# Efficient Differential Trail Searching Algorithm for ARX Block Ciphers

Seojin Kim,[1†] HyungChul Kang,[1] Deukjo Hong,[2‡] Jaechul Sung,[3] Seokhie Hong,[1]

[1]Graduate School of Information Security, Korea University

[2]Department of Information Technology, Chonbuk National University

[3]Department of Mathematics, University of Seoul

ABSTRACT

In this paper, we suggest an advanced method searching for differential trails of block cipher with ARX structure. We use two techniques to optimize the automatic search algorithm of differential trails suggested by Biryukov et al. and obtain 2~3 times faster results than the previous one when implemented in block cipher SPECK.

**Keywords: ARX structure, Differential trails, Automatic search algorithm, SPECK**

## I. Introduction

Differential Cryptanalysis is one of the most powerful methods for the analysis of block ciphers. It uses a differential charateristic, which is a transition from the input difference to the output difference. The probability of a differential characteristic is estimated as the product of probabilities for nonlinear operations like S-boxes. So, the attack begins with computing differential distribution tables of nonlinear operations.

ARX cipher consists of modular Addition, Rotation, and eXclusive-or. Since these computations are rather simple, ARX cipher is commonly applied to a variety of light-weight block ciphers[6, 7].

In this structure, modular addition takes the role of non-linear computation like S-box in other ciphers, so considering input and output differences of modular addition is needed. Calculating differential probability of modular addition, unlike simple S-box, two inputs and one output differences should be considered, and input and output bit size is much bigger than S-box. As a result, calculating differential probability of modular addition requires much more complexity. Therefore differential analysis of ARX cipher should begin with another approach, and Biryukov et al. suggested new method[2]. The method suggested in [2] is fairly efficient, but the more rounds are required to be analyzed, the exponentially more time is needed.

In the paper, we suggests an algorithm that improves algorithm suggested by Biryukov in aspect of computing time. Suggesting algorithm uses two techniques to reduce time complexity.

First, we introduce properties of differential probability of modular

addition, and describe block cipher SPECK in Section 2. Then we explain the algorithm suggested by Biryukov and new improved algorithm in Section 3, and 4 respectively. Finally, we concludes the paper in Section 5.

## II. Related Works

### 2.1. Notation

The notation used in this paper is as follows:

- o $\oplus$: XOR(eXclusive-Or) computation
- o $\wedge$: And computation
- o $\neg$: Complementary
- o $x|y$: Concatenation of bit string $x$, $y$
- o $x \gg i$ $(x \ll i)$: Bitwise shift operation to the right (left) by i
- o $x \ggg i (x \lll i)$: Bitwise rotation operation to the right (left) by i
- o $w$: Word size
- o $a_i$: $i$-th bit of word $a$
- o $x[i:1]$: The sequence of bits $x[i], x[i-1], \ldots, x[1]$

### 2.2. Block Cipher SPECK

SPECK is a family of lightweight block ciphers publicly released by National Security Agency (NSA) in June 2013[3]. SPECK supports a variety of block and key sizes.

Fig. 1 shows round function of SPECK. In this paper, we analyze SPECK32, and SPECK48 that consist of 32, and 48 block sizes respectively.

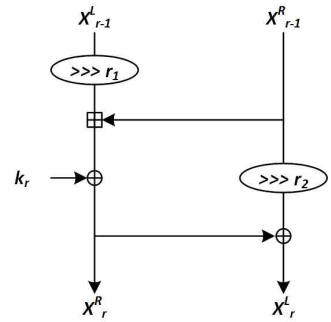


Fig. 1. Round Function of SPECK

In Fig. 1, $X_L$ and $X_R$ means 16-bit word in SPECK32, and 32-bit word in SPECK48. $k_r$ means round key of $r$-th round.

$r$-th round can be expressed as follows:

$$X_L^r = ((X_L^{r-1} \ggg r_1) \boxplus X_R^{r-1}) \oplus k_r$$
$$X_R^r = (X_R^{r-1} \lll r_2) \oplus X_L^r$$

Rotation constant $r_1$ and $r_2$ are $r_1 = 7$, $r_2 = 2$ in SPECK32, and $r_1 = 8$, $r_2 = 3$ in rest of SPECK.

Since key schedule of SPECK has little thing to do with searching differential path, for more details about key schedule, see [3].

### 2.3. Properties of Modular Addition

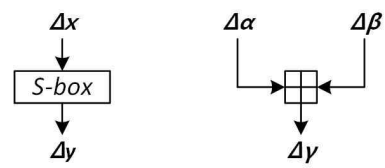


Fig. 2. Input/Output difference on S-box and modular addition

XOR difference of modular addition requires two input difference and one output difference unlike S-box requiring one input and output difference. XOR difference of S-box can be calculated

through exhaustive search of input difference and corresponding output difference. This computation requires $2^{n+1}$ complexity for $n$-bit input size. In case of modular addition, since we need to consider two inputs, complexity increases to $2^{2n+2}$. Although the greater $n$ increases, the more complexity is required, H. Lipmaa proposed an algorithm that decreases complexity to $n$[4].

First, we define eXclusive-or Differential Probability, $xdp$, of two input differences, $\alpha$ and $\beta$, and output difference $\gamma$, as follows:

Definition 1. [4] Given two input differences $\alpha$ and $\beta$, and an output difference $\gamma$, $xdp$ can be calculated as:

$$xdp(\alpha,\beta\rightarrow\gamma) = \Pr[((x\oplus\alpha)+(y\oplus\beta))\oplus(x+y)=\gamma]$$

Also, $xdp$ value an be found looking at carries occurring when operating modular addition and difference of carries.

Theorem 1. [4] Given two input difference $\alpha$ and $\beta$, and an output difference $\gamma$, for $x,y\in\{0,1\}^w$, if $carry(x,y):=c\in\{0,1\}^w$, then $xdp(\alpha,\beta\rightarrow\gamma) = \Pr[carry(x,y)\oplus carry(x\oplus\alpha,y\oplus\beta)=(\alpha\oplus\beta\oplus\gamma)]$ where $c_0=0$ and $c_{i+1}:=(x_i\wedge y_i)\oplus(x_i\wedge c_i)\oplus(y_i\wedge c_i)$.

That is, $xdp$ calculation requires $n$ complexity, for $n$-bit $xdp$ calculation checks $n$-bit information, $carry$.

Theorem 2. [4] Given two input difference $\alpha$ and $\beta$, an output difference $\gamma$, every possible (every differential probability that is not zero) differential probability satisfies following equation. Also the converse is true.

$eq(\alpha\ll 1,\beta\ll 1,\gamma\ll 1)\wedge((\alpha\oplus\beta\oplus\gamma)\oplus(\alpha\ll 1))=0$
where, $eq(x,y,z):=(\neg x\oplus y)\wedge(\neg x\oplus z)$.
(i.e. $eq(x,y,z)_i=1\Leftrightarrow x_i=y_i=z_i$)

Therefore if $xdp$ of two input difference $\alpha$ and $\beta$, and an output difference $\gamma$ is not zero, then for $i\in\{0,1,...,w-1\}$, $\alpha_{i-1}=\beta_{i-1}=\gamma_{i-1}=\alpha_i\oplus\beta_i\oplus\gamma_i$.

Two theorems above is verified in [4], and in [4], Lipmaa proposed an algorithm calculating $xdp$.

**Algorithm 1. [4] Log-time Algorithm for xdp**

**Input:** $\delta=(\alpha,\beta\rightarrow\gamma)$
**Output:** $xdp(\delta)$
1. If $eq(\alpha\ll 1,\beta\ll 1,\gamma\ll 1)\wedge((\alpha\oplus\beta\oplus\gamma)\oplus(a\ll 1))\neq 0$, then return 0;
2. Return $2^{-W_h(\neg eq(\alpha,\beta,\gamma)\wedge(2^w-1))}$

$W_h(x)$: **Hamming weight of** $x$

## III. Algorithm suggested by Biryukov[2]

Differential attack begins with generating difference distribution table, DDT, of non-linear operation. In case of S-box, if input difference has $n$-bit, and output difference has $m$-bit, then the size of DDT is $n\times m$-bit. On the other hand, DDT of modular addition requires two input and on output difference of $n$-bit, making size of DDT $2^{3n}$-bit. If input and output difference consist of more than 16-bit each, DDT should have more than $2^{48}$ elements, requiring more than 32TB memory. Biryukov suggested new method that can find differential trail not using DDT[2].

Theorem 3. [2] $xdp$ decreases, as size of input and output difference increase. Therefore, given input and output

differences, $\alpha$, $\beta$, and $\gamma$, the following equation is satisfied.

$p_w \leqq p_{w-1} \leqq p_{w-2} \cdots \leqq p_1$,
where $p_i = xdp(\alpha[i:1], \beta[i:1] \rightarrow \gamma[i:1])$

Biryukov et al. use theorem 3 to search for differential trail bit by bit based on a branch-and-bound search strategy. For $r$-th round, given input difference, we can calculate each $xdp$ adding one bit into output difference. If calculated $xdp$ is greater than the bound probability, we add one more bit to output difference or go to the next round. In case of first round, we need to consider the input differences also.

The bound probability, $B_r$, is the best probability of $r$-th round differential trail that is already known. For the first round, we consider every input and output difference pair that the differential probability, $xdp_1$, satisfies the inequality $xdp_1 \times B_{r-1} \geqq B_r$. For the $j(2 < j < r)$-th round we consider output difference that the $xdp_j$ satisfies the inequality $xdp_1 \times xdp_2 \times \cdots \times xdp_j \times B_{r-j} \geqq B_r \times xdp_j \times B_{r-j} \geqq B_r$. Finally for the last round, consider the output difference that satisfies inequality $xdp_1 \times xdp_2 \times \cdots \times xdp_r \geqq B_r$, and if such output difference exists, update the new bound $B_r$ and save the path obtained.

For unknown $B_r$, set $B_r = B_{r-1}$ and search for the trail. If there exists no such trail, then decrease $B_r \leftarrow B_r \times 1/2$, and then do the search algorithm repeatedly. The Algorithm that Biryukov applied to SPECK can be found in Appendix (Appendix A. Algorithm 2).

Table 1. Searching time of Algorithm 2〔2〕 (Intel Core™ E5-2637 CPU 3.50GHz)

| $r$ | SPECK32 | | SPECK48 | |
|---|---|---|---|---|
| | Prob.($\log_2(p)$) | time | Prob.($\log_2(p)$) | time |
| 1 | 0 | 0sec | 0 | 0sec |
| 2 | -1 | 0sec | -1 | 0sec |
| 3 | -3 | 0sec | -3 | 0sec |
| 4 | -5 | 0sec | -6 | 0sec |
| 5 | -9 | 0sec | -10 | 1sec |
| 6 | -13 | 1sec | -14 | 3sec |
| 7 | -18 | 1min | -19 | 1min |
| 8 | -24 | 34min | - | |

## IV. Suggesting Algorithm

In this chapter, we introduce how to reduce computation complexity of algorithm 2 given in〔2〕. In particular, we describe two techniques. One is reduce the complexity by not calculating impossible trails. The other is calculating $xdp$ bitwise.

### 4.1 Optimizing Differential Trail Searching

In order to take shorter time, one of the main ideas suggested in this paper is to discard impossible differential trails (zero probability differential trails).

In Algorithm 2, one bit (0 or 1) is added to a trail and check whether $xdp$ is 0 or not. If the probability is 0, then the algorithm returns and consider the next bit. For $i$-bit word, the $xdp$ computation complexity, $i$, is necessary.

However in Algorithm 1, $xdp$ is first calculated to check if $xdp$ is zero or not (1.). This part can be expressed as following theorem.

Theorem 4.〔5〕 $xdp(\alpha, \beta \rightarrow \gamma) \neq 0$
iff $(\alpha[0] \oplus \beta[0] \oplus \gamma[0]) = 0$ and $\alpha[i-1] = \beta[i-1] = \gamma[i-1] = \alpha[i] \oplus \beta[i] \oplus \gamma[i]$
for $\alpha[i-1] = \beta[i-1] = \gamma[i-1]$, $i \in \{0, \ldots, n-1\}$

According to Theorem 4, we can expect the next possible bit by checking previous bit($i-1$-th bit). For example, if $i$-th bit of input differences $\alpha$, $\beta$, and output difference $\gamma$ is $(\alpha_i,\beta_i,\gamma_i)=(0,0,0)$, then $i+1$-th bit of input and output differences can be one of these four occasions - $(0,0,0)$, $(0,1,1)$, $(1,0,1)$, $(1,1,0)$.

Using this technique, we can delete $xdp$ computation in (11.), (18.), (22.), (34.) of Algorithm 2. This removes $i$ complexity for $i$-bit word.

To sum up, given input differences, if $i$-th bit of input and output differences are one of these form - $(0,0,0)$ or $(1,1,1)$, we can expect only one bit possible for the next bit. Total complexity decreases by $1/4\times1/2=1/8$((Probability of $(0,0,0)$ or $(1,1,1)$) $\times$ (Possible next output bit)).

### 4.2 Bitwise $xdp$ computation

For $r$-th round, to add one more round to differential trail, $\sum_{k=1}^{w}k$ computation complexity is required since we need to calculate $xdp$ every time one more bit is added. For total $r$ round, $r\times w(w+1)/2$ computation complexity is needed.

Look at $i$-bit of two input differences and output difference, $\alpha[i:1]$, $\beta[i:1]$, $\gamma[i:1]$. Then we can expect possible next bit $x_\alpha$, $x_\beta$, and $x_\gamma$ by 4.1 section. If we check $i$-th bit of differences, $\alpha_i$, $\beta_i$, and $\gamma_i$, we can conclude $xdp(\alpha[i+1:1], \beta[i+1:1] \rightarrow \gamma[i+1:1])$ is equal to $xdp(\alpha[i:1], \beta[i:1] \rightarrow \gamma[i:1])$ multiplied by 1 or 1/2. This procedure is same as find out whether for $j$ such that $1 \leqq j < i$, the equation $\alpha_j=\beta_j=\gamma_j$ is true or not. Let total number of $j$ that makes the equation false $p$. Then the $xdp$ is $2^{-p}$. In other words, to compute $xdp(\alpha[i+1:1], \beta[i+1:1] \rightarrow \gamma[i+1:1])$, first see if the equation $\alpha_i=\beta_i=\gamma_i$ is true or not. If that is false, then we can compute $xdp(\alpha[i+1:1], \beta[i+1:1] \rightarrow \gamma[i+1:1]) = xdp(\alpha[i:1], \beta[i:1] \rightarrow \gamma[i:1])\times1/2$. In conclusion, to add one more round to the trail, we can calculate it with $\sum_{k=1}^{w}1$ that is bitwise calculation, not $\sum_{k=1}^{w}k$.

Thus, for total $r$ round and $w$-bit word, total complexity is $r\times w$.

Table 2 shows $i$-th bit and corresponding $(i-1)$-th bit of input and output differences possible and probability which will be multiplied

Table 2. $i^{th}$ bit and corresponding $(i-1)^{th}$ bit, and probability which will be multiplied

| $(i-1)^{th}$ bit | $i^{th}$ bit | Prob. |
|---|---|---|
| (0, 0, 0) | (0, 0, 0) | 1 |
| | (0, 1, 1) | |
| | (1, 0, 1) | |
| | (1, 1, 0) | |
| (0, 0, 1) | All possible | 1/2 |
| (0, 1, 0) | | |
| (0, 1, 1) | | |
| (1, 0, 0) | | |
| (1, 0, 1) | | |
| (1, 1, 0) | | |
| (1, 1, 1) | (0, 0, 1) | 1 |
| | (0, 1, 0) | |
| | (1, 0, 0) | |
| | (1, 1, 1) | |

By Table 2, if $i-1$-th bit of input and output differences are same, the probability remains still, and $i$-th bit is determined as four cases. If not, all cases are possible and the probability decreases by 1/2. Applying this method lets not to search for the zero probability trails, and lets to compute bitwise.

### 4.3 Results

Biryukov used HPC cluster to reduce

the computing time for more than 7 rounds. However we failed to make same system environment, so we compared the time in less than 8 rounds of SPECK32, and 7 rounds of SPECK48.

Table 3. Time comparison of Algorithm 2 and 3 (time: Intel CoreTM E5-2637 CPU 3.50GHz time': Intel CoreTM i7-2600 CPU 3.40GHz)

| $r$ | SPECK32 | | | SPECK48 | | |
|---|---|---|---|---|---|---|
| | Prob. | time | time' | Prob. | time | time' |
| 1 | 0 | 0sec | 0sec | 0 | 0sec | 0sec |
| 2 | -1 | 0sec | 0sec | -1 | 0sec | 0sec |
| 3 | -3 | 0sec | 0sec | -3 | 0sec | 0sec |
| 4 | -5 | 0sec | 0sec | -6 | 0sec | 0sec |
| 5 | -9 | 0sec | 0sec | -10 | 1sec | 1sec |
| 6 | -13 | 1sec | 1sec | -14 | 3sec | 3sec |
| 7 | -18 | 1min | 26sec | -19 | 1min | 33sec |
| 8 | -24 | 34min | 11min | - | | |

Prob. = $\log_2(p)$

Table 3 shows that despite poor computer specifications, the results are two to three times better. Thus if we use same computer specifications or HPC cluster, it is obvious that we can get better results.

## V. Conclusion

In this paper, we proposed more efficient algorithm than algorithm that Biryukov suggested. However for more than 8 rounds, time took too long to get better trails in SPECK48. So we suggested some ways to reduce time by 1/2 to 1/3 (Algorithm 3). If we apply this method, we are sure that we can get longer differential trail.

Algorithm 3 can be applied to many different ARX cipher.

## References

[1] Biham, Eli, and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems," *Journal of CRYPTOLOGY* vol. 4, no. 1, pp. 3-72, Jan. 1991.

[2] Biryukov, Alex, Vesselin Velichkov, and Yann Le Corre. "Automatic search for the best trails in arx: Application to block cipher speck," *Fast Software Encryption - FSE*. pp. 268-288, Mar. 2016.

[3] Beaulieu, Ray, et al. "The SIMON and SPECK lightweight block ciphers," *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015

[4] Lipmaa, Helger, and Shiho Moriai. "Efficient algorithms for computing differential properties of addition," *International Workshop on Fast Software Encryption*. Springer Berlin Heidelberg, 2001

[5] Fu, Kai, et al. "MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck," *Fast Software Encryption - FSE*. pp. 289-310, Mar. 2016.

[6] Hong, Deukjo, et al. "HIGHT: A new block cipher suitable for low-resource device," *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer Berlin Heidelberg, 2006

[7] Mouha, Nicky, et al. "Chaskey: an efficient MAC algorithm for 32-bit microcontrollers," *International Workshop on Selected Areas in Cryptography*. Springer International Publishing, 2014

## VI. Appendix A

**Algorithm 2. [2] Search for the Best Differential Trail in ARX (Application to SPECK)**

**Input** - $n$: number of rounds
$w$: word size in bits
$r_1$, $r_2$: right and left rotation constant
$r$: current round ($n \geqq r \geqq 1$)
$i$: current bit position ($w > i \geqq 0$)
$B = (B_1, B_2, \ldots, B_{n-1})$: probabilities of the best trails for rounds 1, 2, ..., (n-1) (global)
$\overline{B_n}$: underestimate of the best probability for n rounds: $\overline{B_n} \leqq B_n$
$T = (T_1, T_2, \ldots, T_{r-1})$, $T_i = (\alpha_i, \beta_i, \gamma_i, p_i)$, $p_i = xdp(\alpha_i, \beta_i \rightarrow \gamma_i)$, $1 \leqq i < r$
$(\alpha_r, \beta_r, \gamma_r)$: input and output differences to the modular addition at round $r$
$\widetilde{p_r}$: probability of the partial differential $(\alpha_r[0:i], \beta_r[0:i] \rightarrow \gamma_r[0:i])$
**Output** - $B_n, T$: the best probability for $n$ rounds and corresponding trail

1. // Initialization : $r \leftarrow 1, i \leftarrow 0, \alpha_r \leftarrow 0, \beta_r \leftarrow 0, \gamma_r \leftarrow 0$
2. procedure best_diff_search$(r, i, \alpha_r, \beta_r, \gamma_r)$ do
3. //Fisrt round
4. if $(r = 1) \wedge (r \neq 1)$ then
5. if $i = w$ then
6. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p_r)$; add $T_r$ to $T$
7. $i \leftarrow 0$; $\alpha_{r+1} \leftarrow (\gamma_r \gg r_1)$; $\beta_{r+1} \leftarrow \gamma_r \oplus (\beta_r \ll r_2)$; $\gamma_{r+1} \leftarrow 0$;
8. call best_diff_search$(r+1, i, \alpha_{r+1}, \beta_{r+1}, \gamma_{r+1})$
9. else
10. for $j_\alpha, j_\beta, j_\gamma \in \{0, 1\}$ do
11. $\alpha_r[i] \leftarrow j_\alpha$; $\beta_r[i] \leftarrow j_\beta$; $\gamma_r[i] \leftarrow j_\gamma$; $\widetilde{p_r} \leftarrow xdp^+(\alpha_r[0:i], \beta_r[0:i] \rightarrow \gamma_r[0:i])$;
12. if $(\widetilde{p_r} \times B_{n-1}) \geqq \overline{B_n}$ then
13. cal best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r)$
14. //Intermediate rounds
15. if $(r > 1) \wedge (r \neq 1)$ then
16. if $i = w$ then
17. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p_r)$; add $T_r$ to $T$
18. $i \leftarrow 0$; $\alpha_{r+1} \leftarrow (\gamma_r \gg r_1)$; $\beta_{r+1} \leftarrow \gamma_r \oplus (\beta_r \ll r_2)$; $\gamma_{r+1} \leftarrow 0$;
19. call best_diff_search$(r+1, i, \alpha_{r+1}, \beta_{r+1}, \gamma_{r+1})$
20. else
21. for $j_\gamma \in \{0, 1\}$ do
22. $\gamma_r[i] \leftarrow j_\gamma$; $\widetilde{p_r} \leftarrow xdp^+(\alpha_r[0:i], \beta_r[0:i] \rightarrow \gamma_r[0:i])$;
23. if $(\widetilde{p_r} \times B_{n-1}) \geqq \overline{B_n}$ then
24. call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r)$
25. //Last round
26. if $(r = n)$ then
27. if $i = w$ then
28. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p_r)$; add $T_r$ to $T$
29. if $(p_1 \times p_2 \ldots \times p_n) \geqq \overline{B_n}$ then
30. //Update bound and return to upper round

31. $\overline{B_n} \leftarrow (p_1 \times p_2 \times ... \times p_n)$
32. else
33. for $j_\gamma \in \{0,1\}$ do
34. $\gamma_r[i] \leftarrow j_\gamma$; $\widetilde{p_r} \leftarrow xdp^+(\alpha_r[0:i], \beta_r[0:i] \rightarrow \gamma_r[0:i])$;
35. if $(\widetilde{p_r} \times B_{n-1}) \geqq \overline{B_n}$ then
36. cal best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r)$
37. return

## VII. Appendix B

Algorithm 3 is an improved version of Algorithm 2 as explained in Section 4. Different parts are highlighted.

**Algorithm 3. Advanced Search for the Best Differential Trail in ARX (Application to SPECK)**

**Input** - $n$: number of rounds
$w$: word size in bits
$r_1$, $r_2$: right and left rotation constant
$r$: current round ($n \geqq r \geqq 1$)
$i$: current bit position ($w > i \geqq 0$)
$B = (B_1, B_2, ..., B_{n-1})$: probabilities of the best trails for rounds 1, 2, ..., (n-1) (global)
$\overline{B_n}$: underestimate of the best probability for n rounds: $\overline{B_n} \leqq B_n$
$T = (T_1, T_2, ..., T_{r-1})$, $T_i = (\alpha_i, \beta_i, \gamma_i, p)$, $1 \leqq i < r$
$(\alpha_r, \beta_r, \gamma_r)$: input and output differences to the modular addition at round $r$
**Output** - $B_n$, $T$: the best probability for $n$ rounds and corresponding trail

1. // Initialization : $r \leftarrow 1, i \leftarrow 0, \alpha_r \leftarrow 0, \beta_r \leftarrow 0, \gamma_r \leftarrow 0$, $p \leftarrow 1$
2. procedure best_diff_search$(r, i, \alpha_r, \beta_r, \gamma_r, p)$ do
3. //Fisrt round
4. if $(r = 1) \wedge (r \neq 1)$ then
5. if $i = w$ then
6. $p_r \leftarrow xdp^+(\alpha_r, \beta_r \rightarrow \gamma_r)$; $T_r \leftarrow (\alpha_r, \beta_r, \gamma_r, p)$; add $T_r$ to $T$
7. $i \leftarrow 0$; $\alpha_{r+1} \leftarrow (\gamma_r \ggg r_1)$; $\beta_{r+1} \leftarrow \gamma_r \oplus (\beta_r \lll r_2)$; $\gamma_{r+1} \leftarrow 0$;
8. call best_diff_search$(r+1, i, \alpha_{r+1}, \beta_{r+1}, \gamma_{r+1}, p)$
9. else if $i = 0$
10. for $j_\alpha, j_\beta \in \{0,1\}$
11. $\alpha_r[i] \leftarrow j_\alpha$; $\beta_r[i] \leftarrow j_\beta$; $\gamma_r[i] \leftarrow j_\alpha \oplus j_r$
12. else
13. $temp = (\alpha_r[i-1] = \beta_r[i=1] = \gamma_r[i-1])\ ?\ \alpha_{r[i-1]} : 2$;
14. if $temp = 2$
15. if $(p \times B_{n-1}) \geqq \overline{B_n}$ then
16. for $j_\alpha, j_\beta, j_\gamma \in \{0,1\}$ do
17. $\alpha_r[i] \leftarrow j_\alpha$; $\beta_r[i] \leftarrow j_\beta$; $\gamma_r[i] \leftarrow j_\gamma$; call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r, p \times 1/2)$
18. else
19. for $j_\alpha$, $j_\beta$ $\in \{0,1\}$
20. $\alpha_r[i] \leftarrow j_\alpha$; $\beta_r[i] \leftarrow j_\beta$; $\gamma_r[i] \leftarrow j_\alpha \oplus j_\beta \oplus temp$; call best_diff_search$(r, i+1, \alpha_r, \beta_r, \gamma_r, p)$

```
21. //Intermediate rounds
22.   if (r > 1) ∧ (r ≠ 1) then
23.       if i = w then
24.         p_r ← xdp+(α_r, β_r → γ_r); T_r ← (α_r, β_r, γ_r, p); add T_r to T
25.         i ← 0; α_{r+1} ← (γ_r ⋙ r_1); β_{r+1} ← γ_r ⊕ (β_r ⋘ r_2); γ_{r+1} ← 0;
26.         call best_diff_search(r+1, i, α_{r+1}, β_{r+1}, γ_{r+1}, p)
27.       else if i = 0
28.         γ_r[0] = α_r[0] ⊕ β_r[0]; call best_diff_search(r, i+1, α_r, β_r, γ_r, p)
29.       else
30.         temp = (α_r[i-1] = β_r[i=1] = γ_r[i-1])? α_{r[i-1]} : 2;
31.         if temp = 2
32.           if (p × 1/2 × B_{n-r}) ≧ B̄_n then
33.             for j_γ ∈ {0,1} do
34.               γ_r[i] = j_γ; call best_diff_saerch(r, i+1, α_r, β_r, γ_r, p × 1/2)
35.         else
36.           γ_r[i] = α_r[i] ⊕ β_r[i] ⊕ temp; call best_diff_search(r, i+1, α_r, β_r, γ_r, p)
37. //Last round
38.   if (r = n) then
39.       if i = w then
40.         p_r ← xdp+(α_r, β_r → γ_r); T_r ← (α_r, β_r, γ_r, p); add T_r to T
41.         if p ≧ B̄_n then
42.           //Update bound and return to upper round
43.           B̄_n ← p
44.         else if i = 0
45.           γ_r[0] = α_r[0] ⊕ β_r[0]; call best_diff_search(r, i+1, α_r, β_r, γ_r, p)
46.         else
47.           temp = (α_r[i-1] = β_r[i=1] = γ_r[i-1])? α_{r[i-1]} : 2;
48.           if temp = 2
49.             if (p × 1/2 × B_{n-r}) ≧ B̄_n then
50.               for j_γ ∈ {0,1} do
51.                 γ_r[i] = j_γ; call best_diff_saerch(r, i+1, α_r, β_r, γ_r, p × 1/2)
52.           else
53.             γ_r[i] = α_r[i] ⊕ β_r[i] ⊕ temp; call best_diff_search(r, i+1, α_r, β_r, γ_r, p)
54. return
```