Cryptanalysis of Subterranean-SAE

Fukang Liu[1,3,a)] Takanori Isobe[2,3]

Abstract: Subterranean 2.0 designed by Daemen, Massolino and Rotella is a Round 1 candidate of the NIST Lightweight Cryptography Standardization process. In the official document of Subterranean 2.0, the designers have made a cryptanalysis of the state collisions in unkeyed absorbing by reducing the number of rounds to absorb the message from 2 to 1. However, no cryptanalysis of the authenticated encryption scheme Subterranean-SAE is made. For Subterranean-SAE, the designers introduce 8 blank rounds to separate the controllable input and output, and expect that 8 blank rounds can achieve a sufficient diffusion. Therefore, it is meaningful to investigate the security by reducing the number of blank rounds. By reducing the number of blank rounds to 4, we can mount a key-recovery attack with time complexity 2^{122} and data complexity $2^{69.5}$.

Keywords: AEAD, Subterranean 2.0, key-recovery attack, conditional cube tester

1. Introduction

The National Institute of Standards and Technology (NIST) started a public lightweight cryptography competition project in as early as 2013 and initiated the call for submissions in 2018, with the hope to select a lightweight cryptographic standard by combining the efforts of both academia and industry. The 56 Round 1 candidates of the NIST Lightweight Cryptography Standardization project became public on April 18, 2019.

Benefiting from the development in cryptanalysis in these years, some submitted primitives have been well analyzed by the designers. However, to have a better understanding, the third-party cryptanalysis is important as well. In this paper, our target is the primitive Subterranean 2.0 [1] designed by Daemen, Massolino and Rotella. The main reason is that we observed that its structures in keyed and unkeyed mode are interesting and its round function is very simple. Moreover, the degree of one-round permutation is only 2, which gives us an impression that the conditional cube attack [2] may be feasible. Since the 8 blank rounds in Subterranean-SAE are used to separate the controllable input and output, we believe that it is still interesting and meaningful to investigate its security when the number of blank rounds is reduced.

1.1 Our Contributions

When the number of blank rounds is reduced to 4, the key-recovery attack will be feasible. The attack procedure is composed of two steps. The first step is to recover some secret state bits by using a conditional cube tester. The second step is to guess some key bits to construct a linear boolean equations system, each solution of which corresponds to the full key. In this way, we can achieve the key-recovery attack with time complexity 2^{122} and data complexity $2^{69.5}$.

[1] East China Normal University
[2] NICT
[3] University of Hyogo
[a)] liufukangs@163.com

2. Description of Subterranean 2.0

In this section, we will briefly describe the round function of Subterranean 2.0 and the authenticated encryption scheme Subterranean-SAE.

2.1 Round Function

The subterranean 2.0 round function is composed of 4 simple operations and operates on a 257-bit state. Denote the 257-bit state by s and the four operations by χ, ι, θ, π. The one-round permutation $R = \pi \circ \theta \circ \iota \circ \chi$ is detailed as follows, where $s[i]$ represents the i-th bit of s.

$$\begin{aligned}
\chi &: s[i] \leftarrow s[i] \oplus \overline{s[i+1]}s[i+2],\\
\iota &: s[0] \leftarrow s[0] \oplus 1,\\
\theta &: s[i] \leftarrow s[i] \oplus s[i+3] \oplus s[i+8],\\
\pi &: s[i] \leftarrow s[12i],
\end{aligned}$$

where $0 \le i \le 256$. In addition, we denote the state after χ, ι, θ operation by s_χ, s_ι and s_θ, respectively.

2.2 The Subterranean-SAE Authenticated Encryption Scheme

Based on the subterranean 2.0 round function, the designers have constructed an authenticated encryption scheme named Subterranean-SAE, as illustrated in Figure 1. In this scheme, the input consists of 128-bit key K, 128-bit nonce N, the associated data A and the message M. The output is the ciphertext C and tag T. The procedure to generate the ciphertext and tag can be briefly described as follows:

Step 1: **Absorb the key:** Initialize a state s with all bits set to 0. Split the 128-bit key K into four 32-bit blocks K_0, K_1, K_2 and K_3. Then, make four times of consecutive

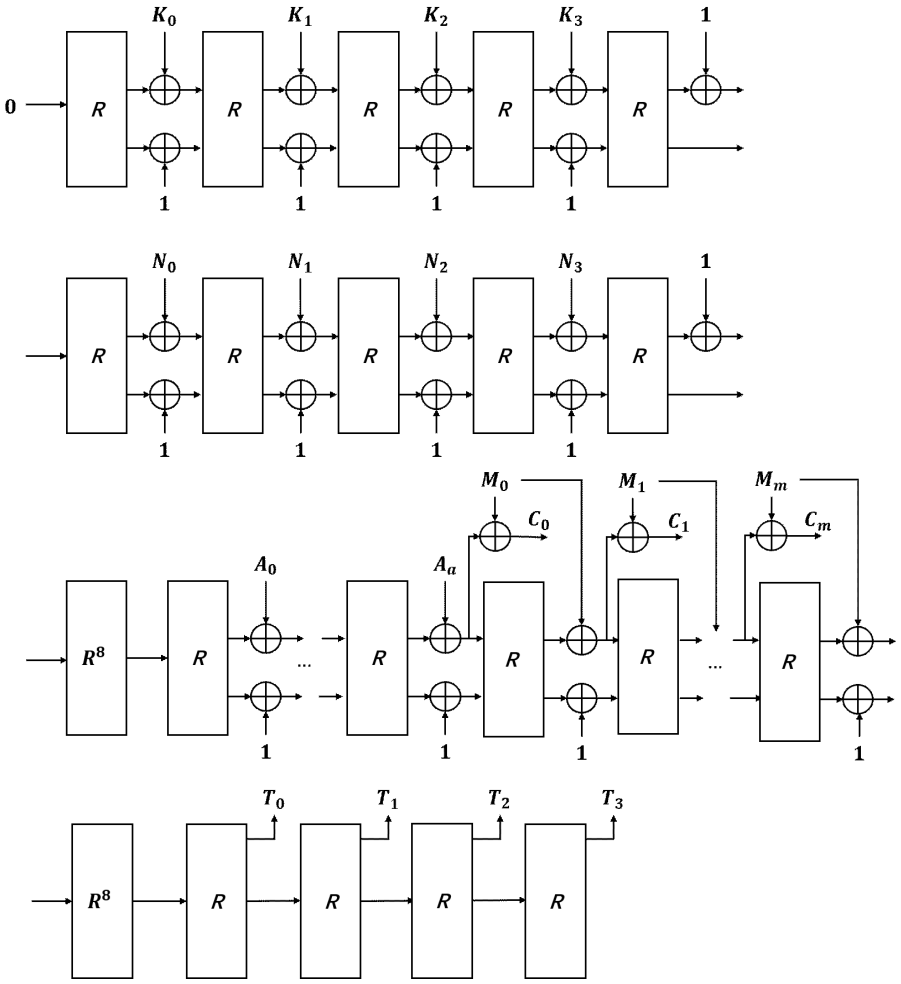

Fig. 1: The construction of Subterranean-SAE

calls to $duplex(s, K_i)$ $(0 \leq i \leq 3)$ to update the internal state. Finally, make a call to $duplex(s, NULL)$ to further update the internal state, where $NULL$ represents an empty string.

Step 2: **Absorb the nonce:** Split the 128-bit nonce N into four 32-bit blocks N_0, N_1, N_2 and N_3. Then, make four times of consecutive calls to $duplex(s, N_i)$ $(0 \leq i \leq 3)$ to update the internal state. Finally, make a call to $duplex(s, NULL)$ to further update the internal state.

Step 3: **Blank rounds:** Make 8 times of consecutive calls to $duplex(s, NULL)$ to update the internal state.

Step 4: **Absorb the associated data:** Split the $|A|$-bit associated data A into a series of 32-bit blocks, denoted by A_i $(0 \leq i < \lceil |A|/32 \rceil)$, where $|A|$ denotes the length of A. Then, make $\lceil |A|/32 \rceil$ times of consecutive calls to $duplex(s, A_i)$ $(0 \leq i < \lceil |A|/32 \rceil)$ to update the internal state. If $|A|$ is a multiple of 32 (the case when A is empty also belongs to this case), make one more call to $duplex(s, NULL)$ to update the internal state.

Step 5: **Message encryption:** Split the $|M|$-bit $(|M| \geq 0)$ message M into a series of 32-bit blocks, denoted by M_i $(0 \leq i < \lceil |M|/32 \rceil)$, where $|M|$ denotes the length of M. Then, make $\lceil |M|/32 \rceil$ times of consecutive calls to $duplex(s, M_i)$ $(0 \leq i \leq \lceil |M|/32 \rceil)$ to update the internal state. Before each call to $duplex(s, M_i)$, make a call to $extract(s)$ $(0 \leq i < \lceil |M|/32 \rceil)$ and then the corresponding ciphertext is $C_i = extract(s) \oplus M_i$. If $|M|$ is a multiple of 32 (the case when M is empty also belongs to this case), make one more call to $duplex(s, NULL)$ to update the internal state.

Step 6: **Blank rounds:** Make 8 times of consecutive calls to $duplex(s, NULL)$ to update the internal state.

Step 7: **Extract tag:** Make 4 times of consecutive calls to $duplex(s, NULL)$. After each call to $duplex(s, NULL)$, make a call to $extract(s)$ to obtain 32-bit T_i $(0 \leq i \leq 3)$.

The details of $duplex(s, \sigma)$ and $extract(s)$ are described in Algorithm 1 and Algorithm 2, where σ is a bit string with at most 32 bits. The readers can also refer to the official document of Subterranean 2.0 to have a better understanding.

The above pseudocode is slightly different from the official document since we introduced two extra arrays order0[] and order1[]. The details of the order0[] and order1[] are specified in Table 1.

Algorithm 1 $duplex(s, \sigma)$

1: $R(s)$
2: $j = 0$
3: **for** j from 0 to $|\sigma| - 1$ **do**
4: s[order0[j]]=s[order0[j]]$\oplus\sigma$[j]
5: **end for**
6: s[order0[j]]=s[order0[j]]$\oplus$1

Algorithm 2 $extract(s)$

1: **for** j from 0 to 31 **do**
2: z[j]=s[order0[j]]$\oplus$ s[order1[j]]
3: **end for**
4: return z

Table 1: The details of order0[] and order1[]

i	0	1	2	3	4	5	6	7	8
order0[i]	1	176	136	35	249	134	197	234	64
i	9	10	11	12	13	14	15	16	17
order0[i]	213	223	184	2	95	15	70	241	11
i	18	19	20	21	22	23	24	25	26
order0[i]	137	211	128	169	189	111	4	190	30
i	27	28	29	30	31	32	–	–	–
order0[i]	140	225	22	17	165	256	–	–	–
i	0	1	2	3	4	5	6	7	8
order1[i]	256	81	121	222	8	123	60	23	193
i	9	10	11	12	13	14	15	16	17
order1[i]	44	34	73	255	162	242	187	16	246
i	18	19	20	21	22	23	24	25	26
order1[i]	120	46	129	88	68	146	253	67	227
i	27	28	29	30	31	–	–	–	–
order1[i]	117	32	235	240	92	–	–	–	–

3. Key-recovery Attack on 4-Blank-Round Subterranean-SAE

When the number of blank rounds is reduced to 4, a key-recovery attack will be feasible. The attack procedure can be divided into two steps on the whole.

Step 1: Recover y secret bits of the state after N_1 is absorbed with the conditional cube tester.

Step 2: Guess $(128 - x)$ bits of the secret key and let the remaining x secret key bits as variables. Then with the y recovered state bits, construct a quadratic boolean equation system in terms of the x variables. There will be $\frac{x(x-1)}{2}$ quadratic terms and we replace them with $\frac{x(x-1)}{2}$ new variables. In this way, we can obtain y linear equations in terms of $x+\frac{x(x-1)}{2}$ variables. If $y \geq x+\frac{x(x-1)}{2}$, the x secret key bits can be computed by solving this linear equation system.

To make this part more clear, we first consider an equivalent representation of the state transform. Suppose s^i $(i \geq 0)$ denotes the input state of the $(i+1)$-th round. Then, after χ, ι, θ and π operation, the state is denoted by s^i_χ, s^i_ι, s^i_θ and s^{i+1}, respectively. Moreover, to simulate the construction of Subterranean-SAE, we suppose 32 bits of s^i can be influenced by an attacker and he can also extract 32-bit information from s^i, which is $z^i = extract(s^i)$.

Similar with the idea of conditional cube attack, we hope the cube sum will depend on whether the predefine conditions hold. In other words, supposing there is one condition, the cube sum will always be zero if the condition holds. However, when this condition does not hold, the cube sum is uncertain and equals zero with a very low probability of 2^{-32}. Hence, according to the cube sum, we can directly determine whether the condition holds and obtain a linear equation of the secret state bits.

To make the condition cube tester work, we will set cube variables at s^i $(0 \leq i \leq 2)$ and suppose the attacker can only get z^i $(i \geq 8)$ in the key-recovery attack. The main idea can be briefly described as follows:

(1) Set 32 cube variables in s^2, denoted by $v^2_j = s^2$[order0[j]] $(0 \leq j \leq 31)$.

(2) Set n cube variables in s^1, denoted by $v^1_j = s^1$[order0[r]] where $0 \leq j < n$ and $r \in \{k | 0 \leq k \leq 31\}$.

(3) Set $33-n$ cube variables in s^0, denoted by $v^0_j = s^0$[order0[r]] where $0 \leq j < 33 - n$ and $r \in \{k | 0 \leq k \leq 31\}$.

Suppose $f(s^0[x])$ represents either $s^0[x]$ or $s^0[x]\oplus 1$. There will be some constraints on v^0 and v^1 as follows:

Constraint 1: v^0 are not next to each other in s^0, i.e. they will not multiply with each other after one-round permutation. After one-round permutation for v^0, they still will not be next to each other in s^1.

Constraint 2: v^1 are not next to each other in s^1, i.e. they will not multiply with each other after one-round permutation.

Constraint 3: If the specified bit condition $f(s^0[x]) = 0$ holds, after one-round permutation for v^0, none of v^0 will be next to any of v^1.

Constraint 4: If the specified bit condition $f(s^0[x]) = 0$ does not hold, after one-round permutation for v^0, v^0 will be next to at least one of v^1.

With the above constraints, we can know that s^2 will be linear with (v^0, v^1) if $f(s^0[x]) = 0$ holds. Since there are extra 32 cube variables in s^2 and the degree of one-round permutation is 2, we can know that the degree of z^8 is at most $2^6 = 64$ in terms of s^2. Thus, the term $v^0v^1v^2$ with degree 65 will not appear in the expression of z^8 and the cube sum of s^8 will be zero in this case.

However, when the condition does not hold, s^2 will contain a quadratic term. Then, the term $v^0v^1v^2$ with degree 65 will appear in the expression of z^8 and the cube sum of s^8 cannot be predicted.

Consequently, according to the cube sum, we can directly recover the one secret bit $s^0[x]$ as the full-state recovery attack. With a modified searching method in [3], we can find 22 valid choices for (v^0, v^1) and therefore recover 22 secret bits of s^0, as listed in Table A·1 and Table A·2 in Appendix A.1. For a better understanding of the two tables, we take the first choice for instance and make some explanation.

For the first choice in Table A·1 to recover the secret state $s^0[2]$, the cube variables v^0 are set at 6 bit positions of s^0 and v^1 are set at 27 bit positions of s^1. Specifically,

$$v_0^0 = s^0[1], v_1^0 = s^0[30], v_2^0 = s^0[111], v_3^0 = s^0[137],$$
$$v_4^0 = s^0[189], v_5^0 = s^0[233],$$
$$v_0^1 = s^1[1], v_1^1 = s^1[4], v_2^1 = s^1[11], v_3^1 = s^1[15],$$
$$v_4^1 = s^1[17], v_5^1 = s^1[22], v_6^1 = s^1[30], v_7^1 = s^1[35],$$
$$v_8^1 = s^1[64], v_9^1 = s^1[70], v_{10}^1 = s^1[95], v_{11}^1 = s^1[111],$$
$$v_{12}^1 = s^1[128], v_{13}^1 = s^1[134], v_{14}^1 = s^1[137], v_{15}^1 = s^1[140],$$
$$v_{16}^1 = s^1[165], v_{17}^1 = s^1[169], v_{18}^1 = s^1[176], v_{19}^1 = s^1[184],$$
$$v_{20}^1 = s^1[189], v_{21}^1 = s^1[197], v_{22}^1 = s^1[211], v_{23}^1 = s^1[223],$$
$$v_{24}^1 = s^1[225], v_{25}^1 = s^1[241], v_{26}^1 = s^1[249].$$

Once the condition $s^0[2] = 0$ holds, the cube sum of z^8 is zero. However, when $s^0[2] \neq 0$, three bits of s^2 will always contain a quadratic term $v_0^0 v_0^1$. In addition, we have verified that there will always be a cubic term in a certain bit of s^3. Since there are 65 cube variables and sufficient number of rounds to diffuse v^0, v^1 and v^2, we expect there will be a term of degree 65 in z^8. Therefore, based on the cube sum of z^8, we directly recover the secret state bit $s^0[2]$ as follows:

$$\sum z^8 \neq 0 \Rightarrow s^0[2] = 1,$$
$$\sum z^8 = 0 \Rightarrow s^0[2] = 0.$$

Now, we describe how to use the above method the recover the secret state after N_1 is absorbed. Set the associated data A as empty and the first message block M_0 as a zero constant. Denote the state after N_i is absorbed as NS_i^{in}, as depicted in Figure 2. The attack procedure can be described as follows:

Step 1: Send an encryption query (N, A, M) and obtain (C, T).

Step 2: Keep M_0 and N_0 as constant. Treat NS_1^{in}, NS_2^{in} and NS_3^{in} as s^0, s^1 and s^2 respectively. For each choice of the 65 cube variables in Table A·1 and Table A·2, send= 2^{65} encryption queries (N, A, M) with N taking all possible 2^{65} values and compute the sum of C_0. If the sum is zero, the corresponding condition will hold. If it is not zero, the condition will not hold. Whatever the sum is, we can recover one secret bit of NS_0^{ot}. The time and data complexity to recover the 22 secret bits of NS_0^{ot} are both $22 \times 2^{65} = 2^{69.5}$.

After recovering the 22 secret bits of NS_0^{ot}, we will start to construct 22 equations. Suppose K_0, K_1 and K_2 are fixed, we then use a trivial MILP-based method to find the maximum number of variables in K_3 which are still linear after two-round permutation and the Gurobi solver returns 9. The 9 positions are listed below:

$$11, 35, 70, 95, 140, 165, 190, 213, 241.$$

In other words, if we fix the remaining $32 - 9 = 23$ bits of K_3 as constants, NS_0^{in} will be linear with these 9 secret bits. Since NS_0^{ot} is quadratic with NS_0^{in}, we therefore cannot construct a linear equation system. Guessing 3 more bits among the 9 secret bits will reduce the number of variables to 6. Therefore, there will be $6 \times (6-1)/2 = 15$ quadratic terms. By replacing the 15 quadratic terms with 15 new variables, we can now know that NS_0^{ot} is linear with the $6 + 15 = 21$ variables. Since 22 bits of NS_0^{ot} has been recovered, we can construct 22 linear equations in terms of 21 variables. It is expected there is only one solution for each guess of K_i $(0 \leq i \leq 3)$. For each solution, we compute the tag T' the corresponding ciphertext C', only when $T = T'$ and $C' = C$ will imply the recovered key is correct.

3.1 Complexity Evaluation

The key-recovery attack is divided into two steps. The first step is to recover 22 secret state bits. The time complexity and data complexity at this step is $22 \times 2^{65} \approx 2^{69.5}$. At the second step, we guess 122 secret key bits and construct an equation system to compute the remaining 6 secret key bits. It is expected that there is only one solution for each guess. Therefore, we need to guess 2^{122} times to find the correct key. In total, the time complexity and data complexity of key-recovery attack are 2^{112} and $2^{69.5}$, respectively.

4. Conclusion

We investigate the security of a reduced variant of Subterranean-SAE by reducing the number of blank rounds to 4 from 8. Relying on the idea of conditional cube tester, we can firstly recover 22 secret state bits. Then with a guess-and-determine method, the search space for the secret key is reduced to 2^{122} from 2^{128}. As a result, we can mount key-recovery attack for this variant with time complexity 2^{122} and data complexity $2^{69.5}$.

References

[1] Joan Daemen, Pedro Maat Costa Massolino and Yann Rotella: The Subterranean 2.0 Cipher Suite. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates (2019)

[2] Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang and Jingyuan Zhao: Conditional Cube Attack on Reduced-Round Keccak Sponge Function. In: Coron JS., Nielsen J.(eds) EUROCRYPT 2017. LNCS, vol. 10211, pp. 259-288. Springer, Cham (2017)

[3] Fukang Liu, Zhenfu Cao and Gaoli Wang: Finding Ordinary Cube Variables for Keccak-MAC with Greedy Algorithm. In: Attrapadung N., Yagi T.(eds) IWSEC 2019. LNCS, vol. 11689, pp. 287-305. Springer, Cham (2019)

Appendix

A.1 Tables

We present some tables in this section.

Table A·1: Cube variables for conditional cube tester

Bit positions in s^0	1, 30, 111, 137, 189, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 197, 211, 223, 225, 241, 249
condition	$s^0[2] = 0$
Bit positions in s^0	2, 30, 137, 189,
Bit positions in s^1	2, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 197, 211, 213 , 223, 225, 234 , 241, 249
condition	$s^0[3] = 0$
Bit positions in s^0	2, 30, 111, 137, 189, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 197, 211, 223, 225, 241, 249
condition	$s^0[1] = 1$
Bit positions in s^0	4, 30, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[5] = 0$
Bit positions in s^0	11, 30, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[10] = 1$
Bit positions in s^0	15, 137, 189, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[16] = 0$
Bit positions in s^0	22, 111, 137, 189, 223,
Bit positions in s^1	2, 4, 11, 15, 17, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[21] = 1$
Bit positions in s^0	64, 30, 111, 137, 189, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 128, 137, 140, 165, 169, 176, 184, 189, 197 , 211, 213, 223, 225, 234, 241, 249
condition	$s^0[65] = 0$
Bit positions in s^0	64, 30, 111, 137, 189, 223,
Bit positions in s^1	1, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[63] = 1$
Bit positions in s^0	70, 30, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176 , 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[69] = 1$
Bit positions in s^0	95, 30, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 136, 140, 165, 169, 176, 184, 189, 197, 211, 213, 223 , 225, 234, 241, 249
condition	$s^0[96] = 0$

Table A·2: Cube variables for conditional cube tester

Bit positions in s^0	111, 30, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 136, 140, 165, 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[112] = 0$
Bit positions in s^0	134, 30, 111, 189, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 165, 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[133] = 1$
Bit positions in s^0	136, 30, 189, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[135] = 1$
Bit positions in s^0	165, 30, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184 , 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[166] = 0$
Bit positions in s^0	184, 30, 137, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[185] = 0$
Bit positions in s^0	197, 30, 111, 137, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 169, 176, 184, 189, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[196] = 1$
Bit positions in s^0	211, 30, 137, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 136, 140, 165, 169, 176, 184, 189, 197, 211 , 213, 223, 225, 234, 241, 249
condition	$s^0[212] = 0$
Bit positions in s^0	213, 30, 137, 223,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 136, 140, 165, 169, 176, 184, 190 , 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[214] = 0$
Bit positions in s^0	225, 30, 111, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 176, 184, 189 , 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[226] = 0$
Bit positions in s^0	241, 30, 111, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 176, 184, 190 , 197, 211 , 213 , 223, 225, 234, 241, 249
condition	$s^0[240] = 1$
Bit positions in s^0	249, 30, 111, 137, 189,
Bit positions in s^1	1, 4, 11, 15, 17, 22, 30, 35, 64, 70, 95, 111, 128, 134, 137, 140, 165, 176, 184, 190, 197, 211, 213, 223, 225, 234, 241, 249
condition	$s^0[248] = 1$

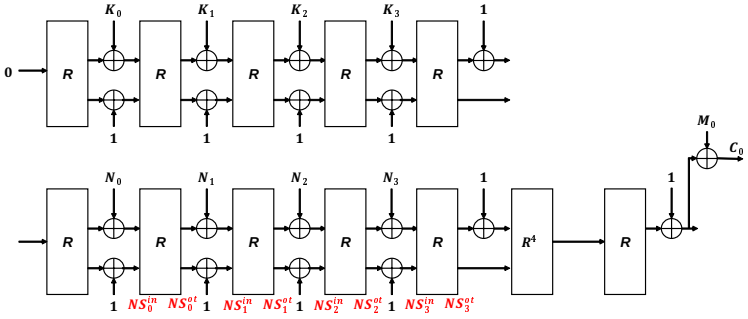

Fig. 2: Key recovery attack