining a longer characteristic from two shorter ones



In this experiment we apply internal difference inputs from the middle of the given number of rounds. For example, if we want to find a path for 14 rounds then we pass inputs to our algorithm and let it run for 7 rounds forwards and 7 rounds backwards (reverse). Once we acquire the result for both directions, we combine them together to get a long characteristic for the 14 rounds (see figure 7). This method also saves time by not needing to search long characteristics and in the end provides us with a better result.

10 Results

In this paper, we used our algorithm with the partial difference distribution table (pDDT) for finding the best differential trails in the ARX cipher LEA. We showed the practical application of the new method on round-reduced variants of the block cipher LEA. For the 128-bit state of the cipher, it only makes sense to analyse the differential paths with probability higher than 2^{-128}. This is true because clearly, we can see that a path with lower probability would not lead to any meaningful attack, and there would be faster than an exhaustive search in the 128-bit state. We run the experiments for long characteristics starting from the first round. We report the differential path for up to 9 rounds. In Table 2, each block represents a 32-bit state size. Differences are encoded as hexadecimal numbers and probability for a given weight is given by 2^{weight}. We use a threshold value for pDDT table equal to 0.1 and our search space contains 3951388 values with probability greater than or equal to 0.1. For any larger threshold value, the pDDT size is small, and the speed of the experiment is fast because the algorithm takes values from a smaller search space. However, increasing the threshold value may lead to missing values which are not in the table may be necessary to make a good differential path. Conversely setting the threshold value for the pDDT table as too small may lead to the experiment speed being slow because of bigger search space. So there is a balance here to make sure and set the threshold properly to include the values which are necessary to make the good differential path.

Table 2: Differential trails for LEA

Round	**Block1**	**Block2**	**Block3**	**Block4**	**log$_2$p**
1	80000000	80000000	80000000	80000000	0
2	00000000	00000000	00000000	80000000	0
3	00000000	00000000	10000000	00000000	-3
4	00000000	01800000	02000000	00000000	-6
5	00000001	00040000	00040000	00000000	-5
6	08000200	00022000	00080000	00000001	-9
7	04440010	00005100	20010000	08000200	-16
8	88a22008	01000a88	05002040	04440010	-25
9	44550113	40200156	0028840a	88a22008	-34
				weight	-98

In the second part of our experiment, we also experiment starting from the middle of the rounds and run our tool in both directions. Using this method, we improved our results and reported the differential path for up to 12 rounds in Table 3.

Table 3: Differential trails for LEA

Round	**Block1**	**Block2**	**Block3**	**Block4**	**log$_2$p**
1	c0402234	80052234	88013224	8a0022a0	-27
2	8a000080	80402080	80402210	c0402234	-17
3	80400014	80000014	88000004	8a000080	-10
4	80000000	80400000	80400010	80400014	-6
5	80000000	80000000	80000000	80000000	0
6	00000000	00000000	00000000	80000000	0
7	00000000	00000000	10000000	00000000	-3
8	00000000	01800000	02000000	00000000	-6
9	00000003	00040000	00400000	00000000	-6
10	08000200	00022000	00080000	00000003	-10
11	04440010	00005100	20010000	08000200	-16
12	88a22008	01000a88	05002040	04440010	-25
				weight	-126

11 Differential Attacks

Dinur proposed an enumeration technique [8] for key recovery in differential attacks against SPECK. The differential equations of addition in the key recovery of LEA are same as SPECK, therefore we can apply the same technique to recover the key in LEA cipher. Consider we have given r-round characteristics of LEA which has probability $p > 2 \cdot 2^{-N}$, where N represents the block size of cipher.

For LEA-128 and LEA-192, the attack can recover the key of a variant of $(r+1)$ rounds. The number of plaintexts required to attack will be $2 \cdot p^{-1}$, with time complexity equal to $2 \cdot p^{-1}$ encryptions. For LEA-256, the attack can recover the key of $(r+2)$ rounds with $2 \cdot p^{-1}$ plaintexts and $2 \cdot p^{-1} \cdot 2^{N}$ encryptions.

Table 4: Differential attack from 9 round characteristics

Variants	Rounds attacked	Time	Data	Memory
LEA-128	10/24	2^{99}	2^{99}	2^{22}
LEA-192	10/28	2^{99}	2^{99}	2^{22}
LEA-256	11/32	2^{227}	2^{99}	2^{22}

Table 5: Differential attack from 12 round characteristics

Variants	Rounds attacked	Time	Data	Memory
LEA-128	13/24	2^{127}	2^{127}	2^{22}
LEA-192	13/28	2^{127}	2^{127}	2^{22}
LEA-256	14/32	2^{255}	2^{127}	2^{22}

12 Conclusion

In this work, we applied our algorithm to find differential characteristics to ARX block ciphers. We were able to find a differential path of 9 rounds with probability 2^{-98} for a LEA cipher. Moreover, we also used a method for constructing long characteristics from short ones in our algorithm. This modification enhanced our results reporting a path of 12 rounds with probability 2^{-126}. We used a partial difference distribution table (pDDT) to reduce the size of the search space in our algorithm. We can produce faster results compared to any other methods of finding a differential path. We have been successfully able to get results in just a few minutes using our algorithm in scenarios which what would have previously taken a few hours.

Acknowledgement

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

References

1. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: Sha-3 proposal blake. Submission to NIST (2008)
2. Beaulieu, R., Treatman-Clark, S., Shors, D., Weeks, B., Smith, J., Wingers, L.: The simon and speck lightweight block ciphers. In: Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE. pp. 1–6. IEEE (2015)
3. Bernstein, D.J.: The salsa20 family of stream ciphers. In: New stream cipher designs, pp. 84–97. Springer (2008)
4. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptology 4(1), 3–72 (1991)

5. Biryukov, A., Velichkov, V.: Automatic search for differential trails in ARX ciphers. In: CT-RSA. Lecture Notes in Computer Science, vol. 8366, pp. 227–250. Springer (2014)
6. Biryukov, A., Velichkov, V., Corre, Y.L.: Automatic search for the best trails in ARX: application to block cipher speck. In: FSE. Lecture Notes in Computer Science, vol. 9783, pp. 289–310. Springer (2016)
7. Cazenave, T.: Nested monte-carlo search. In: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 456–461 (2009)
8. Dinur, I.: Improved differential cryptanalysis of round-reduced speck. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 8781, pp. 147–164. Springer (2014)
9. Dwivedi, A.D., Morawiecki, P., Wójtowicz, S.: Finding differential paths in arx ciphers through nested monte-carlo search. International Journal of electronics and telecommunications 64(2), 147–150 (2018)
10. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The skein hash function family. Submission to NIST (round 3) 7(7.5), 3 (2010)
11. Hong, D., Lee, J., Kim, D., Kwon, D., Ryu, K.H., Lee, D.: LEA: A 128-bit block cipher for fast encryption on common processors. In: WISA. Lecture Notes in Computer Science, vol. 8267, pp. 3–27. Springer (2013)
12. Lai, X., Massey, J.L., Murphy, S.: Markov ciphers and differential cryptanalysis. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 17–38. Springer (1991)
13. Lipmaa, H., Moriai, S.: Efficient algorithms for computing differential properties of addition. In: FSE. Lecture Notes in Computer Science, vol. 2355, pp. 336–350. Springer (2001)
14. Mouha, N., Mennink, B., Van Herrewege, A., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: an efficient mac algorithm for 32-bit microcontrollers. In: International Workshop on Selected Areas in Cryptography. pp. 306–323. Springer (2014)
15. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature 529(7587), 484–489 (2016)
16. Song, L., Huang, Z., Yang, Q.: Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In: ACISP (2). Lecture Notes in Computer Science, vol. 9723, pp. 379–394. Springer (2016)