Hindawi
Security and Communication Networks
Volume 2019, Article ID 7862738, 12 pages
https://doi.org/10.1155/2019/7862738

Research Article

Cryptanalysis of the Lightweight Block Cipher BORON

Huicong Liang [1,2] **and Meiqin Wang** [1,2]

[1]*School of Cyber Science and Technology, Shandong University, Qingdao, China*
[2]*Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao, China*

Correspondence should be addressed to Meiqin Wang; mqwang@sdu.edu.cn

Received 15 March 2019; Revised 1 November 2019; Accepted 25 November 2019; Published 18 December 2019

Academic Editor: Vincenzo Conti

This paper provides security evaluations of a lightweight block cipher called BORON proposed by Bansod et al. There is no third-party cryptanalysis towards BORON. Designers only provided coarse and simple security analysis. To fill this gap, security bounds of BORON against differential and linear cryptanalysis are presented in this paper. By automatic models based on the SMT solver STP, we search for differential and linear trails with the minimal number of active S-boxes and trails with optimal probability and bias. Then, we present key-recovery attacks towards round-reduced BORON. This paper is the first third-party cryptanalysis towards BORON.

1. Introduction

Lightweight cryptography is one of the most actively discussed topics in the current cryptographic community. For the last decade, a great number of lightweight block ciphers have been proposed, such as PRESENT [1], CLEFIA [2], PRINCE [3], SIMON [4], SPECK [4], SKINNY [5], GIFT [6], QARMA [7], and so on, which draw plenty of researchers' attention.

Meanwhile, there are some unknown lightweight block ciphers. BORON [8] is one of them. BORON, a substitution-permutation network cipher, supports 64-bit messages and 80/128-bit keys, which has a total of 25 rounds. The substitution layer is made up of 16 same 4-bit S-Boxes, and the permutation layer includes nibble shift, bit rotation, and block XOR operation.

Except for simple security analysis illustrated by designers, there is no third-party security evaluation in the open literature. To fill this gap, differential [9] and linear cryptanalysis [10] towards BORON are presented in this paper. Differential and linear cryptanalysis are one of the most fundamental and powerful techniques for security evaluations of cryptographic primitives.

Differential cryptanalysis discusses how differences propagate through the objective cryptographic primitives. If input differences and output differences occur in some regular patterns, these could be used to build distinguishers or even to recover secret keys.

Linear cryptanalysis focuses on the linear equation between plaintexts, ciphertexts, and keys. If this linear equation happens with a high probability, the distinguishing attack or key-recovery attack could be presented. Therefore, finding good distinguishers is the first step to evaluate security against differential and linear cryptanalysis.

There are many methods to find differential and linear trails, such as Matsui's algorithm, SAT/SMT and MILP, which are widely used to analyse the security of cryptographic primitives [11–15]. In this paper, we utilize the SMT solver STP (http://stp.github.io) to search for good trails, which is suitable for dealing with bit vectors.

1.1. Contributions. This paper analyses the security of BORON against differential and linear cryptanalysis. Security bounds are depicted in Table 1.

The minimal number of active S-boxes of differential and linear trails could describe approximate security bounds resistance to differential and linear cryptanalysis. We find that the minimal number of active S-boxes proposed by

Table 1: Security bounds of BORON.

Attack type	Distinguisher	Attack	Time	Data	Memory
Differential (80-bit key)	8-round	9-round	2^{56}	2^{63}	2^{24}
Linear (128-bit key)	9-round	11-round	2^{123}	2^{63}	2^{42}
Related-key differential	10-round	—	—	—	—
Impossible differential	7-round	—	—	—	—

Table 2: Minimal number of active S-boxes.

Round	Differential trails	Linear trails
3	6	6
4	9	9
5	12	11
6	16	14
7	21	17
8	24	19
9	27	22
10	31	25

designers is incorrect and intact. Then, differential and linear trails with the accurate minimal number of active S-boxes are found. Our results are listed in Table 2.

We also search for (related-key) differential and linear trails with the optimal probability and bias. Details about the optimal probability and bias are illustrated in Table 3. Specific trails with the optimal probability and bias are listed in Tables4–6. We figure out that there is no effective 9-round differential trail and 10-round linear trail. We also try to find impossible differential trails of BORON, and 7-round BORON impossible differential trails are obtained.

Utilizing the 8-round differential trail with the optimal probability of 2^{-62}, we depict a key-recovery attack towards 9-round BORON with the 80-bit key schedule, whose time complexity is 2^{56}, data complexity is 2^{63}, and memory complexity is 2^{24}. By the 9-round linear trail with the optimal bias of 2^{-30}, a key-recovery attack towards 11-round BORON with the 128-bit key schedule is described, whose time complexity is 2^{123}, data complexity is 2^{63}, and memory complexity is 2^{42}.

And, in this paper, we assume that BORON is the Markov cipher.

1.2. Outline. This paper is organized as follows. Section 2 introduces the description of BORON. In Section 3, automatic models for finding trials are illustrated. Key-recovery attacks towards BORON are presented in Section 4. Section 5 concludes this paper.

2. Description of BORON

2.1. Encryption Algorithm. BORON is a 25-round SPN block cipher, whose message length is 64-bit and the key length is 80-bit or 128-bit. The round function of BORON is made up of Substitution Layer and Permutation Layer. The Substitution Layer consists of 16 same 4-bit S-boxes in parallel. The S-box is described in Table 7. The Permutation Layer includes three parts, nibble shift, bit rotation, and block XOR operation. And, the round function is described in Figure 1. For more details, refer [8].

In Figure 1, we regard 4 bits as a nibble and 16 bits as a block. Note that nibble shift is the permutation between nibbles, while bit rotation and block XOR operation are ones in term of blocks.

2.2. Key Schedule. The key schedule of BORON is inspired by the one of PRESENT [1]. There are two versions, 80-bit and 128-bit. In total, 26 subkeys are generated by the key schedule in the whole procedure of encryption. Each subkey has 64 bits.

Table 3: Optimal probability and bias.

Round	Differential trails	Linear trails	Related-key differential trails
3	2^{-12}	2^{-7}	2^{-2}
4	2^{-19}	2^{-11}	2^{-2}
5	2^{-28}	2^{-15}	2^{-6}
6	2^{-38}	2^{-19}	2^{-15}
7	2^{-51}	2^{-23}	2^{-27}
8	2^{-62}	2^{-27}	2^{-40}
9	*	2^{-30}	2^{-48}
10	*	*	2^{-55}

*Represents that there is no effective trail.

For the 80-bit version, the master key could be stored in a key register and denoted as $K = k_{79}k_{78}\ldots k_0$. First, extract the least significant 64 bits of the master key as the first subkey, that is, the whitening key $K_0 = k_{63}k_{62}\ldots k_0$. In the jth round $(j = 0, 1, \ldots, 24)$, after extracting the least significant 64 bits, the key register is updated as follows:

(1) $K \lll 13$

(2) $k_3\|k_2\|k_1\|k_0 \longleftarrow S[k_3\|k_2\|k_1\|k_0]$

(3) $k_{63}\|k_{62}\|k_{61}\|k_{60}\|k_{59} \longleftarrow (k_{63}\|k_{62}\|k_{61}\|k_{60}\|k_{59}) \oplus j$

For the 128-bit version, the master key could be stored in a key register and denoted as $K = k_{127}k_{126}\ldots k_0$. Extract the least significant 64 bits of the master key as the first subkey, that is, the whitening key $K_0 = k_{63}k_{62}\ldots k_0$. In the jth round $(j = 0, 1, \ldots, 24)$, after extracting the least significant 64 bits, the key register is updated as follows:

(1) $K \lll 13$

(2) $k_3\|k_2\|k_1\|k_0 \longleftarrow S[k_3\|k_2\|k_1\|k_0]$

(3) $k_7\|k_6\|k_5\|k_4 \longleftarrow S[k_7\|k_6\|k_5\|k_4]$

(4) $k_{63}\|k_{62}\|k_{61}\|k_{60}\|k_{59} \longleftarrow (k_{63}\|k_{62}\|k_{61}\|k_{60}\|k_{59}) \oplus j$

3. Automatic Models for Searching Trails

In this paper, we use STP [16], an SMT solver, to search for good differential and linear trails. STP uses CVC and SMTLIB2 languages to encode constraints and then invokes an SAT solver to check for the satisfiability of these constraints. In this paper, the CVC language is used to encode difference and linear mask propagations of BORON.

Table 4: Differential trails with the optimal probability.

Round	Input difference of S-box	Output difference of S-box
3-round differential trail with probability 2^{-12}		
0	0x0000000800000200	0x0000000C00000300
1	0x0000000000000006	0x0000000000000008
2	0x1000100000001000	0xA000A0000000A000
3	0x1141514040014141	—
4-round differential trail with probability 2^{-19}		
0	0xA0000009000F0000	0x4000000100080000
1	0x0000800000000000	0x0000C00000000000
2	0x6000600000000000	0x2000800000000000
3	0x0000400040004000	0x0000900090007000
4	0x48E048E0090009E0	—
5-round differential trail with probability 2^{-28}		
0	0x0000002009000F00	0x0000003001000800
1	0x0008000800100000	0x0003000C00600000
2	0x0000000600000000	0x0000000E00000000
3	0x0007000700000000	0x0001000400000000
4	0x0000000200020002	0x00000003000B0005
5	0x8A018A01B000BA00	—
6-round differential trail with probability 2^{-38}		
0	0x09000A0030080000	0x01000400200C0000
1	0x00000200C000C000	0x0000030010008000
2	0x0080008001000000	0x003000C006000000
3	0x0000006000000000	0x000000C000000000
4	0x0060006000000000	0x0020008000000000
5	0x0000004000400040	0x0000009000700090
6	0x2049204900072006	—
7-round differential trail with probability 2^{-51}		
0	0xB000A00000230000	0x1000400000520000
1	0x0000200000050005	0x0000300000010008
2	0x0800080010000000	0x03000C0060000000
3	0x0000060000000000	0x00000A0000000000
4	0x0500050000000000	0x03000C0000000000
5	0x0000060006000600	0x0000080008000800
6	0x0410041000800090	0x07A007A000F00010
7	0x2C9023D00F4F2F4F	—
8-round differential trail with probability 2^{-62}		
0	0x0000080000100000	0x00000C0000F00000
1	0x06000600000F000F	0x08000E00000E0008
2	0x07001700F000E000	0x0400F10080004000
3	0x7000780000000080	0x60001C0000000060
4	0x0E00CE00C0000000	0x06001800C0000000
5	0x00000C0000000000	0x0000010000000000
6	0x0080008000000000	0x003000C000000000
7	0x0000006000600060	0x0000008000800080
8	0x0041004100080009	—

Table 5: Linear trails with the optimal bias.

Round	Input mask of S-box	Output mask of S-box
3-round linear trail with bias 2^{-7}		
0	0x0D0000000006000B	0x0800000000010008
1	0x0000000000001000	0x0000000000003000
2	0x0000000000600060	0x0000000000100060
4-round linear trail with bias 2^{-11}		
0	0xD0000000006000B0	0x8000000000100080
1	0x0000000000000001	0x0000000000000007
2	0x000000000E000E00	0x0000000001000800
3	0x0010001000000010	0x0030003000000070
5-round linear trail with bias 2^{-15}		
0	0x0000000D00F00000	0x0000000200100000
1	0x0001000000000001	0x0007000000000003
2	0x000E000E06000600	0x0008000401000100
3	0x0000000200100000	0x0000000E00700000
4	0x0007000000000007	0x000C00000000000C
6-round linear trail with bias 2^{-19}		
0	0x0000009006000000	0x0000002001000000
1	0x0010000000000010	0x0070000000000030
2	0x00E000E060006000	0x0090004010001000
3	0x0020000001000000	0x0040000008000000
4	0x0000000000800000	0x0000000000500000
5	0x0005000500050000	0x0003000700030000
7-round linear trail with bias 2^{-23}		
0	0xD0000D00000D000D	0x4000080000080002
1	0x0000040080000000	0x00000A0050000000
2	0x0500000000000500	0x0300000000000300
3	0x0600060000060006	0x0B000C0000010003
4	0x0600000010000000	0x06000000C0000000
5	0x000000000C000000	0x0000000005000000
6	0x0050005000500000	0x0070007000300000
8-round linear trail with bias 2^{-27}		
0	0x00000D00F0000000	0x0000020010000000
1	0x0100000000000100	0x0700000000000300
2	0x0E000E0000060006	0x0800040000010001
3	0x0000020010000000	0x00000E0070000000
4	0x0700000000000700	0x0100000000000300
5	0x0200020000060006	0x0B000C0000010003
6	0x0600000010000000	0x06000000C0000000
7	0x000000000C000000	0x0000000005000000
9-round linear trail with bias 2^{-30}		
0	0x0000000F000D0000	0x0000000100080000
1	0x8000000000008000	0x5000000000007000
2	0xA000A00000E000E0	0x8000400000100010
3	0x0000200000010000	0x0000E00000070000
4	0x7000000000007000	0x1000000000003000
5	0x2000200000600060	0xB000C00000100030
6	0x6000000000010000	0x600000000000C0000
7	0x00000000C0000000	0x0000000070000000
8	0x0700070007000000	0x010001000C000000

Inspired by models in [11–15, 17–20], we present automatic models for finding differential and linear trails of BORON. All intermediate states of BORON are represented by variables which are used to build constraints. Searching for good trails is transformed into checking for the satisfiability of constraints. For different searching problems, there are different constraints. We show constraints for all operations used in BORON as follows.

3.1. Finding Differential Trails. First, we give some notations about variables. We count from the right side and zero.

(i) before_sbox_vaule1_*i*_*j*: input value of the *j*th S-box in the *i*th round

(ii) after_sbox_vaule1_*i*_*j*: output value of the *j*th S-box in the *i*th round whose input value is before_sbox_vaule1_*i*_*j*

(iii) before_sbox_vaule2_*i*_*j*: input value of the *j*th S-box in the *i*th round

Table 6: Related-key differential trails with the optimal probability.

Round	Subkey difference	Input difference of S-box	Output difference of S-box
4-round related-key differential trail with probability 2^{-2}			
0	0x0020000000000000	0x0000000000000000	0x0000000000000000
1	0x0000000000000000	0x0000000000000000	0x0000000000000000
2	0x0000000000000000	0x0000000000000000	0x0000000000000000
3	0x0000000000001000	0x0000000000001000	0x000000000000F000
5-round related-key differential trail with probability 2^{-6}			
0	0x0000004000000000	0x000A000000F00000	0x0004000000800000
1	0x0008000000000000	0x0000000000000000	0x0000000000000000
2	0x0000000000000000	0x0000000000000000	0x0000000000000000
3	0x0000000000000000	0x0000000000000000	0x0000000000000000
4	0x0000000000000400	0x0000000000000400	0x0000000000000900
6-round related-key differential trail with probability 2^{-15}			
0	0x0000004000000000	0x000A000000600000	0x0004000000800000
1	0x0008000000000000	0x0000000000000000	0x0000000000000000
2	0x0000000000000000	0x0000000000000000	0x0000000000000000
3	0x0000000000000000	0x0000000000000000	0x0000000000000000
4	0x0000000000000400	0x0000000000000400	0x0000000000000500
5	0x0000000000800000	0x000A000A0080000A	0x0004000F00C00004
7-round related-key differential trail with probability 2^{-27}			
0	0x0000000002000000	0x003200A003102000	0x0025004002A03000
1	0x0000004000000000	0x000A000000600000	0x0004000000800000
2	0x0008000000000000	0x0000000000000000	0x0000000000000000
3	0x0000000000000000	0x0000000000000000	0x0000000000000000
4	0x0000000000000000	0x0000000000000000	0x0000000000000000
5	0x0000000000000400	0x0000000000000400	0x0000000000000500
6	0x0000000000800000	0x000A000A0080000A	0x000F000400C0000F
8-round related-key differential trail with probability 2^{-40}			
0	0x0000000002000000	0x003200A003102000	0x0025004002A03000
1	0x0000004000000000	0x000A000000600000	0x0004000000800000
2	0x0008000000000000	0x0000000000000000	0x0000000000000000
3	0x0000000000000000	0x0000000000000000	0x0000000000000000
4	0x0000000000000000	0x0000000000000000	0x0000000000000000
5	0x0000000000000400	0x0000000000000400	0x0000000000000700
6	0x0000000000800000	0x000E000E0080000E	0x0003000D00600008
7	0x0000001000000000	0x9000901600001000	0x100090F80000F000
9-round related-key differential trail with probability 2^{-48}			
0	0x0000010000000000	0x00B0000003000000	0x0010000002000000
1	0x0020000000000000	0x0000000000000000	0x0000000000000000
2	0x0000000000000000	0x0000000000000000	0x0000000000000000
3	0x0000000000000000	0x0000000000000000	0x0000000000000000
4	0x0000000000001000	0x0000000000001000	0x0000000000006000
5	0x0000000002000000	0x00C000C0020000C0	0x0050008003000070
6	0x0000004000000000	0xE0E0E0000090E090	0x8030400000103010
7	0x0008000000000000	0x0009006000602000	0x000400C000803000
8	0x0000000000000000	0x0008000000000060	0x000F0000000000E0
10-round related-key differential trail with probability 2^{-55}			
0	0x0000010000000000	0x00B0000003000000	0x0010000002000000
1	0x0020000000000000	0x0000000000000000	0x0000000000000000
2	0x0000000000000000	0x0000000000000000	0x0000000000000000
3	0x0000000000000000	0x0000000000000000	0x0000000000000000
4	0x0000000000001000	0x0000000000001000	0x0000000000006000
5	0x0000000002000000	0x00C000C0020000C0	0x0050008003000070
6	0x0000004000000000	0xE0E0E0000090E090	0x8070400000107010
7	0x0008000000000000	0x000900E000E02000	0x000400D000803000
8	0x0000000000000000	0x0000000800000060	0x00000003000000C0
9	0x0000000000000000	0x0000000000008001	0x000000000000C00F

(iv) after_sbox_vaule2_i_j: output value of the jth S-box in the ith round whose input value is before_sbox_vaule2_i_j

Table 7: S-box S.

X	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S[x]$	e	4	b	1	7	9	c	a	d	2	0	f	8	5	3	6

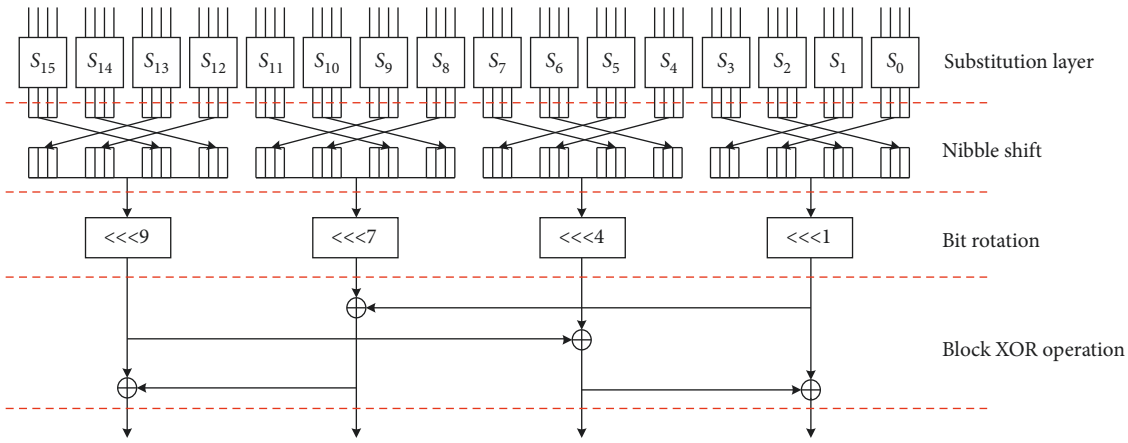

Figure 1: Round function.

(v) before_sbox_difference_i_j = before_sbox_value1_i_j ⊕ before_sbox_value2_i_j

(vi) after_sbox_difference_i_j = after_sbox_value1_i_j ⊕ after_sbox_value2_i_j

(vii) before_rotation_difference_i_j: difference of the jth block before bit rotation of the ith round

(viii) after_rotation_difference_i_j: difference of the jth block after bit rotation of the ith round

(ix) after_blockxor_difference_i_j: difference of the jth block after block XOR operation of the ith round

(x) flag_i_j: represents whether the jth S-box in the ith round is active or not

(xi) total_num: the total number of active S-boxes in the trail

(xii) probability_i_j: probability parameter of the jth S-box in the ith round

(xiii) total_probability: probability parameter of the whole differential trail

3.1.1. Permutation Layer. The Permutation Layer is made up of three linear operations. Constraints about these linear operations are listed in Algorithm 1, 2, and 3, respectively.

3.1.2. Substitution Layer. Constraints built in Algorithm 4 describe difference propagations through S-boxes, which include initializing variables by values of S-boxes and showing relationships among variables.

Other constraints should be added in Algorithm 4, when searching for differential trails with the minimal number of active S-boxes. These constraints of searching for the minimal number of active S-boxes are illustrated in Algorithm 5. The variable flag_i_j is used to explain whether an S-box is active or not. When the jth S-box in the ith round is active, flag_i_j = 1; otherwise, flag_i_j = 0.

We need to check whether total_num = n is satisfied or not under constraints from Algorithm 4 and 5, where n is the expected number of active S-boxes in the whole differential trail, and total_num = n is regarded as the objective constraints.

We first search for 3-round trails with the minimal number of active S-boxes, and set $n = 1$. Put all constraints including Algorithm 1–5 into STP and check the satisfaction of these objective constraints. If STP returns invalid, gradually adjust the value of n by adding 1 till STP returns a trail. When searching for r-round trails with the minimal number of active S-boxes, we set the value of n equal to the minimal number of active S-boxes of $(r-1)$-round trails. Repeat the above steps.

When searching for the optimal differential trail, we need to consider probabilities of difference propagations through S-boxes. Probabilities of S-boxes could be described by the difference distribution table (DDT).

For any input difference x and output difference y of the S-box, the probability of the S-box, named $p\_\mathrm{sbox}(x, y)$, is equal to the corresponding value in the DDT divided by 16, which is

$$p\_\mathrm{sbox}(x, y) = \frac{\mathrm{value\_in\_DDT}(x, y)}{16}. \tag{1}$$

We can write that $p\_\mathrm{sbox}(x, y) = 2^{-\mathrm{probability}(x,y)}$. In our model, we use the parameter probability(x, y) to present the probability of the S-box and

$$\mathrm{probability}(x, y) = -\log_2 \frac{\mathrm{value\_in\_DDT}(x, y)}{16}. \tag{2}$$

Values of probability(x, y) for all x and y are stored in a table called Diff_dist.

Denote the probability of the whole differential trail as $2^{-\mathrm{total\_probability}}$ and

$$\mathrm{total\_probability} = \sum_{i=0,j=0}^{i=r,j=15} \mathrm{probability}\_i\_j. \tag{3}$$

The parameter total_probability is used to represent the probability of the whole trail in the constraints. Constraints of finding the optimal differential trail are presented in Algorithm 6.

Let that the value of total_probability starts from doubling the minimal number of active S-boxes because

```
r: the number of rounds
(1) for i ⟵ 0 to r do
(2)     before_rotation_difference_i_0 =
(3)     after_sbox_difference_i_1 ‖ after_sbox_difference_i_0 ‖
(4)     after_sbox_difference_i_3 ‖ after_sbox_difference_i_2
(5)
(6)     before_rotation_difference_i_1 =
(7)     after_sbox_difference_i_5 ‖ after_sbox_difference_i_4 ‖
(8)     after_sbox_difference_i_7 ‖ after_sbox_difference_i_6
(9)
(10)    before_rotation_difference_i_2 =
(11)    after_sbox_difference_i_9 ‖ after_sbox_difference_i_8 ‖
(12)    after_sbox_difference_i_11 ‖ after_sbox_difference_i_10
(13)
(14)    before_rotation_difference_i_3 =
(15)    after_sbox_difference_i_13 ‖ after_sbox_difference_i_12 ‖
(16)    after_sbox_difference_i_15 ‖ after_sbox_difference_i_14
(17) end for
```

Algorithm 1: Difference propagations through nibble shift.

```
r: the number of rounds
(1) for i ⟵ 0 to r do
(2)     after_rotation_difference_i_0 = before_rotation_difference_i_0 ⋘ 1
(3)     after_rotation_difference_i_1 = before_rotation_difference_i_1 ⋘ 4
(4)     after_rotation_difference_i_2 = before_rotation_difference_i_2 ⋘ 7
(5)     after_rotation_difference_i_3 = before_rotation_difference_i_3 ⋘ 9
(6) end for
```

Algorithm 2: Difference propagations through bit rotation.

```
r: the number of rounds
(1) for i ⟵ 0 to r do
(2) after_blockxor_difference_i_1 =
(3) after_rotation_difference_i_1 ⊕ after_rotation_difference_i_3
(4) after_blockxor_difference_i_2 =
(5) after_rotation_difference_i_0 ⊕ after_rotation_difference_i_2
(6) after_blockxor_difference_i_0 =
(7) after_rotation_difference_i_0 ⊕ after_blockxor_difference_i_1
(8) after_blockxor_difference_i_3 =
(9) after_rotation_difference_i_3 ⊕ after_blockxor_difference_i_2
(10) end for
```

Algorithm 3: Difference propagations through block XOR operation.

consider probabilities of all active S-boxes in the trail are optimal. Adjust the value of total_probability by adding 1 till obtaining the trail with the optimal probability.

3.2. Finding Linear Trails. Due to differences between difference propagations and linear mask propagations, we need to build constraints suitable for tracking linear masks in all operations of BORON. Notations are given as follows:

(i) before_sbox_mask_*i*_*j*: input mask of the *j*th S-box in the *i*th round

(ii) after_sbox_mask_*i*_*j*: output mask of the *j*th S-box in the *i*th round

(iii) branch_*i*_*j*_up, branch_*i*_*j*_down, branch_*i*_*j*_another: masks of the *j*th branch operation in the block XOR operation of the *i*-th round

(iv) xor_*i*_*j*_up, xor_*i*_*j*_down, xor_*i*_*j*_another: masks of the *j*th xor operation in the block XOR operation of the *i*th round

(v) bias_*i*_*j*: bias parameter of the *j*th S-box in the *i*th round

```
r: the number of rounds
S[x]: output of the S-box with the input x
 (1) for i ⟵ 1 to r do
 (2)   for j ⟵ 0 to 15 do
 (3)     for x ⟵ 0 to 15 do
 (4)       before_sbox_value1_i_j = x ⟹ after_sbox_value1_i_j = S[x]
 (5)       before_sbox_value2_i_j = x ⟹ after_sbox_value2_i_j = S[x]
 (6)     end for
 (7)   end for
 (8) end for
 (9)
(10) for i ⟵ 1 to r do
(11)   for j ⟵ 0 to 15 do
(12)     before_sbox_difference_i_j
(13)     = before_sbox_value1_i_j ⊕ before_sbox_value2_i_j
(14)     after_sbox_difference_i_j
(15)     = after_sbox_value1_i_j ⊕ after_sbox_value2_i_j
(16)   end for
(17) end for
```

Algorithm 4: Difference propagations through S-boxes.

```
r: the number of rounds
n: the expected number of active S-boxes in the whole trail
 (1) for i ⟵ 1 to r do
 (2)   for j ⟵ 0 to 15 do
 (3)     if before_sbox_difference_i_j = 0 AND after_sbox_difference_i_j = 0 then
 (4)       flag_i_j = 0
 (5)     else
 (6)       flag_i_j = 1
 (7)     end if
 (8)   end for
 (9) end for
(10)
(11) total_num = ∑_{i=0,j=0}^{i=r,j=15} flag_i_j
(12)
(13) total_num = n
```

Algorithm 5: Finding differential trails with the minimal number of active S-boxes.

(vi) total_bias: bias parameter of the whole linear trail

3.2.1. Permutation Layer. Situations that linear masks propagate nibble shift and bit rotation are the same as situations that differences do. So, we omit constraints about tracking linear masks through nibble shift and bit rotation. Refer Algorithm 1 and 2 for details.

It is little complicated that linear masks propagate through block XOR operation due to the property of linear masks. Detailed constraints are described in Algorithm 7. Figure 2 illustrates the variables which are used in tracking linear masks through block XOR operation.

3.2.2. Substitution Layer. Similar to finding differential trails, we also use flag_*i*_*j* to represent whether an S-box is active or not when finding linear trails with the minimal number of active S-boxes. And, the objective constraint is total_num = *n*, where *n* is the expected number of active S-boxes in the whole linear trail. The choice of *n* is the same as above. Detailed constraints are illustrated in Algorithm 8.

When searching for the optimal linear trail, we need to consider biases of S-boxes which could be described by the linear approximation table (LAT).

For any input mask α and output mask β of the S-box, the bias of the S-box, named $b\_\mathrm{sbox}(\alpha, \beta)$ is equal to the corresponding value in the LAT divided by 16, which is

$$b\_\mathrm{sbox}(\alpha, \beta) = \left| \frac{\mathrm{value\_in\_LAT}(\alpha, \beta)}{16} \right|. \tag{4}$$

We can write that $b\_\mathrm{sbox}(\alpha, \beta) = 2^{-\mathrm{bias}(\alpha,\beta)}$.

Use the parameter $\mathrm{bias}(\alpha, \beta)$ to present the bias of the S-box in the constraints and

```
r: the number of rounds
2^{-p}: the expected probability of the whole trail
(1) for x ⟵ 0 to 15 do
(2)   for y ⟵ 0 to 15 do
(3)     Diff_dist[x][y] = probability(x, y)
(4)   end for
(5) end for
(6)
(7) for i ⟵ 0 to r do
(8)   for j ⟵ 0 to 15 do
(9)     probability_i_j
(10)    = Diff_dist [before_sbox_difference_i_j] [after_sbox_difference_i_j]
(11)  end for
(12) end for
(13)
(14) total_probability = ∑_{i=0,j=0}^{i=r,j=15} probability_i_j
(15)
(16) total_probability = p
```

Algorithm 6: Finding differential trails with the optimal probability.

```
r: the number of rounds
input mask: branch_i_3_up‖xor_i_2_up‖xor_i_1_up‖branch_i_0_up
output mask: xor_i_3_down‖branch_i_2_down‖branch_i_1_down‖xor_i_0_down
(1) for i ⟵ 1 to r do
(2)   for j ⟵ 0 to 4 do
(3)     branch_i_j_up ⊕ branch_i_j_down ⊕ branch_i_j_another = 0
(4)   end for
(5) end for
(6)
(7) for i ⟵ 1 to r do
(8)   for j ⟵ 0 to 4 do
(9)     xor_i_j_up = xor_i_j_down = xor_i_j_another
(10)  end for
(11) end for
(12)
(13) for i ⟵ 1 to r do
(14)  branch_i_0_another = xor_i_2_another
(15)  branch_i_1_another = xor_i_0_another
(16)  branch_i_2_another = xor_i_3_another
(17)  branch_i_3_another = xor_i_1_another
(18) end for
(19)
(20) for i ⟵ 1 to r do
(21)  branch_i_0_down = xor_i_0_up
(22)  branch_i_1_up = xor_i_1_down
(23)  branch_i_2_up = xor_i_2_down
(24)  branch_i_3_down = xor_i_3_up
(25) end for
```

Algorithm 7: Linear mask propagations through block XOR operation.

$$\text{bias}(\alpha,\beta) = -\log_2\left|\frac{\text{value\_in\_LAT}(\alpha,\beta)}{16}\right|. \tag{5}$$

Values of $\text{bias}(\alpha,\beta)$ for all α and β are stored in a table named linear_dist.

Denote the bias of the whole linear trail as $2^{-\text{total\_bias}}$ and due to the Piling-up Lemma [10], we have

$$\text{total\_bias} = 1 - \text{total\_num} + \sum_{i=0,j=0}^{i=r,j=15} \text{bias\_}i\text{\_}j. \tag{6}$$

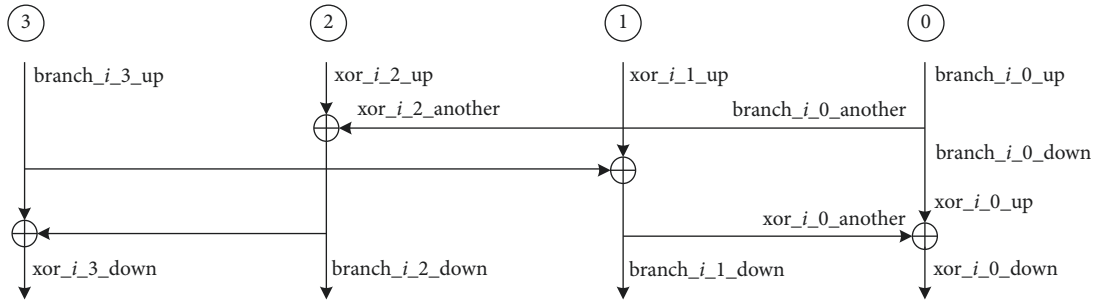

Figure 2: Partial variables used in block XOR operation.

```
r: the number of rounds
n: the expected number of active S-Boxes in the whole trail
(1) for i ⟵ 1 to r do
(2)    for j ⟵ 0 to 15 do
(3)       if before_sbox_mask_i_j = 0 AND after_sbox_mask_i_j = 0 then
(4)          flag_i_j = 0
(5)       else
(6)          flag_i_j = 1
(7)       end if
(8)    end for
(9) end for
(10)
(11) total_num = ∑_{i=0,j=0}^{i=r,j=15} flag_i_j
(12)
(13) total_num = n
```

Algorithm 8: Finding linear trails with the minimal number of active S-boxes.

Constraints of finding linear trails with the optimal bias are presented in Algorithm 9.

Set the initial value of total_bias by considering all S-boxes with the optimal bias. Adjust the value of total_bias by adding 1 till obtaining the trail with the optimal bias.

3.3. Finding Related-Key Differential Trails. When searching for differential trails in related-key setting, some constraints are given in Algorithm 10 to illustrate the related-key relationship. And we also need to construct some constraints which describe difference propagations through the key schedule. Operations in the key schedule are similar to those in the encryption algorithm and constraints could refer to Algorithm 2, 3, and 6.

> Detailed codes are presented in the GitHub and the optimal trails are illustrated in Table 6.

3.4. Finding Impossible Differential Trails. In order to find impossible differential trails, we need to set the pattern of input and output differences and check the satisfaction of constraints. If the solver returns a trail, there is no impossible differential trail under this pattern. If returns invalid, an impossible differential trails are found. Constraints are listed in Algorithm 11.

As a result, we obtain 4 different 7-round impossible differential trails, which are

$$\begin{aligned}
&0x0000000000010000 \longrightarrow 0x0001000000000000,\\
&0x0000000000100000 \longrightarrow 0x0010000000000000,\\
&0x0000000001000000 \longrightarrow 0x0100000000000000,\\
&0x0000000010000000 \longrightarrow 0x1000000000000000.
\end{aligned} \tag{7}$$

The former is the input pattern, and the latter is the output pattern of the 7-round BORON, where "1" represents the active nibble and "0" represents the inactive nibble.

4. Key-Recovery Attacks

4.1. Differential Cryptanalysis. Considering the 8-round differential trail with the probability of 2^{-62} as a distinguisher illustrated in Table 4, a key-recovery attack towards 9-round BORON is presented in the following by adding one more round on the tail of the distinguisher. We omit the permutation layer of the last round due to linearity. We adopt the 80-bit key schedule in this key-recovery attack.

The input difference of this distinguisher is 0x0000080000100000, and the output difference is 0x0041004100080009. We choose 2^{62} plaintext pairs whose differences are equal to 0x0000080000100000. Filter these plaintext pairs and keep ones whose ciphertext differences are equal to 0x00??00??000?000?, where ? represents the

```
r: the number of rounds
2^{-b}: the expected bias of the whole trail
 (1) for α ⟵ 0 to 15 do
 (2)    for β ⟵ 0 to 15 do
 (3)       linear_dist[α][β] = bias(α, β)
 (4)    end for
 (5) end for
 (6)
 (7) for i ⟵ 0 to r do
 (8)    for j ⟵ 0 to 15 do
 (9)       bias_i_j
(10)       = linear_dist[before_sbox_mask_i_j][after_sbox_mask_i_j]
(11)    end for
(12) end for
(13)
(14) total_bias = 1 − total_num + ∑_{i=0,j=0}^{i=r,j=15} bias_i_j
(15)
(16) total_bias = b
```

Algorithm 9: Finding linear trails with the optimal bias.

```
r: the number of rounds
 (1) for i ⟵ 1 to r do
 (2)    before_sbox_value1_i =
 (3)    before_xorsubkey_value1_i ⊕ subkey_value1_i
 (4)    before_sbox_value2_i =
 (5)    before_xorsubkey_value2_i ⊕ subkey_value2_i
 (6) end for
 (7)
 (8) for i ⟵ 1 to r do
 (9)    after_blockxor_difference_(i + 1) =
(10)    before_xorsubkey_value1_i ⊕ before_xorsubkey_value2_i
(11) end for
```

Algorithm 10: Finding related-key differential trails.

```
r: the number of rounds
x: input pattern of the trail
y: output pattern of the trail
 (1) flag_head = flag_0_0
 (2) for i ⟵ 1 to 15 do
 (3) flag_head = flag_0_i‖flag_head
 (4) end for
 (5)
 (6) flag_tail = flag_(r − 1)_0
 (7) for i ⟵ 1 to 15 do
 (8) flag_tail = flag_(r − 1)_i‖flag_tail
 (9) end for
(10)
(11) flag_head = x
(12) flag_tail = y
```

Algorithm 11: Finding impossible differential trails.

unknown difference. On average, 2^{22} plaintext pairs are left because the probability that a pair satisfies the limitation of this distinguisher is 2^{-40}.

Guess 24 bits of subkey K_9 including k_0, k_1, k_2, k_3, k_{16}, k_{17}, k_{18}, k_{19}, k_{32}, k_{33}, k_{34}, k_{35}, k_{36}, k_{37}, k_{38}, k_{39}, k_{48}, k_{49}, k_{50}, k_{51}, k_{52}, k_{53}, k_{54}, and k_{55}. Set 2^{24} counters and initialize their

values as zero. Each candidate subkey has a counter. And, values of counters represent the number of plaintext pairs that satisfy the distinguisher under the corresponding candidate subkey.

For the right candidate subkey, one plaintext pair satisfies the distinguisher. However, for wrong candidate subkeys, 2^{-24} plaintext pair satisfies the distinguisher on average. We pick the candidate subkey whose counter is maximum as the right subkey. The time complexity is 2^{46}, the memory complexity is 2^{24}, and the data complexity is 2^{63}.

For the 80-bit master key, we have already guessed 24 bits. To obtain the left 56 bits, we use a brute-force search. The time complexity is 2^{56}. The memory and data complexity could be omitted.

Hence, we give a key-recovery attack towards 9-round BORON with time complexity 2^{56}, memory complexity 2^{24}, and data complexity 2^{63}.

We use the method in [21] to evaluate the success probability of this attack. The success probability P_S is computed as follows:

$$P_S = \Phi\left(\frac{\sqrt{\mu S_N} - \Phi^{-1}(1 - 2^{-a})}{\sqrt{S_N + 1}}\right), \quad (8)$$

where $\mu = pN$, p is the probability of the differential trail, N is the number of plaintext pairs, a is the advantage, and S_N is the signal-to-noise ratio. In our differential cryptanalysis,

$$P_S = \Phi\left(\frac{\sqrt{2^{-62} \times 2^{62} \times 2^{24}} - \Phi^{-1}\left(1 - 2^{-1}\right)}{\sqrt{2^{24} + 1}}\right) = 0.841. \quad (9)$$

4.2. Linear Cryptanalysis. Based on the 9-round optimal linear trail listed in Table 5, we could present a key-recovery attack against 11-round BORON by linking one more round at the head and tail of this 9-round linear trail.

The number of plaintexts required in the key-recovery attack is equal to $c\varepsilon^{-2}$, where ε is the bias of the linear approximation and c is a constant [10]. Set $c = 8$, and then we need 2^{63} plaintexts to achieve this attack and the success probability of this attack is 96.7%.

We need to guess 28 bits of subkey K_0 and 24 bits of subkey K_{11}. Considering the 128-bit key schedule, there are 10 same bits between K_0 and K_{11}. In total, we need to guess 42 bits to obtain 5-bit key information. This key-recovery attack requires $2^{63} \cdot 2^{42} = 2^{105}$ one-round encryptions, which is equivalent to $2^{101.54}$ 11-round encryptions. The left 123 bits need to be searched by brute-force.

Hence, the time complexity is 2^{123}, memory complexity is 2^{42}, and data complexity is 2^{63}.

5. Conclusion

In this paper, we present the first third-party cryptanalysis of the lightweight block cipher BORON against differential and linear cryptanalysis. By the automatic tool, we search for differential and linear trails with the minimal number of active S-boxes and trails with the optimal probability and bias. Considering the optimal trails as distinguishers, we mount key-recovery attacks. Utilizing the 8-round differential trail with the optimal probability 2^{-62}, we give a key-recovery attack towards 9-round BORON whose time complexity is 2^{56}, data complexity is 2^{63}, and memory complexity is 2^{24}. By the 9-round linear trail with the optimal bias 2^{-30}, we describe the key-recovery attack towards 11-round BORON whose time complexity is 2^{123}, data complexity is 2^{63}, and memory complexity is 2^{42}. Besides differential and linear cryptanalysis, there are other powerful cryptanalysis techniques. Further security evaluations could be made in future work.

Data Availability

The codes and trails used to support the findings of this study have been deposited in the GitHub (https://github.com/CatherineLiang/Cryptanalysis-of-BORON).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors thank Associate Professor Wu Hongjun from Nanyang Technological University for careful discussion. This work was supported by the National Cryptography Development Fund (no. MMJJ20170102), National Natural Science Foundation of China (grant no. 61572293), Major Scientific and Technological Innovation Projects of Shandong Province, China (no. 2017CXGC0704), Fundamental Research Fund of Shandong Academy of Sciences (grant no. 2018:12-16), and the China Scholarship Council (no. CSC201906220073).

References

[1] A. Bogdanov, L.-R. Knudsen, G. Leander et al., "PRESENT: an ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems—CHES 2007*, P. Paillier and I. Verbauwhede, Eds., vol. 4727, pp. 450–466, Springer, Heidelberg, Germany, 2007.

[2] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-bit blockcipher CLEFIA (extended abstract)," in *Fast Software Encryption*, A. Biryukov, Ed., vol. 4593, pp. 181–195, Springer, Heidelberg, Germany, 2007.

[3] J. Borghoff, A. Canteaut, T. Güneysu et al., "PRINCE—a low-latency block cipher for pervasive computing applications," in *Advances in Cryptology—ASIACRYPT 2012*, X. Wang and K. Sako, Eds., vol. 7658, pp. 208–225, Springer, Heidelberg, Germany, 2012.

[4] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," 2013, https://eprint.iacr.org/2013/404.pdf.

[5] C. Beierle, J. Jean, S. Kölbl et al., "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *Advances in Cryptology—CRYPTO 2016*, M. Robshaw and J. Katz, Eds., vol. 9815, pp. 123–153, Springer, Heidelberg, Germany, 2016.

[6] S. Banik, S.-K. Pandey, T. Peyrin, Y. Sasaki, S.-M. Sim, and Y. Todo, "GIFT: a small present," in *Lecture Notes in Computer Science*, W. Fischer and N. Homma, Eds., vol. 10529, pp. 321–345, Springer, Heidelberg, 2017.

[7] R. Avanzi, "The QARMA block cipher family," 2016, https://eprint.iacr.org/2016/444.pdf.
[8] G. Bansod, N. Pisharoty, and A. Patil, "BORON: an ultra-lightweight and low power encryption design for pervasive computing," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 3, pp. 317–331, 2017.
[9] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptology*, vol. 4, no. 1, pp. 3–72, 1991.
[10] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Advances in Cryptology—EUROCRYPT '93*, vol. 765, pp. 386–397, Springer, Heidelberg, Germany, 1993.
[11] S. Kölbl, G. Leander, and T. Tiessen, "Observations on the SIMON block cipher family," in *Lecture Notes in Computer Science*, R. Gennaro and M. Robshaw, Eds., vol. 9215, pp. 161–185, Springer, Heidelberg, Germany, 2015.
[12] Y. Liu, Q. Wang, and V. Rijmen, "Automatic search of linear trails in ARX with applications to SPECK and Chaskey," in *Applied Cryptography and Network Security*, M. Manulis, A. R. Sadeghi, and S. Schneider, Eds., vol. 9696, pp. 485–499, Springer, Heidelberg, Germany, 2016.
[13] N. Mouha and B. Preneel, "towards finding optimal differential characteristics for ARX: application to Salsa20," 2013, https://eprint.iacr.org/2013/328.pdf.
[14] N. Mouha, Q. Wang, D. Gu, and B. Preneel, "Differential and linear cryptanalysis using mixed-integer linear programming," in *Information Security and Cryptology*, C.-K. Wu, M. Yung, and D. Lin, Eds., vol. 7537, pp. 57–76, Springer, Heidelberg, Germany, 2011.
[15] S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song, "Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers," in *Lecture Notes in Computer Science*, P. Sarkar and T. Iwata, Eds., vol. 8873, pp. 158–178, Springer, Heidelberg, Germany, 2014.
[16] V. Ganesh, T. Hansen, M. Soos, D. Liew, and R. Govostes, "STP constraint solver," 2007, https://github.com/stp/stp.
[17] R. Ankele and S. Kölbl, "Mind the gap—a closer look at the security of block ciphers against differential cryptanalysis," in *Selected Areas in Cryptography—SAC 2018*, C. Cid and M.-J. Jacobson Jr., Eds., vol. 11349, pp. 163–190, Springer, Heidelberg, Germany, 2019.
[18] A. Biryukov and V. Velichkov, "Automatic search for differential trails in ARX ciphers," in *Topics in Cryptology—CT-RSA 2014*, J. Benaloh, Ed., vol. 8366, pp. 227–250, Springer, Heidelberg, Germany, 2016.
[19] A. Biryukov, V. Velichkov, and Y.-L. Corre, "Automatic search for the best trails in ARX: application to block cipher SPECK," in *Fast Software Encryption*, T. Peyrin, Ed., vol. 9783, pp. 289–310, Springer, Heidelberg, Germany, 2016.
[20] Y. Sasaki and Y. Todo, "New impossible differential search tool from design and cryptanalysis aspects," in *Lecture Notes in Computer Science*, J. S. Coron and J. Nielsen, Eds., vol. 10212, pp. 185–215, Springer, Heidelberg, Germany, 2017.
[21] A.-A. Selçuk and A. Biçak, "On probability of success in linear and differential cryptanalysis," in *Security in Communication Networks*, S. Cimato, G. Persiano, and C. Galdi, Eds., vol. 2576, pp. 174–185, Springer, Heidelberg, Germany, 2002.