Practical Key-recovery Attacks on Round-Reduced Ketje Jr, Xoodoo-AE and Xoodyak

Haibo Zhou\textsuperscript{1}, Zheng Li\textsuperscript{1}, Xiaoyang Dong\textsuperscript{2,*}, Keting Jia\textsuperscript{3,*}, and Willi Meier\textsuperscript{4}

\textsuperscript{1} Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, School of Mathematics, Shandong University, Jinan 250100, China \{zhouhaibo, lizhengcn\}@mail.sdu.edu.cn

\textsuperscript{2} Institute for Advanced Study, Tsinghua University, P. R. China

\textsuperscript{3} Department of Computer Science and Technology, Tsinghua University, P. R. China \{xiaoyangdong, ktjia\}@tsinghua.edu.cn

\textsuperscript{4} FHNW, Switzerland willi.meier@fhnw.ch

\textbf{Abstract.} Conditional cube attack was proposed by Huang et al. at EUROCRYPT 2017 to attack Keccak keyed mode. Inspired by dynamic cube attack, they reduce the degree by appending key bit conditions on the initial value (IV). Recently, Li et al. proposed new conditional cube attacks on Keccak keyed mode with extremely small degrees of freedom. In this paper, we find a new property on Li et al.’s method, and modify the new conditional cube attack for lightweight encryption algorithms using a 8-2-2 pattern, and apply it to 5-round Ketje Jr, 6-round Xoodoo-AE and Xoodyak, where Ketje Jr is among the 3rd round CAESAR competition candidates and Xoodyak is a Round 1 submission of the ongoing NIST lightweight cryptography project. Then we give the updated conditional cube attack analysis. All our results are of practical time complexity with negligible memory cost and our test codes are given in this paper. Notably, it is the first third-party cryptanalysis result for Xoodyak.

\textbf{Keywords:} Conditional Cube Attack, Keccak, Ketje Jr, Xoodoo, Xoodyak

1 Introduction

Authenticated encryption (AE) can provide confidentiality, integrity and authenticity for messages simultaneously. CAESAR competition [Com14], launched in 2014, aimed to find AE schemes with Security, Applicability, and Robustness. Totally, 57 candidates have been submitted to CAESAR in the first round competition. After three rounds of competition, only 6 authenticated encryption algorithms survived in the final portfolio.

\textsuperscript{*} Corresponding authors
Recently, NIST has initiated a process to solicit, evaluate, and standardize lightweight authenticated encryption algorithms with associated data (AEAD) and hashing, that are suitable for use in constrained environments where the performance of current NIST cryptographic standards is not acceptable. Till April 18, 2019, NIST has received 57 submissions, out of which 56 were selected as Round 1 Candidates.

**Ketje** [BDP+16] is one of the third round candidates of CAESAR competition [Com14], whose lightweight version is **Ketje Jr**, which leverages Keccak-p permutation with 200-bit internal state. Using a cube-attack-like method [DMP+15], Dong et al. [DLWQ17] give the first third party cryptanalysis on **Ketje Jr** with initialization phase reduced to 5 rounds. Then, Bi et al. [BDL+18] and Song et al. [SG18] improved the cube-attack-like cryptanalysis by MILP method, and obtained better attacks [SG18] on 5-round **Ketje Jr**. Song et al. [SG18] also give 6-round attacks when the recommended 96-bit key is reduced to 72 bits for Version 1 and 80 bits for Version 2. When targeting the encryption phase of **Ketje Jr**, Fuhr et al. [FNR18] described key recovery attacks on **Ketje Jr** with a rate extended to 32 or 40 bits (instead of the nominal 16 bits).

**Xoodoo** is another permutation proposed by Daemen et al. [DHAK18] at ToSC 2018. Song et al. [SG18] gave the first key-recovery attack on 6-round **Xoodoo-AE**, which is an artificial AE by using **Xoodoo** in Ketje style. In addition, the official AEAD scheme **Xoodyak** [DHAK18] based on **Xoodoo** is included in the Round 1 candidates of NIST lightweight cryptography competition. Besides the nonce-respecting security claim, the authors [DHAK18] also clarify the nonce-misuse case that: “Nonce violation and release of unverified decrypted ciphertext have no consequences for integrity and do not put the key in danger for **Xoodyak**”.

**Our Contribution.** In this paper, we investigate three lightweight AEAD schemes, namely **Ketje Jr** (v1 and v2), **Xoodoo-AE** and **Xoodyak**, with Li et al.’s [LDB+19] new conditional cube attack. Conditional cube attack was first introduced by Huang et al. [HWX+17] at EUROCRYPT 2017 to attack Keccak keyed modes. However, Huang et al.’s attack becomes invalid when applied to targets with very small degrees of freedom even with the help of MILP models [LDW17, SGSL18]. **Ketje Jr** (v1 and v2), **Xoodoo-AE** and **Xoodyak** are exactly such schemes. In fact, the previous attacks [DLWQ17, SG18] on reduced **Ketje Jr** or **Xoodoo-AE** are mainly based on Dinur et al.’s cube-attack-like cryptanalysis [DMP+15].

Recently, Li et al. [LDB+19] proposed a new conditional cube attack, which could work even on targets with extremely small degrees of freedom. They introduced the so-called kernel quadratic term to replace the Huang et al.’s conditional cube variables, which makes sure that no cubic term appears in the output of 2nd round.

In this paper, we study Li et al.’s method and discover some new properties, such as using a 8-2-2 pattern for the lightweight algorithm can control the conditional cube variables’ diffusion better; we also can adapt the 8-2-2 pattern
to the 7-2-2 pattern for some algorithms whose state has short columns. Then we apply it and get improved key-recovery attacks on reduced Ketje Jr (v1 and v2), Xoodoo-AE, as well as the first attack on reduced Xoodyak. The advantages of our attacks are summarized in Table 1, and described as follows:

- For 5-round initialization of Ketje Jr v1 with recommended key length, we could recover the 96-bit key in $2^{26.6}$ time with negligible memory cost, while the best previous attack needs $2^{36.86}$ time and $2^{18}$ memory.
- For 5-round initialization of Ketje Jr v2 with recommended key length, we could recover the 96-bit key in $2^{27.5}$ time with negligible memory cost, while the best previous attack needs $2^{34.91}$ time and $2^{15}$ memory.
- For 6-round Xoodoo-AE, we could recover the 128-bit key in $2^{40.5}$ time with negligible memory cost, while the best previous attack needs $2^{89}$ time and $2^{55}$ memory.
- For 6-round Xoodyak in nonce-misuse settings, we could recover the 128-bit key in $2^{43.8}$ time with negligible memory cost.

### Table 1: Summary of Key-recovery Attacks

<table>
<thead>
<tr>
<th>Target</th>
<th>b</th>
<th>K (b)</th>
<th>DF</th>
<th>Rounds</th>
<th>T</th>
<th>M</th>
<th>Source</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ketje Jr v1</td>
<td>200</td>
<td>96</td>
<td>86</td>
<td>5/13</td>
<td>$2^{36.86}$</td>
<td>$2^{26.6}$</td>
<td>[DLWQ17]</td>
<td>T1</td>
</tr>
<tr>
<td>Ketje Jr v2</td>
<td>200</td>
<td>96</td>
<td>86</td>
<td>5/13</td>
<td>$2^{34.91}$</td>
<td>$2^{27.5}$</td>
<td>[SG18]</td>
<td>T1</td>
</tr>
<tr>
<td>Xoodoo-AE</td>
<td>384</td>
<td>128</td>
<td>238</td>
<td>6/-</td>
<td>$2^{40.5}$</td>
<td>$2^{30}$</td>
<td>[SG18]</td>
<td>T2</td>
</tr>
<tr>
<td>Xoodyak†</td>
<td>384</td>
<td>192</td>
<td>192</td>
<td>6/12</td>
<td>$2^{43.8}$</td>
<td>—</td>
<td>—</td>
<td>T2</td>
</tr>
</tbody>
</table>

†: The attack on Xoodyak works in nonce-reuse setting.

T1: Cube-attack-like attack
T2: Conditional cube attack
DF: Degrees of freedom

2 Preliminaries

2.1 Notations

Some notations for Keccak variants are as follows:

- $S_0$ the initial state of Keccak-p permutation,
- $S_{i-1,\theta}$ the internal state after $\theta$ in $i$-th round of Keccak-p, $i \geq 1$,
- $S_{i-1,\pi}$ the internal state after $\pi$ in $i$-th round, $i \geq 1$,
- $S_i$ the output state of the $i$-th round, $i \geq 1$.

Thus the internal states of $i$-th round Keccak are as follows:

$$S_{i-1} \mathrel{\xrightarrow{\theta}} S_{i-1,\theta} \mathrel{\xrightarrow{\rho}} S_{i-1,\rho} \mathrel{\xrightarrow{\pi}} S_{i-1,\pi} \mathrel{\xrightarrow{\chi}} S_{i-1,\chi} \rightarrow S_i. \quad (1)$$
Similarly, the internal states of i-th round Xoodoo are as follows:

\[ S_{i-1} \xrightarrow{\theta} S_{i-1,\theta} \xrightarrow{\rho} S_{i-1,\rho} \xrightarrow{\pi} S_{i-1,\pi} \xrightarrow{\chi} S_{i-1,\chi} \xrightarrow{\iota} S_i. \]  

\( (\ast, j, k) \) the index of row,  
\( (i, \ast, k) \) the index of column,  
\( (i, j, \ast) \) the index of lane,  
\( (i, j, k) \) the index of bit,  
\( A[i][j] \) the lane indexed by \((i, j, \ast)\) of state \(A\),  
\( A[i][j][k] \) the bit indexed by \((i, j, k)\) of state \(A\).

2.2 The Keccak-p permutations

The Keccak-p permutations are derived from the Keccak-f permutations and have a tunable number of rounds. A Keccak-p permutation is defined by its width \(b = 25 \times 2^l\), with \(b \in \{25, 50, 100, 200, 400, 800, 1600\}\), and its number of rounds \(n_r\), denoted as \(\text{Keccak-p}[b, n_r]\). The round function \(R\) consists of five operations, denoted as \(R = \iota \circ \chi \circ \pi \circ \rho \circ \theta\), and the details are as follows:

\[
\theta : A[x][y] = A[x][y] \oplus \sum_{j=0}^{4} (A[x-1][j] \oplus (A[x+1][j] \gg 1)).
\]

\[
\rho : A[x][y] = A[x][y] \gg \rho[x,y].
\]

\[
\pi : A[y][2x+3y] = A[x][y].
\]

\[
\chi : A[x][y] = A[x][y] \oplus ((\neg A[x+1][y]) \wedge A[x+2][y].
\]

\[
\iota : A[0][0] = A[0][0] \oplus RC.
\]

Keccak-p[b, n_r] works on a state \(A\) of size \(b\), which can be represented as \(5 \times 5 \times \frac{b}{25}\)-bit lanes, as depicted in Figure 1. \(A[i][j]\) with \(i\) for the index of column and \(j\) for the index of row. In what follows, indexes \(i\) and \(j\) are in set \(\{0, 1, 2, 3, 4\}\) and they are working modulo 5 without other specification.

2.3 Ketje

Ketje is a submission by the Keccak team. It is a sponge-like construction. In Ketje v1, two instances are proposed, Ketje Sr and Jr with
400-bit and 200-bit state sizes, respectively. In the latest Ketje v2, another two instances Ketje Minor and Major are added to the family, with 800-bit and 1600-bit state sizes, respectively. Ketje Sr is the primary recommendation. The four concrete instances of Ketje v2 are shown in Table 2. In the following, we give a brief overview about Ketje v2. For a complete description, we refer to the design document [BDP+16].

The structure of Ketje is an authenticated encryption mode MonkeyWrap, which is based on MonkeyDuplex [BDPA11]. It consists of four parts: initialization, processing associated data, processing the plaintext, finalization. Figure 2 illustrates the scheme of Ketje v2, where the finalization is omitted. In Ketje v2, the twisted permutations, Keccak-p*[b]=π◦Keccak-p[b]◦π−1, are introduced to effectively re-order the bits in the state. π−1: A[x + 3y][x] = A[x][y] is the inverse of π, shown in Figure 3. Specially, f0=Keccak-p*[b, 12] and f1=Keccak-p*[b, 1].

Table 2: Four Instances in Ketje v2

<table>
<thead>
<tr>
<th>Name</th>
<th>f</th>
<th>ρ</th>
<th>Main use case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ketje Jr</td>
<td>Keccak-p*[200]</td>
<td>16</td>
<td>lightweight</td>
</tr>
<tr>
<td>Ketje Sr</td>
<td>Keccak-p*[400]</td>
<td>32</td>
<td>lightweight</td>
</tr>
<tr>
<td>Ketje Minor</td>
<td>Keccak-p*[800]</td>
<td>128</td>
<td>lightweight</td>
</tr>
<tr>
<td>Ketje Major</td>
<td>Keccak-p*[1600]</td>
<td>256</td>
<td>high performance</td>
</tr>
</tbody>
</table>

2.4 Xoodoo

At ToSC 2018, Daemen et al. [DHAK18] proposed a 384-bit permutation, called Xoodoo, whose design is similar to Keccak. The state is presented as a three-dimension matrix of bits A[4][3][w], where w = 32. The round function of
Xoodoo has five operations, denoted as $R = \rho_{\text{east}} \circ \chi \circ \iota \circ \rho_{\text{west}} \circ \theta$, and the details are as follows:

\[ \theta : A[x][y][z] = A[x][y][z] \oplus \sum_{j=0}^{2} (A[x-1][j][z-5] \oplus A[x-1][j][z-14]). \]
\[ \rho_{\text{west}} : A[x][1][z] = A[x-1][1][z], \ A[x][2][z] = A[x][2][z-11]. \]
\[ \iota : A[0][0] = A[0][0] \oplus RC_i. \]
\[ \chi : A[x][y][z] = A[x][y][z] \oplus ((A[x][y+1][z] \oplus 1) \land A[x][y+2][z]). \]
\[ \rho_{\text{east}} : A[x][1][z] = A[x][1][z-1], \ A[x][2][z] = A[x-2][2][z-8]. \]

As pointed out in [DBH+], Xoodoo could be used as an AE scheme in Ketje style.

### 2.5 Xoodyak

As a Round 1 candidate of NIST lightweight cryptography competition, **Xoodyak** [DHAK18] includes an AEAD scheme and a hashing scheme. Both of them are based on Xoodoo permutation.

Xoodyak-AEAD uses a new mode of operation, called Cyclist. As shown in Figure 4, $f$ is the 12-round permutation Xoodoo[12]. Xoodyak-AEAD first absorbs a 128-bit key into the 384 state, then applies Xoodoo[12] to the state; second, absorbs a 128-bit nonce; then, the 192-bit associated data blocks; then absorb 192-bit plaintext blocks and output ciphertext blocks; then go to the finalization phase. Please refer to [DHAK18] for detailed information.

Besides the nonce-respecting security claim, the authors [DHAK18] also clarify the nonce-misuse case that: “Nonce violation and release of unverified decrypted ciphertext have no consequences for integrity and do not put the key in danger for Xoodyak”. In this paper, we assume the nonce is reused and we try to recover the key of reduced Xoodyak. We apply our conditional cube attack in phase of absorbing $M_0$, and select cube variables in $M_0$ with 192-bit degrees of freedom. Our target is to recover the other 192-bit unknown state. Once we recover it, the full 384-bit state is known and we could compute the key inversely.
3 Related Works

3.1 Cube Attack

The cube attack [DS09] was introduced by Dinur and Shamir at EUROCRYPT 2009. It assumes that the output bit of a symmetric cryptographic scheme can be regarded as a polynomial over $GF(2)$.

**Theorem 1.** ([DS09]) Let $f(k_0, ..., k_{n-1}, v_0, ..., v_{m-1})$ be a polynomial over $GF(2)$, where $k_0, ..., k_{n-1}$ are secret variables, and $v_0, ..., v_{m-1}$ are public variables.

For a set $I = \{i_1, i_2, ..., i_|I|\} \subset \{0, ..., m-1\}$, $f(k_0, ..., k_{n-1}, v_0, ..., v_{m-1})$ can be represented uniquely as

$$f(k_0, ..., k_{n-1}, v_0, ..., v_{m-1}) = T_I \cdot P + Q(k_0, ..., k_{n-1}, v_0, ..., v_{m-1}),$$

(3)

where $T_I = v_{i_1} \cdot v_{i_2} \cdot ... \cdot v_{i_|I|}$. The polynomial $P$ only relates to $v_s$’s ($s \notin I$) and the secret variables, and $Q(k_0, ..., k_{n-1}, v_0, ..., v_{m-1})$ misses at least one variable in $T_I$. $T_I$ is called maxterm and $P$ is called superpoly.

Denote by $C_I$ the structure, called cube, consisting of all $2^{|I|}$ different vectors with $v_i, i \in I$ being active (traversing all 0-1 combinations) and non-cube indices $v_s, s \notin I$ being static constants.

Then the sum of $f$ over all values of the cube $C_I$ (cube sum) is

$$\sum_{v_{i_1}, ..., v_{i_|I|} \in C_I} f(k_0, ..., k_{n-1}, v_0, ..., v_{m-1}) = P.$$  

(4)

The basic idea is to find enough $T_I$’s $P$ is linear and not a constant. This enables the key recovery through solving a system of linear equations.

3.2 Conditional Cube Attack

Conditional cube attack [HWX+17] was proposed by Huang et al. at EUROCRYPT 2017 to attack Keccak keyed mode. Then it soon was applied to many
cryptanalysis, such as \cite{LDW17,LBDW17,SG18,BLD18}. Inspired by dynamic cube attack \cite{DS09}, which reduces the degree of output polynomials of cube variables by adding some bit conditions on the initial value (IV), they reduce the degree by appending key bit conditions. The techniques are similar to message modification technique \cite{WY05,WYY05} and conditional differential cryptanalysis \cite{KMN10} which used bit conditions to control differential propagation.

Definition 1. (\cite{HWX17}) Cube variables that have propagation controlled in the first round and are not multiplied with each other after the second round of Keccak are called conditional cube variables. Cube variables that are not multiplied with each other after the first round and are not multiplied with any conditional cube variable after the second round are called ordinary cube variables.

Theorem 2. (\cite{HWX17}) For \((n+2)\)-round Keccak sponge function \((n > 0)\), if there are \(p\) \((0 \leq p < 2^n + 1)\) conditional cube variables \(v_0, ..., v_{p-1}\), and \(q = 2^{n+1} - 2p + 1\) ordinary cube variables, \(u_0, ..., u_{q-1}\) (If \(q = 0\), we set \(p = 2^n + 1\)), the term \(v_0v_1...v_{p-1}u_0...u_{q-1}\) will not appear in the output polynomials of \((n+2)\)-round Keccak sponge function.

Actually, in the previous conditional cube attacks \cite{HWX17,LBDW17,SGSL18}, they only use the special case of the above theorem when \(p = 1\). We describe it as a corollary for clearness.

Corollary 1. For \((n + 2)\)-round Keccak sponge function \((n > 0)\), if there is one conditional cube variable \(v_0\), and \(q = 2^{n+1} - 1\) ordinary cube variables, \(u_0, ..., u_{q-1}\), the term \(v_0u_0...u_{q-1}\) will not appear in the output polynomials of \((n + 2)\)-round Keccak sponge function.

3.3 New Conditional Cube Attack

Recently, Li et al. \cite{LDB19} proposed new conditional cube attacks on Keccak keyed mode with extremely small degrees of freedom. In Huang et al.’s attack \cite{HWX17}, all cube variables must not be multiplied together in the first round. However, Li et al. \cite{LDB19} relaxed this constraint by introducing the so-called kernel quadratic term.

Definition 2. (\cite{LDB19}) Suppose all the \((q+2)\) cube variables are \(v_0, v_1, u_0, ..., u_{q-1}\), and constraints are as follows:

- After the first round, \(v_0v_1\) is the only quadratic term;
- In the second round, if the bit conditions are satisfied, \(v_0v_1\) does not multiply with any of \(u_0, ..., u_{q-1}\), i.e. no cubic term occurs.
- In the second round, if the bit conditions are not satisfied, \(v_0v_1\) multiplies with some of \(u_0, ..., u_{q-1}\), i.e. some cubic terms like \(v_0v_1u_i\) \((i = 0, ..., q - 1)\) occur.

Then \(v_0v_1\) is called kernel quadratic term. The remaining cube variables except \(v_0\) and \(v_1\), i.e. \(u_0, ..., u_{q-1}\), are called ordinary cube variables.
Then the following corollary is deduced.

**Corollary 2.** [LDB+19] For \((n+2)\)-round Keccak sponge function \((n > 0)\), if there is one kernel quadratic term \(v_0v_1\), and \(q = 2^{n+1} - 1\) ordinary cube variables, \(u_0, u_1, \ldots, u_{q-1}\), the term \(v_0v_1u_0u_1\ldots u_{q-1}\) will not appear in the output polynomials of \((n+2)\)-round Keccak sponge function under certain bit conditions.

So the distinguisher is approached as follows:

- under right bit conditions, the degree of output polynomials of \(n+2\) rounds is no more than \(2^{n+1}\);
- under the wrong bit conditions, the degree of output polynomials of \(n+2\) rounds is \(q + 2 = 2^{n+1} + 1\).

According to Definition 2 in Li et al.'s [LDB+19] new conditional cube attack, we only care that the kernel quadratic term (i.e. \(v_0v_1\)) should not be multiplied with ordinary cube variables in the second round. So they devised the so-called 6-2-2 pattern to reduce the diffusion of \(v_0v_1\). Therefore, they could get more degrees of freedom to find ordinary cube variables.

Concretely, in the initial state \(S_0\), Li et al. selected \(v_0\) and \(v_1\), such that \(S_0[2][0][0] = S_0[2][1][0] = S_0[3][0][34] = S_0[3][1][34] = v_0\), are in CP-kernel, \(S_0[0][1][60] = S_0[1][1][1] = e_1\) are not in CP-kernel. Then, before the first \(\chi\), the distribution of \(v_0\) and \(v_1\) is shown in Figure 5. Moreover, after \(\chi\), kernel quadratic term \(v_0v_1\) is also in CP-kernel. Thus, the second \(\theta\) becomes the identity for \(v_0v_1\). Therefore, before the second \(\chi\), only two bit positions are related to \(v_0v_1\). So, it is called 6-2-2 pattern. Li et al. also gave a method to derive the 6-2-2 pattern. For more details, please refer to [LDB+19].

![Fig. 5: 6-2-2 Pattern: Generation of Kernel Quadratic Terms in the First \(\chi\)](image)
3.4 MILP Model of Conditional Cube Attack

At ASIACRYPT 2017, Li et al. [LBDW17] for the first time applied MILP method to cube attacks on keyed Keccak. Later, the MILP model was improved by [SGSL18], and also applied to cube-attack-like method by Bi et al. [BDL+18] and Song et al. [SG18]. In the previous MILP model of conditional cube attack, no cube variables multiply with each other in the first round, and the conditional cube variable of degree one is considered not to multiply with any ordinary cube variables in the second round. To limit the diffusion of conditional cube variables, conditions are added in the first round. Actually, as described in Corollary 1, \((2^n+1-1)\) ordinary cube variables are needed to perform a \((n+2)\)-round attack. To obtain enough ordinary cube variables for the attack, the objective of Li et al.’s MILP model [LBDW17] at ASIACRYPT 2017 was to maximize the number of ordinary cube variables. To reduce the attack complexity further, Song et al. [SGSL18] proposed a new MILP model to minimize the number of conditions at ASIACRYPT 2018.
4 8-2-2 Pattern

Based on Li et al.'s [LDB+19] new discovery, we introduce the 8-2-2 pattern instead of the 6-2-2 pattern to attack targets with fewer degrees of freedom, namely Ketje Jr, Xoodoo-AE and Xoodyak-AEAD.

According to the new conditional cube attack given in Sect. 3.3, we only care that the kernel quadratic term (i.e. $v_0v_1$) should not be multiplied with ordinary cube variables in the second round. Li et al.’s 6-2-2 pattern allows $v_1$ to be not in the CP-kernel. Thus, it will distribute to 22 bit positions before the first $\chi$ as shown in Figure 5. This reduces the degrees of freedom to select ordinary cube variables $u_i$, since according to Definition 2, $u_i$ has to be prevent from multiplying with $v_1$ in the first round.

Hence, we introduce the so-called 8-2-2 pattern, i.e. both $v_0$ and $v_1$ are set in CP-kernel in the initial state. For example, in the initial state $S_0$ of Ketje Jr, $v_0$ occupies 4 bits of $S_0$, i.e. $A[4][3][0]=A[4][4][0]=A[2][3][1]=A[2][4][1]=v_0$, and $v_1$ also occupies 4 bits, i.e. $A[0][4][6]=A[0][3][6]=A[3][4][0]=A[3][3][0]=v_1$. Before the first $\chi$ operation, $v_0$ only appears in the 4 bits, while $v_1$ also appears in 4 bits. Similar to Li et al.’s work [LDB+19], only 2 S-boxes are related to $v_0$ and $v_1$ simultaneously and the quadratic term $v_0v_1$ is generated in two bit positions after $\chi$ operation, as shown in Figure 7. Moreover, the two bits containing $v_0v_1$ in $S_1$ are also in CP-kernel. Similar to Li et al. [LDB+19], we also give the method to find the 8-2-2 pattern.

Table 3: Related Indexes of Bits Containing $v_0$, $v_1$ and $v_0v_1$

<table>
<thead>
<tr>
<th>Index</th>
<th>$v_0$</th>
<th>$v_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0, 0$</td>
<td>$(x + 3y_0 + 1, x + 1, z)$</td>
<td>$(x + 3y_0 + 2, x + 2, z)$</td>
</tr>
<tr>
<td>$S_0, 0$</td>
<td>$(x + 3y_0 + 1, x + 1, z)$</td>
<td>$(x + 3y_0 + 2, x + 2, z)$</td>
</tr>
<tr>
<td>$S_0, 0$</td>
<td>$(x + 3y_0 + 1, x + 1, z - \rho[x + 3y_0 + 1, x + 1])$</td>
<td>$(x + 3y_0 + 2, x + 2, z - \rho[x + 3y_0 + 2, x + 2])$</td>
</tr>
<tr>
<td>$S_0, 0$</td>
<td>$(x + 3y_0 + 1, x + 1, z - \rho[x + 3y_0 + 1, x + 1])$</td>
<td>$(x + 3y_0 + 2, x + 2, z - \rho[x + 3y_0 + 2, x + 2])$</td>
</tr>
</tbody>
</table>

The method to find 8-2-2 pattern. The most important constraint is that $v_0v_1$ has to be in a CP-kernel in $S_1$. So we start with setting $v_0v_1$ in a CP-kernel. Denote the two bits containing $v_0v_1$ as $(x, y_0, z)$, $(x, y_1, z)$. According to the expression of $\chi$, $v_0v_1$ in $S_1[x][y_0][z]$ is generated by multiplying $v_0$ in $S_0, 0[x + 1][y_0][z]$ and $v_1$ in $S_0, 0[x + 2][y_0][z]$, or $v_0$ in $S_0, 0[x + 2][y_0][z]$ and $v_1$.
in $S_0[x+1][y_0][z]$. The same happens to $v_0v_1$ in $S_1[x][y_1][z]$. So there will be 4 cases to determine the bit positions for $v_0$ and $v_1$ in reverse. For example, in Figure 6, $v_0v_1$ appears in $S_1[1][2][z]$ by multiplication of $v_0$ in $S_0[x][2][2][z]$ and $v_1$ in $S_0[x][3][2][z]$, and similarly $v_0v_1$ appears in $S_1[1][4][z]$ by multiplication of $v_1$ in $S_0[x][2][4][z]$ and $v_0$ in $S_0[x][3][4][z]$.

Under one of the 4 cases, Table 3 describes the bit positions of $v_0v_1$, $v_0$, and $v_1$ inversely from $S_1$ to $S_0$, while the other cases are similar. In Table 3, in order to reduce the diffusion of $v_0$ and $v_1$, the 4 bit positions containing $v_0$ and $v_1$ in $S_0$ are set in CP-kernel, where $x_1, x_2 \neq x + 1$, $x_3, x_4 \neq x + 2$. At last, all the 8 bits in $S_0$ should be selected in free space for ordinary cube variables. Accordingly, we can determine 8-2-2 patterns.

5 Applications to Ketje Jr, Xoodoo-AE and Xoodyak-AEAD

5.1 5-Round Attack against Ketje Jr

For Ketje Jr with 200-bit state, the recommended key size is 96 bits, while the shortest padding occupies 18 bits. A 5-round attack can be performed with 17 cube variables.

According to the new conditional cube attack illustrated in Sect. 3.3, we find a 8-2-2 pattern at first, and then search for the minimal number of key bit conditions and the 15 ordinary cube variables satisfying the corresponding rules by previous MILP models \cite{LBDW17,SGSL18,LDB+19}.

![Fig. 8: The Initial State of Ketje Jr v1](image)

5-round Attack on Ketje Jr v1. In the procedure of attack on 5-round Ketje Jr v1, we select $v_0$ and $v_1$ following the 8-2-2 pattern. As shown in Figure 8, the 96-bit key is located at the red parts, while the padding part is shown in blue. $v_0$ is
The white bits represent free space to be selected as ordinary cube variables.

The cube variables and bit conditions to attack 5-round initialization phase of Ketje Jr v1 are given in Table 4.

Table 4: Parameters set for attack on 5-round Ketje Jr v1

<table>
<thead>
<tr>
<th>kernel quadratic term</th>
<th>Bit Conditions</th>
<th>Ordinary Cube Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A[3][4][4] = k_{12} + k_{27} + k_{62} + k_7 + k_{92}$</td>
<td>$A[4][2][5] = k_{14} + k_{29} + k_{54} + k_{62} + k_{94} + n_{30} + n_{45} + n_{65} + 1$</td>
</tr>
<tr>
<td></td>
<td>$A[1][3][1] = k_1 + k_{26} + k_{41} + k_{66} + k_{81} + n_{50} + n_{57}$</td>
<td>$A[4][2][4]=u_{10},A[4][3][4]=u_{11},A[4][4][4]=u_{12},A[1][3][0]=u_3$,</td>
</tr>
<tr>
<td></td>
<td>$A[1][3][3] = k_3 + k_{28} + k_{43} + k_{68} + k_{83} + n_{52} + n_{59}$</td>
<td>$A[4][2][5] = k_{14} + k_{29} + k_{54} + k_{62} + k_{94} + n_{30} + n_{45} + n_{65} + 1$</td>
</tr>
<tr>
<td></td>
<td>$A[4][2][5] = k_{14} + k_{29} + k_{54} + k_{62} + k_{94} + n_{30} + n_{45} + n_{65} + 1$</td>
<td>$A[4][4][3]=u_{13},A[4][4][3]=u_{12},A[4][4][2]=u_{10} + u_{11},A[4][2][4]=u_{12}$,</td>
</tr>
</tbody>
</table>

Note that the number of the key bits to be guessed and assigned is 6. If the key guessing is right, it is expected to output zero cube sums. The time complexity to recover the 6-bit key is $2^6 \times 2^{17}$. According to the property of the permutation, it is totally symmetric in z-axis. Thus we can obtain corresponding parameter sets with any i-bit rotation $(0 \leq i < 8)$ in z-axis. Therefore, the related key bits rotated by i bits can be recovered. Besides, according to Sect. 4 we can use other kernel quadratic terms not only different in z-axis to recover more key bits.

Totally, 12 iterations could recover the 72-bit key. Then guess 96-72=24 bits to determine the full key. The time complexity is $12 \times 2^{26} \times 2^{17} = 2^{26.58}$.

To support our theory, we have run more than 1000 experiments and obtained correct key recovery of 5-round Ketje Jr v1 with 100 percent success rate. For saving space, we give an example here for intuition, in which the key is generated randomly and all the controllable nonce bits are set to zero. The program is run in Visual Studio 2012 with x64 platform Release. The time is less than 8 seconds for recovery of 6 key bits using one CPU core (Intel i7 3.6GHz), and parallelism can reduce time. Using the test code that we provided, one can verify it easily.
We calculate the cube sum at lane (0,0), (1,0). As the probability, for which the cube sum on these two lanes is zero, is $2^{-16}$ for a random function. Therefore, if the 17-dimension cube sums of 5-round output is zero, we declare that the key guess is correct with high probability. Actually, we also calculate other lanes’ cube sum, and all of them turn out to be zero. The test code is given in https://github.com/alicebobb/aabb/tree/alicebobb-patch-1.

96-bit key $K$:
101000001101011001110100110111000111001000011101110110111001011011011111100100111010010110001010101

The correct value for the guessed key bits in Table 4 is 001110.

To see the full key guess results, we put them in Appendix A.

5-round Attack on Ketje Jr v2 As Figure 9 shows us, after $\pi^{-1}$ transformation, the 96-bit key is located at the red parts, while the padding part is shown in blue. And $v_0$ is set in CP-kernel as $A[4][2][0]=A[4][0][0]=A[0][1][3]=A[0][4][3]=v_0$ in black, and $v_1$ is located at 4 grey bits, i.e. $A[3][1][0]=A[3][4][0]=A[1][2][5]=A[1][4][5]=v_1$. The white bits represent free space to be selected as ordinary cube variables.

![Fig. 9: The Initial State of Ketje Jr v2](image-url)

The number of bit conditions related to key is 7, hence, we have to guess a 7-bit key to assign conditions. The time complexity to recover the 7-bit key is
Using the test code that we provided, one can verify it easily. The time is less than 16 seconds for recovery of 5-round key guess is correct with high probability. Actually, we also calculate complexity is $2^{7} \times 2^{17}$. According to the property of the permutation, it is totally symmetric in $z$-axis. Thus we can obtain corresponding parameter sets with any $i$-bit rotation ($0 \leq i < 8$) in $z$-axis. Therefore, the guessed key bits rotated by $i$ bits can be recovered. 11 iterations of the above process could recover a 77-bit key and the remaining 19 key bits are recovered by exhaustive search. The total time complexity is $11 \times 2^{7} \times 2^{17} = 2^{27.46}$.

Similarly, we have run more than 1000 experiments and obtained correct key recovery of 5-round KETJE JR v2 with 100 percent success rate. We also give an example here for intuition, in which the key is generated randomly and all the controllable nonce bits are set to zero. The program is run in Visual Studio 2012 with x64 platform Release. The time is less than 16 seconds for recovery of 7 key bits using one CPU core (Intel i7 3.6GHz), and parallelism can reduce time of the above process could recover a 77-bit key and the remaining 19 key bits are recovered by exhaustive search. The total time complexity is $11 \times 2^{7} \times 2^{17} = 2^{27.46}$.

We calculate the cube sum at lane (0, 0), (1, 1). The probability, for which the cube sum on these two lanes the cube sum is zero, is $2^{-16}$ for a random function. Therefore, if the 17-dimension cube sums of 5-round output is zero, we declare that the key guess is correct with high probability. Actually, we also calculate other lanes’ cube sum, and all of them turn out to be zero. The test code is given in https://github.com/alicebobb/aabb/tree/alicebobb-patch-1.

96-bit key $K$:

- 1001011000010011000101001010110101101101011011001
- 11001001110110100111111010110110101101101111011001
- 11001001110110100111111010110110101101101111011001

The correct value for the guessed key bits in Table 5 is 0000110.
guessed value: 0111010,   cube sums: 0xc, 0xca
guessed value: 1111010,   cube sums: 0xa4, 0xe5
guessed value: 0000110,   cube sums: 0x0, 0x0
guessed value: 1000110,   cube sums: 0x2e, 0xea
guessed value: 0100110,   cube sums: 0xc8, 0x7f

To see the full key guess results, we put them in Appendix A.

5.2 6-Round Attack against Xoodoo-AE

As pointed out in [DBH̊+], Xoodoo [DHAK18] could be used as an AE scheme in Ketje style. We assume that the Xoodoo-AE has a 128-bit key and follows the Ketje’s packing. As shown in Figure 10, the 128-bit key $K$ is located at the 5 red lanes, and the padding parts are blue. The white part represents nonce bits. The operations $\theta$ and $\chi$ of Xoodoo are very similar to those of Keccak-$p$ and $\rho_{west}$ just reorder the state bits which is similar to $\rho$ and $\pi$. So it is easy to modify the attack strategy of Keccak-$p$ to Xoodoo.

However, Xoodoo’s state is $3\times4$, not Keccak-$p$’s $5\times5$ state, which means columns in Xoodoo are shorter than those in Keccak-$p$. Another different feature is the S-box, which is applied to every 3-bit column. Moreover, the first row and part of the second row are occupied by the key and padding bits in Xoodoo, which means, there are fewer degrees of freedom for us to search ordinary cube variables in CP-kernel. All those features will affect our decisions for choosing conditional cube variables and ordinary cube variables. Consequently, we modify the 8-2-2 pattern into a 7-2-2 pattern, which means that $v_0$ appears in 4 bits and $v_1$ appears in 3 bits in the initial state.

![Fig. 10: The Initial State of Xoodoo-AE](image)

Assume that $v_0v_1$ is the kernel quadratic term. As shown in Figure 10, $v_0$ is set in CP-kernel as $S_0[0][1][16] = S_0[0][2][16] = S_0[1][1][4] = S_0[1][2][4] = v_0$ in
black, and $v_1$ is located at 3 grey bits, i.e. $S_0[1][1][5] = S_0[1][2][5] = S_0[0][2][1] = v_1$. After operation $\theta$ and $\rho_{\text{west}}$,

$S_0,\rho_{\text{west}}[1][1][16] = S_0,\rho_{\text{west}}[1][2][15] = v_0$

as well as $S_0,\rho_{\text{west}}[0][2][27] = S_0,\rho_{\text{west}}[2][1][4] = v_0$, and

$S_0,\rho_{\text{west}}[1][2][16] = S_0,\rho_{\text{west}}[1][0][15] = v_1$

as well as $S_0,\rho_{\text{west}}[0][2][12] = S_0,\rho_{\text{west}}[1][2][17] = S_0,\rho_{\text{west}}[1][0][6] = S_0,\rho_{\text{west}}[2][1][6]$

$= S_0,\rho_{\text{west}}[1][2][26] = S_0,\rho_{\text{west}}[1][0][15] = S_0,\rho_{\text{west}}[2][1][5] = v_1$.

Table 6: Parameters set for attack on 6-round Xoodoo-AE

<table>
<thead>
<tr>
<th>Kernel quadratic term</th>
<th>Bit Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho[0][1][16]=\rho[0][2][16]=\rho[1][2][4]=\rho[1][1][4]=v_0$, $\rho[1][2][5]=\rho[1][1][5]=\rho[0][2][1]=v_1$</td>
<td></td>
</tr>
</tbody>
</table>

**Ordinary Cube Variables**


The cube variables and bit conditions are shown in Table 6. With 31 ordinary cube variables, there is only one bit condition related to key. We guess the key bit $k_{66} + k_{75} + k_{112}$ to assign the condition. The time complexity of one recovery is $2^1 \times 2^{33}$. According to the property of the permutation, it is totally symmetric in z-axis. Thus we can obtain corresponding parameter sets with any $i$-bit rotation ($0 \leq i < 32$) in z-axis. Therefore, the guessed key bits rotated by $i$ bits can be recovered. 90 iterations could recover a 90-bit key and the remaining key bits could be recovered by exhaustive search. Totally, it consumes $90 \times 2^1 \times 2^{33} + 2^{38} = 2^{40.5}$.

Similarly, we have run more than 1000 experiments and obtained correct key recovery of 6-round Xoodoo-AE with 100 percent success rate, we also give an example here for intuition, in which the key is generated randomly and all the controllable nonce bits are set to zero. The program is run in Visual Studio 2012 with x64 platform Release. The time is about 2 hours for recovery of one key bit.
using one CPU core (Intel i7 3.6GHz), and parallelism can reduce time. Using
the test code that we provided, one can verify it easily.

We just list the cube sum at lane (0, 0), (1, 0), and the probability that the
cube sum for these two lanes cube sum is zero is $2^{-64}$ for a random function.
Therefore, if the 33-dimension cube sums of 5-round output is zero, we declare
that the key guess is correct with high probability. Actually, we also calculate
other lanes’ cube sum, and all of them turn out to be zero. The test code is given

128-bit key $K$:

```
10100000110101100111010011011100011101110101101101
11111100100111010010110001010101000101111010011110010110000101
```

The correct value for the guessed key bit in Table 6 is 1.

- right key: 1
- guessed value: 0, cube sums: 0x9df359, 0xc614c263
- guessed value: 1, cube sums: 0x0, 0x0

### 5.3 6-Round Attack against Xoodyak

We attack reduced XOODYAK-AEAD in nonce-reuse setting. The targeted part
of the XOODYAK-AEAD is the absorbing plaintext phase. As shown in Figure
4 we select cube variables from the 192-bit $M_0$, and we are going to recover
the other 192-bit unknown state, which we denote as 192-bit equivalent key. We
reduced the Xoodoo to 6 rounds, i.e., $M_0$ is processed by 6-round XOODOO,
then the corresponding ciphertexts outputs are used to compute the cube sums.
As shown in Table 11 the 192-bit key $K$ is located at the 6 red lanes. The white
part are tweakable bits and could be selected as cube variables.

![Fig. 11: The Initial State of XOODYAK](image)

Similarly, assume that $v_0v_1$ is the kernel quadratic term. $v_0$ is set in CP-
kernel as $S_0[2][1][0] = S_0[2][2][0] = S_0[3][1][20] = S_0[3][2][20] = v_0$ in black, and
$v_1$ is located at 3 grey bits, i.e. $S_0[3][1][21] = S_0[3][2][21] = S_0[2][1][17] = v_1$. After operation $\theta$ and $\rho_{west}$, $S_0[\rho_{west}[3][1][0] = S_0[\rho_{west}[3][2][31] = v_0$ as well as $S_0[\rho_{west}[2][2][11] = S_0[\rho_{west}[0][1][20] = v_0, S_0[\rho_{west}[3][2][0] = S_0[\rho_{west}[3][0][31] = v_1$ as well as $S_0[\rho_{west}[3][1][17] = S_0[\rho_{west}[0][1][22] = S_0[\rho_{west}[3][2][1] = S_0[\rho_{west}[3][0][22] = S_0[\rho_{west}[0][1][31] = S_0[\rho_{west}[3][2][10] = S_0[\rho_{west}[0][1][21] = v_1$. The white bits represent free space to be selected as ordinary cube variables.

To prevent ordinary cube variables to multiply with other cube variables at the first round, we try to select the ordinary cube variables in CP-kernel. With 31 ordinary cube variables, the number of bit conditions related to key is 6. Then a 6-round attack on XOODDYAK can be performed. Both the cube variables and conditions are listed in Table 7. The 6 equivalent key bits to be guessed are as follows: $k_{66} + k_{89}, k_{75} + k_{99}, k_{8} + k_{99} + k_{113}, k_{33} + k_{42} + k_{79} + k_{161} + k_{170}, k_{49} + k_{58} + k_{95} + k_{172} + k_{186}$ and $k_{42} + k_{56} + k_{79} + k_{170} + k_{184}$. The time complexity to recover the 6-bit key is $2^{6} \times 2^{33}$. Similar to the attack on XOODDO-AE, it is totally symmetric in $z$-axis. Thus we can obtain corresponding parameter sets with any $i$-bit rotation ($0 \leq i < 32$) in $z$-axis. We need 27 iterations of the above procedures to recover $27 \times 6 = 162$ bits key, and leave the other 30 bits key to exhaustive search. The total time complexity is $27 \times 2^{6} \times 2^{33} + 2^{30} = 2^{33.8}$.

Table 7: Parameters set for attack on 6-round XOODDYAK

<table>
<thead>
<tr>
<th>kernel quadratic term</th>
</tr>
</thead>
</table>

**Bit Conditions**

| $A[1][2][1] = k_{33} + k_{42} + k_{79} + k_{161} + k_{170} + n_{106}$ |
| $A[1][2][24] = k_{42} + k_{56} + k_{79} + k_{170} + k_{184} + n_{106}$ |
| $A[3][1][3] = k_{8} + k_{31} + k_{99} + k_{113} + n_{163}$ |
| $A[2][1][2] = k_{66} + k_{89} + n_{25} + n_{130} + n_{153} + n_{167}$ |
| $A[3][2][16] = k_{75} + k_{89} + n_{25} + n_{153} + n_{167}$ |
| $A[1][2][20] = k_{49} + k_{85} + n_{96} + k_{177} + k_{186}$ |

**Ordinary Cube Variables**

| $A[3][1][20] = u_{0}, A[3][5][2][26] = u_{0}, A[3][3][1][29] = u_{1}, A[3][2][2][29] = u_{1}, A[0][0][2][7] = u_{2}$ |
Similarly, we also have run some experiments and obtained correct key recovery of 6-round \textsc{Xoodyak} with 100 percent success rate. In our verification experiments the key is generated randomly and all the controllable nonce bits are set to zero. The program is run in Visual Studio 2012 with x64 platform Release. Recovery a 6-bit key need about 60 hours using one CPU core (Intel i7 3.6GHz), and parallelism can reduce time. Using the test code that we provided, one can verify it easily.

We calculate the cube sum at lane $(0, 0), (1, 0), (2, 0), (3, 0), (0, 1), (1, 1)$, and the probability, for which the cube sum on these six lanes is zero, is $2^{-192}$ for a random function. Therefore, if the 33-dimension cube sums of 6-round output is zero, we declare that the key guess is correct with high probability. Actually, we also calculate other lanes’ cube sum, and all of them turn out to be zero. The test code is given in \url{https://github.com/alicebob/aabb/tree/alicebob2-patch-1}.

192-bit key $K$:

```
101000001101011001100011101100011101110110110110110111101110010
111110011011010001101010101001001101010010011011011011010111101
1101101001000100010111010100010110101010101011011110110101011001
1101101001000100010111010100010110101010101010101010010011010110
1101101001000100010111010100010110101010101010101010010011010110
1101101001000100010111010100010110101010101010101010010011010110
```

The 21-bit key in Table 6 is 001000.

```
right key: 001000
guessed value: 001000, cube sums: 0x0, 0x0, 0x0, 0x0, 0x0
```

```
guessed value: 000100, cube sums: 0xd2c47032, 0x456c766f, 0xd74569ed, 0x4a96a204, 0xd6f503b8, 0x6f6e9541
```

```
guessed value: 010000, cube sums: 0xa6dd98e3, 0xdf935915, 0xafbc25d9, 0x939c401d, 0x95e04808, 0x3caecd13
```

6 Conclusion

In this paper, we give several practical key-recovery attacks on 5-round initialization of \textsc{Ketje Jr} v1 and v2, 6-round \textsc{Xoodoo}-AE in nonce-respecting setting and 6-round \textsc{Xoodyak} in nonce-reuse setting, whose time complexities are $2^{26.6}$, $2^{27.5}$, $2^{43.5}$ and $2^{43.8}$ with negligible memory cost. All the attacks are practically implemented.
References


HWX+17. Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional Cube Attack on Reduced-Round Keccak Sponge Function. In Advances in Cryptology - EUROCRYPT 2017 - 36th


Table 8: Guess key and cube sum for 5-round Ketje Jr v1

<table>
<thead>
<tr>
<th>Guess key</th>
<th>Cube sum</th>
<th>Guess key</th>
<th>Cube sum</th>
<th>Guess key</th>
<th>Cube sum</th>
<th>Guess key</th>
<th>Cube sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000 0x91,0xe5</td>
<td>000010 0xc4,0xefe</td>
<td>000001 0xec9,0xc9</td>
<td>000011 0xf4,0xfa</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000 0x9d,0x9b</td>
<td>100010 0x9a,0xda</td>
<td>100001 0x28,0x71</td>
<td>100011 0xf7,0x19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010000 0x9a,0xa5</td>
<td>010010 0xf8,0x19</td>
<td>010001 0x9c,0x54</td>
<td>010011 0xdc,0x4d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110000 0x9c,0x69</td>
<td>110010 0xc2,0x43</td>
<td>110001 0x8c,0x38</td>
<td>110011 0x27,0x53</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001000 0xf8,0xf1</td>
<td>001010 0x85,0x3c</td>
<td>001001 0xf8,0x95</td>
<td>001011 0xeex,0x33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101000 0x1,0xa7</td>
<td>101010 0x2e,0x1c</td>
<td>101001 0x5b,0x2d</td>
<td>101011 0x21,0x81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011000 0x15,0xbb</td>
<td>011010 0x8d,0x7b</td>
<td>011001 0x31,0x46</td>
<td>011011 0xdf,0xae</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111000 0x48,0x4e</td>
<td>111010 0xb2,0x96</td>
<td>111001 0xfb,0x2f</td>
<td>111011 0xed,0xaf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000100 0xa7,0xd6</td>
<td>000110 0xcb,0xb9</td>
<td>000101 0x81,0xc6</td>
<td>000111 0x3b,0x4e</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100100 0x45,0x60</td>
<td>100110 0xf0,0x81</td>
<td>100101 0xe,0xd</td>
<td>100111 0x37,0x3a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010100 0x15,0x38</td>
<td>010110 0x88,0x52</td>
<td>010101 0xf9,0xda</td>
<td>010111 0xecf,0xa1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110100 0x3,0x9c</td>
<td>110110 0xd5,0x44</td>
<td>110101 0x40,0x1a</td>
<td>110111 0x43,0xba</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001100 0x38,0x1e</td>
<td>001110 0x0,0x0</td>
<td>001101 0x45,0x67</td>
<td>001111 0x4b,0x31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101100 0x37,0xc9</td>
<td>101110 0xc5,0xa5</td>
<td>101101 0xc4,0x18</td>
<td>101111 0x4,0x2f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011100 0xda,0xa7</td>
<td>011110 0x1b,0x49</td>
<td>011101 0xda9,0x21</td>
<td>011111 0x23,0x58</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111100 0x0,0xce</td>
<td>111110 0xba,0x49</td>
<td>111101 0x8b,0x75</td>
<td>111111 0x44,0x8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A Experimental results on Ketje Jr v1, Ketje Jr v2
Table 9: Guess key and cube sum for 5-round Ketje Jr v2

<table>
<thead>
<tr>
<th>Guess key</th>
<th>Cube sum</th>
<th>Guess key</th>
<th>Cube sum</th>
<th>Guess key</th>
<th>Cube sum</th>
<th>Guess key</th>
<th>Cube sum</th>
<th>Guess key</th>
<th>Cube sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>0x55,0xfd</td>
<td>0000010</td>
<td>0x7b,0xa5</td>
<td>0000001</td>
<td>0xc0,0x85</td>
<td>0000011</td>
<td>0x6b,0x9b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>0x19,0x48</td>
<td>1000010</td>
<td>0x7f,0x4b</td>
<td>1000001</td>
<td>0xa0,0xfb</td>
<td>1000011</td>
<td>0x2d,0x5f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100000</td>
<td>0x58,0x52</td>
<td>0100010</td>
<td>0x2a,0x4c</td>
<td>0100001</td>
<td>0x87,0x19</td>
<td>0100011</td>
<td>0x13,0x6d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100000</td>
<td>0x86,0x20</td>
<td>1100010</td>
<td>0x55,0x32</td>
<td>1100001</td>
<td>0x40,0x60</td>
<td>1100011</td>
<td>0x19,0x33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010000</td>
<td>0x8c,0x9f</td>
<td>0010010</td>
<td>0x48,0xca</td>
<td>0010001</td>
<td>0x61,0x8a</td>
<td>0010011</td>
<td>0x0c,0x61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010000</td>
<td>0x8d,0x62</td>
<td>1010010</td>
<td>0xb0,0x8b</td>
<td>1010001</td>
<td>0x1a,0x23</td>
<td>1010011</td>
<td>0x0f,0x2d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110000</td>
<td>0x78,0x17</td>
<td>0110010</td>
<td>0x8f,0xf2</td>
<td>0110001</td>
<td>0x8b,0x29</td>
<td>0110011</td>
<td>0x0c,0x6c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110000</td>
<td>0x1c,0x6c</td>
<td>1110010</td>
<td>0xcb,0x33</td>
<td>1110001</td>
<td>0x5e,0x9a</td>
<td>1110011</td>
<td>0x1d,0x73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001000</td>
<td>0xa9,0x73</td>
<td>0001010</td>
<td>0x4e,0x82</td>
<td>0001001</td>
<td>0x00,0x83</td>
<td>0001011</td>
<td>0x59,0x92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001000</td>
<td>0x25,0x39</td>
<td>1001010</td>
<td>0x38,0xcb</td>
<td>1001001</td>
<td>0x5d,0x29</td>
<td>1001011</td>
<td>0x57,0xff</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101000</td>
<td>0x43,0x6c</td>
<td>0101010</td>
<td>0x0f,0x88</td>
<td>0101001</td>
<td>0x78,0x3b</td>
<td>0101011</td>
<td>0x52,0xb4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1101000</td>
<td>0x55,0x9a</td>
<td>1101010</td>
<td>0x50,0x15</td>
<td>1101001</td>
<td>0x64,0x62</td>
<td>1101011</td>
<td>0x51,0x6d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011000</td>
<td>0x29,0x1d</td>
<td>0011010</td>
<td>0x7d,0x3e</td>
<td>0011001</td>
<td>0x85,0x1f</td>
<td>0011011</td>
<td>0x0c,0x6d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011000</td>
<td>0x7c,0x6c</td>
<td>1011010</td>
<td>0x9e,0x52</td>
<td>1011001</td>
<td>0x2c,0x27</td>
<td>1011011</td>
<td>0x64,0x6d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111000</td>
<td>0x0c,0x0a</td>
<td>0111010</td>
<td>0xcb,0x0a</td>
<td>0111001</td>
<td>0x30,0x71</td>
<td>0111011</td>
<td>0x0c,0x6d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111000</td>
<td>0x77,0x78</td>
<td>1111010</td>
<td>0xa4,0x6d</td>
<td>1111001</td>
<td>0x22,0x29</td>
<td>1111011</td>
<td>0xd1,0x77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000100</td>
<td>0xc2,0x9b</td>
<td>0000110</td>
<td>0x00,0x00</td>
<td>0000101</td>
<td>0x7f,0x23</td>
<td>0000111</td>
<td>0x1a,0x37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000100</td>
<td>0x23,0x6d</td>
<td>1000110</td>
<td>0x2e,0xea</td>
<td>1000101</td>
<td>0x2d,0x89</td>
<td>1000111</td>
<td>0xe4,0x8b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100100</td>
<td>0x65,0x33</td>
<td>0100110</td>
<td>0x8c,0x7f</td>
<td>0100101</td>
<td>0xc3,0x37</td>
<td>0100111</td>
<td>0x32,0xc6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100100</td>
<td>0x90,0x87</td>
<td>1100110</td>
<td>0xcb,0x67</td>
<td>1100101</td>
<td>0x4d,0x8b</td>
<td>1100111</td>
<td>0xa9,0x9c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010100</td>
<td>0x5d,0x0d</td>
<td>0010110</td>
<td>0x6e,0xad</td>
<td>0010101</td>
<td>0x3f,0xf1</td>
<td>0010111</td>
<td>0x97,0xd2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010100</td>
<td>0x60,0xac</td>
<td>1010110</td>
<td>0xb6,0xe4</td>
<td>1010101</td>
<td>0xcf,0x9e</td>
<td>1010111</td>
<td>0x17,0x25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110100</td>
<td>0x54,0x4d</td>
<td>0110110</td>
<td>0x47,0x11</td>
<td>0110101</td>
<td>0xca,0x59</td>
<td>0110111</td>
<td>0x1c,0xc4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110100</td>
<td>0x8e,0x6b</td>
<td>1110110</td>
<td>0xc6,0x50</td>
<td>1110101</td>
<td>0xed,0xb7</td>
<td>1110111</td>
<td>0x1d,0x6f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001100</td>
<td>0x51,0x72</td>
<td>0001110</td>
<td>0x44,0xed</td>
<td>0001101</td>
<td>0x5c,0x41</td>
<td>0001111</td>
<td>0x02,0xcf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001100</td>
<td>0xe5,0x2a</td>
<td>1001110</td>
<td>0x2d,0xe5</td>
<td>1001101</td>
<td>0x2a,0xf4</td>
<td>1001111</td>
<td>0x11,0x3a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101100</td>
<td>0x28,0x68</td>
<td>0101110</td>
<td>0x15,0x41</td>
<td>0101101</td>
<td>0xea,0x9e</td>
<td>0101111</td>
<td>0xda,0x8e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1101100</td>
<td>0x2a,0x1c</td>
<td>1101110</td>
<td>0x11,0x56</td>
<td>1101101</td>
<td>0x43,0xce</td>
<td>1101111</td>
<td>0x9a,0x79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011100</td>
<td>0xe7,0xb9</td>
<td>0011110</td>
<td>0x2c,0x99</td>
<td>0011101</td>
<td>0x8f,0x26</td>
<td>0011111</td>
<td>0x88,0x6d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011100</td>
<td>0xef,0x89</td>
<td>1011110</td>
<td>0x66,0x67</td>
<td>1011101</td>
<td>0x60,0xb9</td>
<td>1011111</td>
<td>0xa2,0x9c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111100</td>
<td>0x56,0xda</td>
<td>0111110</td>
<td>0x8a,0x43</td>
<td>0111101</td>
<td>0x6d,0x3f</td>
<td>0111111</td>
<td>0x9c,0x85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111100</td>
<td>0xc6,0x4b</td>
<td>1111110</td>
<td>0xc2,0x21</td>
<td>1111101</td>
<td>0x8b,0xff</td>
<td>1111111</td>
<td>0x20,0x6c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>