# State-Recovery Attacks on Modified KETJE JR

Thomas Fuhr[1], María Naya-Plasencia[2] and Yann Rotella[2]

[1] ANSSI, France
[2] Inria, France

**Abstract.** In this article we study the security of the authenticated encryption algorithm KETJE against divide-and-conquer attacks. KETJE is a third-round candidate in the ongoing CAESAR competition, which shares most of its design principles with the SHA-3 hash function. Several versions of KETJE have been submitted, with different sizes for its internal state. We describe several state-recovery attacks on the smaller variant, called KETJE JR. We show that if one increases the amount of keystream output after each round from 16 bits to 40 bits, KETJE JR becomes vulnerable to divide-and-conquer attacks with time complexities $2^{71.5}$ for the original version and $2^{82.3}$ for the current tweaked version, both with a key of 96 bits. We also propose a similar attack when considering rates of 32 bits for the non-tweaked version. Our findings do not threaten the security of KETJE, but should be taken as a warning against potential future modifications that would aim at increasing the performance of the algorithm.

**Keywords:** Ketje · Authenticated Encryption · cryptanalysis · divide-and-conquer · nonlinear sieving

## 1 Introduction

Authenticated encryption algorithms aim at providing both integrity and confidentiality with only one cryptographic primitive. For example, `AES-GCM` is widely used but is often considered not strong enough. Indeed, when too long messages are encrypted under the same key, or if an initialization value is reused, its security collapses and the integrity key leaks. These features impose heavy conditions on the use of `AES-GCM`.
Hence the CAESAR competition was launched in 2014. The purpose of this competition is to provide a portfolio of robust algorithms with better performances for different environments. These algorithms aim at providing confidentiality and integrity to a message, along with integrity of the so-called (unencrypted) associated data. Such mechanisms are called Authenticated Encryption with Associated Data (AEAD) algorithms.

KETJE [BDP+df] is a family of Authenticated Encryption algorithms that was submitted to this competition, and that is one of the 15 candidates that were selected for its (currently ongoing) third round. It was designed by Bertoni, Daemen, Peeters, Van Assche and Van Keer and reuses some of the internal components of KECCAK [BDPA13], the winner of the SHA-3 competition. KETJE was initially submitted to the competition as a set of two lightweight AEAD, denoted KETJE JR and KETJE SR [BDP+df]. These algorithms rely on a version of the internal permutation of KECCAK, used in a specific mode of operation called the MonkeyWrap construction [BDPA12], which is derived from the sponge construction [BDPA08]. KETJE JR and KETJE SR act on internal states of respective sizes 200 and 400 bits. At the beginning of the third round of the CAESAR competition, the designers proposed a new version of KETJE [BDP+df]. This new version includes two new variants called KETJE MINOR and KETJE MAJOR, with larger internal states of sizes 800 bits and 1600 bits, as well as one modification of previous variants.

The amount of publicly available security analysis of an algorithm is known as an important selection criterion in cryptographic competitions. Therefore, the designers of Ketje are organizing the Ketje cryptanalysis contest. In particular, they show interest in State-Recovery attacks on weakened versions of Ketje, including increases in the rate of the MonkeyWrap construction.

**Our contributions.** In this article, we focus on the smaller variant Ketje Jr, and denote by v1 and v2, the initial and tweaked version of the algorithm. We aim at recovering the internal state of the algorithm, by applying a divide-and-conquer strategy. We target modified versions of Ketje Jr with an increased rate. The security claimed by the authors for Ketje Jr is determined by $\min(96, k)$ for a key of size $k$.

The MonkeyWrap construction that is used by Ketje works as follows. First, the key and initialization vector are loaded in a 200 bits internal state. Then, an initialization step is performed. After this step, the associated data is processed, and then the plaintext. Finally, the tag is extracted. Our attacks focus on the message processing phase, which informally works as follows. The message is divided into $r$-bit blocks, where $r$ is called the encryption rate. At each step, $r$ bits of keystream are extracted from the state, x-ored to a message block to obtain a ciphertext block, which is loaded in the internal state, then an update of the state is performed by the application of a variant of the Keccak permutation.

In this phase, keystream bits are output after each permutation round. Trying to determine the security margin offered by Ketje by studying round-reduced variants is therefore pointless. Instead, we try to evaluate this security margin by targeting variants of Ketje Jr with an increased rate, which can be viewed as the main security parameter during the message processing phase.

We aim at recovering the internal state of Ketje Jr from several consecutive $r$-bit output blocks. For Ketje Jr, the internal state contains 200 bits. Any sequence of $n$ consecutive $r$-bit output blocks can then be generated by $2^{200-nr}$ values of the internal state. As long as $nr \leq 200$, we cannot get a unique candidate for the internal state during encryption. Instead, we aim at getting $2^{200-nr}$ candidates, among which the effective value can be searched exhaustively if more output blocks are available.

In this paper we describe several attacks, that all consist in applying a divide-and-conquer strategy in a known plaintext scenario. Knowing 3 or 4 consecutive blocks of a plaintext-ciphertext pair, and thus of keystream, the adversary tries to guess the remaining bits of the internal state independently half by half, when the second block is extracted. Then, he tries to deduce information on the internal state by computing both forwards and backwards. Information arising from both guesses on half states are then combined with the other 2 or 3 known blocks of keystream to determine all possible values of the internal state.

Our attacks using 3 consecutive blocks can be adapted to both v1 and v2 of Ketje Jr. When the rate is increased to 40, $2^{200-3\times40} = 2^{80}$ values of the state are consistent with a given value of 3 keystream blocks. Our attack recovers these $2^{80}$ values in $2^{82.3}$ operations, which is below the $2^{96}$ security target of Ketje Jr. We also describe an attack that makes use of the knowledge of 4 consecutive keystream blocks, and recovers the $2^{40}$ possible values of the internal state with a complexity of $2^{71.5}$ operations when applied to Ketje Jr v1 with rate 40 bits. Eventually, a variant of these attacks can be applied to Ketje Jr v1 with rate 32, with a total complexity of $2^{92}$, below the $2^{96}$ computations from the generic attack.

We also give estimations on the complexity of our attacks when applied to the recommended parameter set of Ketje Jr, including a rate of 16 bits, and show that they do not threaten the version of Ketje Jr that was submitted to the CAESAR competition.

**Related work.** Since the competition SHA-3, the Keccak-$p$ permutation (described in 2.3) has been widely analyzed [BCC11, DGPW12, JN15]. In the literature, reduced versions of Ketje have been cryptanalyzed with cube attacks [DLWQ17] and linearization techniques [GLS16]. However, all the previous available cryptanalysis focus on the initialization phase of Ketje. As our technique aims at recovering the internal state during the message processing phase, it might show new analysis direction of interest for the crypto community and the CAESAR competition.

**Organization of the paper.** In Section 2, we give a brief description of Ketje Jr. Then, we describe generic divide-and-conquer algorithms and show how to apply them to recover the internal state from 3 consecutive output blocks of Ketje Jr in Section 3. We show in Section 4 how to improve our algorithm and apply it to 4 consecutive output blocks of Ketje Jr v1 with an increased rate of 40 bits. In Section 5 we propose an attack on a version with 32 bits of rate, and conclude in Section 6.

## 2 Description of Ketje Jr

Ketje is an authenticated encryption mode proposed by Bertoni et al. In this section we describe the variant Ketje Jr which is the one that we will center the analysis on in this paper. The Ketje family of AEAD algorithms is a third-round CAESAR candidate designed by Bertoni et al [BDP$^+$df]. It relies on the use of a round-reduced version of the Keccak permutation, together with the MonkeyWrap mode of operation. In the following we give a short description of Ketje Jr, and focus on those of its components that our attacks exploit.

### 2.1 The monkeyWrap mode of operation

The MonkeyWrap mode of operation [BDPA12] is a mode for authenticated encryption with associated data built upon the sponge construction [BDPA08]. It allows for the authenticated encryption of sequences of plaintexts, each one with optional associated data. In the following, we focus w.l.o.g. on the encryption of one plaintext $P$, without associated data. As our attack targets the internal state during the processing of the plaintext, it does not depend on the number of plaintext and associated data fields. The encryption of $P$ consists in the following operations.

**Initialization.** A $b$-bit internal state $S$ is initialized with a key $K$ of variable length (of bit length $k$ recommended 96 bits, up to a max of 182) and a variable length nonce $N$. $S$ is divided into two parts $R$ and $C$ of respective lengths $r$ and $c$. The initial value of $S$ is $enc(k/8)||K||pad_K||N||pad$, where $enc(k/8)$ is a specific encoding of integer $k/8$, $pad_K$ and $pad$ are constant padding strings. Then, 12 rounds of the Keccak-$p$ permutation are applied to $S$.

**Plaintext processing.** The plaintext is padded and divided into a sequence of $r$-bit blocks, $P_0, \ldots, P_{\ell-1}$, which are processed iteratively as follows. First, the state $S$ is divided into $S_r||S_c$, where $S_r$ are its first $r$ bits[1]. One then computes the $i$-th ciphertext block $C_i = P_i + S_r$, and replace $S$ with $C_i||s_c$. Then, one round of the Keccak-$p$ permutation is performed on $S$.

[1]Domain separation bits are appended to each plaintext block. The description of the domain separation mechanism is however not necessary for our attack and is therefore not detailed here.

**Tag extraction.** The final step is the extraction of the authentication tag. After the plaintext processing, 6 rounds are performed on $S$ before the tag extraction. Then, the tag is computed as a sequence of $r$-bit blocks, which are defined as the $S_r$ part of the state. The state is updated by one round between consecutive extractions.

## 2.2 The KETJE JR state

We briefly recall here that the claimed security on KETJE JR is roughly given by $\min(96, k)$. The description of the KECCAK-$p$ permutation rounds that are used in KETJE JR relies on a specific representation of the $b = 200$-bit internal state. The state of KETJE is a three-dimensional array of elements of $\mathbb{F}_2$ of size $5 \times 5 \times 8$. For simplicity, we use the same vocabulary as the authors, namely if $A$ denotes the state, then:

- $A_{x,y,z}$ denotes the bit at position $(x, y, z)$.
- A *row* is a set of 5 bits with constant $y$ and $z$ coordinates $(A_{*,y,z})$.
- A *column* is a set of 5 bits with constant $x$ and $z$ coordinates $(A_{x,*,z})$.
- A *lane* is a set of 8 bits with constant $x$ and $y$ coordinates $(A_{x,y,*})$.
- A *sheet* is a set of 40 bits with constant $x$ coordinate $(A_{x,*,*})$.
- A *plane* is a set of 40 bits with constant $y$ coordinate $(A_{*,y,*})$.
- A *slice* is a set of 25 bits with constant $z$ coordinate $(A_{*,*,z})$

The 200 bits of the state are ordered from 0 to 199 according to their position $(x, y, z)$, as $40x + 8y + z$. We adopt the same representation of the state as the designers. Each slice is represented by a $5 \times 5$ array, with index $(x, y) = (0, 0)$ at the center:

$$S_{*,*,z} = \begin{pmatrix} (2,3) & (2,4) & (2,0) & (2,1) & (2,2) \\ (1,3) & (1,4) & (1,0) & (1,1) & (1,2) \\ (0,3) & (0,4) & (0,0) & (0,1) & (0,2) \\ (4,3) & (4,4) & (4,0) & (4,1) & (4,2) \\ (3,3) & (3,4) & (3,0) & (3,1) & (3,2) \end{pmatrix}$$

## 2.3 The KECCAK permutation

KETJE JR is an iterated authenticated encryption mode where the round function is derived from the KECCAK-$p$ permutation. The KECCAK-$p$ round function consists of five steps: $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$. We now give more details about each of these operations.

### 2.3.1 Specification of $\theta$

$\theta$ is a linear transform that works as represented in figure 1. This operation provides linear diffusion, which is achieved by relying on the sum of all bits of each column of the state $P(A)_{x,z} = \sum_{y=0}^{4} A_{x,y,z}$. We refer to these sums as *parity bits* of the state. Each bit (at position $x, y, z$) is x-ored with the two parity bits $P(A)_{x-1,z}$ and $P(A)_{x+1,z-1}$, where indexes are taken modulo their maximal value (i.e. 5 for the $x$-coordinate and 8 for the $z$-coordinate). The output of $\theta$ is given by $A_{x,y,z} \leftarrow A_{x,y,z} + P(A)_{x-1,z} + P(A)_{x+1,z-1}$.

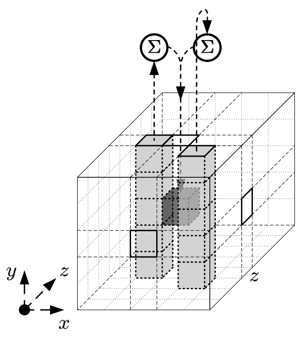

**Figure 1:** Computation of one output bit of $\theta$

### 2.3.2 Specification of $\rho$

$\rho$ is linear and provides diffusion between the slices of the state. It consists of a different circular rotation applied to each lane. Its output is given by $A_{x,y,z} \leftarrow A_{x,y,z-(t+1)(t+2)/2}$, with $t$ satisfying $0 \leq t < 24$ and $\begin{pmatrix}0 & 1\\ 2 & 3\end{pmatrix}^t \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}x\\y\end{pmatrix}$ in $\mathbb{F}_5^{2\times 2}$, or $t = -1$ if $x = y = 0$.

More concretely, the number of positions each lane is rotated by is given by the following matrix:

$$\begin{pmatrix} 1 & 7 & 3 & 2 & 3\\ 7 & 4 & 4 & 4 & 6\\ 4 & 3 & 0 & 1 & 6\\ 0 & 6 & 2 & 2 & 5\\ 5 & 0 & 1 & 5 & 7 \end{pmatrix}$$

### 2.3.3 Specification of $\pi$

$\pi$ is also linear and is shuffles the positions of lanes. It is therefore applied slice by slice, according to the pattern represented on Figure 2. Its output is given by $\begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix}0 & 1\\ 2 & 3\end{pmatrix}\begin{pmatrix}x'\\y'\end{pmatrix}$,

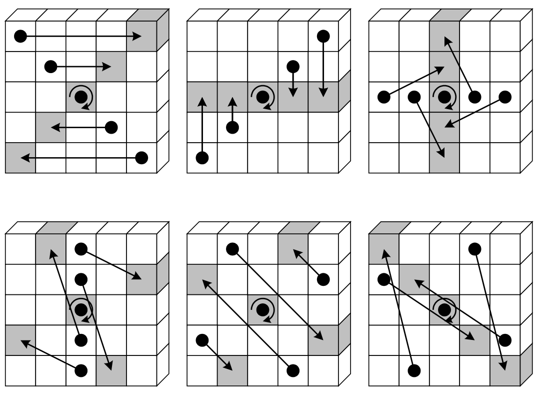

**Figure 2:** Representation of $\pi$ on each slice of the state

### 2.3.4 Specification of $\chi$

$\chi$ is the nonlinear layer of the permutation used in Ketje Jr. It can be interpreted as a layer of 5-bit to 5-bit Sboxes of algebraic degree 2, computed on the rows of the state. Moreover, each output bit depends on only 3 input bits, according to the formula: $A_{x,*,*} \leftarrow A_{x,*,*} + (A_{x+1,*,*} + 1)A_{x+2,*,*}$,

### 2.3.5 Specification of $\iota$

$\iota$ is an addition of a round constant. The round constants are of the following form:

$$\mathrm{RC}[i_r][0,0,2^j-1] = \mathrm{rc}[j+7i_r] \text{ for all } 0 \leq j \leq \ell,$$

the other values are zero and the values $\mathrm{rc}[t] \in \mathbb{F}_2$ are defined as the output of a binary linear feedback shift register (LFSR):

$$\mathrm{rc}[t] = \left(x^t \mod x^8 + x^6 + x^5 + x^4 + 1\right) \mod x \in \mathbb{F}_2[x]\,.$$

Eventually, the Ketje Jr permutation is defined by the application of $n_r$ rounds R, indexed with $i_r$ from $18 - n_r$ to 17.

## 2.4 Keystream extraction

The MonkeyWrap construction with rate $r$ states that the first $r$ bits of the state are extracted as keystream. As the bits are ordered first on their $x$ coordinate, then on their $y$ coordinate, and finally on their $z$ coordinate, keystream bits are extracted by lane. For Ketje Jr v1, as long as the rate does not exceed 40 bits, keystream bits are concentrated on the plane $x = 0$, and on lanes $y = 0 \ldots \lfloor r/8 \rfloor$. Please note that if the rate is 40 bits, the full plane $x = 0$ is output as keystream.

However, in Ketje Jr v2, the authors replace the direct use of the Keccak-$p$ permutation by using a variant of it, called the *twisted permutation*. The twisted permutation Keccak-$p^\star$ is defined as:

$$\text{Keccak-}p^\star[b, n_r] = \pi \circ \text{Keccak-}p \circ \pi^{-1}$$

It is also important to notice that applying Keccak-$p^\star$ is equivalent to output different lanes, not belonging to the same plane, because all intermediate $\pi$ and $\pi^{-1}$ will cancel out. The keystream bits are now still concentrated on lanes, on the diagonal of the state with equation $x = y$, starting from $x = y = 0$ and up to $x = y = \lfloor r/8 \rfloor$.

# 3 Divide-and-Conquer attack using 3 output blocks on Ketje Jr

In all the paper, we describe different attacks on Ketje Jr v1 or v2. Those attacks rely on the following fundamental idea: separate the state in 2 parts, then construct 2 lists independently that cover each part and eventually merge both lists using linear or non-linear sieving relations.

In this section we describe a state-recovery attack that works on both versions of Ketje Jr (*i.e.*, with the initial or the twisted permutations). This attack enables the adversary to recover the state of Ketje Jr during one encryption, under the hypothesis that he knows three consecutive keystream blocks. We first show our attack on Ketje Jr v1, and then show how to adapt it to the new version with the twisted permutation. The strategy is rather generic and applies for any rate. We point out that its complexity decreases with the encryption rate, and is in any case too high to contradict the security claims made by the designers. The complexity of the attacks in this section can not be smaller than $2^{200-3\times r}$ for a rate $r$, as this is the number of possible solutions that we recover, and have to test in order to find the correct one. For a rate of $r = 40$ the best possible achievable complexity is therefore $2^{80}$. Therefore, in order to have an attack better than generic ones, we have to consider a rate of at least $r = 40$ bits.

## 3.1 A basic attack against Ketje Jr v1 with rate 40 bits

The attack strategy against Ketje Jr v1 is rather straightforward. In this subsection we suppose that the rate of Ketje Jr is 40 bits. In other words, a full plane is output after each round.

**Divide-and-conquer framework.** Our attack is based on a divide-and-conquer technique. A generic formulation of the problem we aim at solving is the following:

Given two sets $\mathcal{U}$ and $\mathcal{V}$, two functions $f : \mathcal{U} \to GF(2)^c$ and $g : \mathcal{V} \to GF(2)^c$ and one element $t \in GF(2)^c$, find all $u \in \mathcal{U}$ and $v \in \mathcal{V}$ such that $f(u) + g(v) = t$.

Solving this problem is folklore in cryptography (many examples can be found for instance in meet-in-the-middle attacks [BR10, DSP07, Sas11] or rebound attacks [LMR+09, KNRS10]). One computes $f(u)$ for all $u \in \mathcal{U}$ and stores $(u, f(u))$ in a table, sorted according to the value of $f(u) + t$. Then, one computes $g(v)$ for all $v \in \mathcal{V}$ and search the table for a match. For all $u$ such that $f(u) + t = g(v)$, $(u, v)$ is one solution to the problem.

The memory complexity is $|\mathcal{U}|(\log(|\mathcal{U}|) + c)$ bits, where $|\mathcal{U}|$ is the number of elements in $\mathcal{U}$ and assuming they can be represented on $\log(|\mathcal{U}|)$ bits. The time required to mount the attack is $|\mathcal{U}| + |\mathcal{V}|$ computations of functions $f$ and $g$, and $|\mathcal{U}| \times |\mathcal{V}| \times 2^{-c}$ to describe the set of solutions.

**Divide-and-conquer against Ketje Jr.** This technique can be applied to 3 consecutive output blocks of Ketje Jr, covering two Keccak rounds. Let us introduce the following notation: $A^0$ is the state containing the first known output, $B^0$ the value of the state after applying $\theta$ to $A^0$ and $C^0$ the state after $\rho$ and $\pi$. Then, $\chi$ is applied and a new output block is computed. We also denote by $A^1$, $B^1$ and $C^1$ the same values one round later, and by $A^2$ the value after the last $\chi$ layer. These notations are displayed on Figure 3.

$$\begin{tikzcd}
A^0_{*,0,*} & & & A^1_{*,0,*} & & & A^2_{*,0,*} \\
\boxed{A^0} \arrow[u] \arrow[r, "\theta"] & \boxed{B^0} \arrow[r, "\rho,\pi"] & \boxed{C^0} \arrow[r, "\chi"] & \boxed{A^1} \arrow[u] \arrow[r, "\theta"] & \boxed{B^1} \arrow[r, "\rho,\pi"] & \boxed{C^1} \arrow[r, "\chi"] & \boxed{A^2} \arrow[u]
\end{tikzcd}$$

**Figure 3:** Notations used for our attack on 3 consecutive outputs $A^0_{*,0,*}, A^1_{*,0,*}, A^2_{*,0,*}$ for Ketje Jr v1 and a rate of 40 bits.

We can now describe our attack. The idea is to recover the full state $A^1$ by using a divide-and-conquer strategy, sieving with the information known from blocks $A^0$ and $A^2$. Before starting the core of our attack, we can compute the planes $C^1_{*,0,*}$ by computing the inverse Sbox layer on $A^2_{*,0,*}$. Then, we can express $C^1_{*,0,*}$ as a function of $A^1$, and $A^0_{*,0,*}$ as a function of $C^0$. A key to our attack is that as $\theta, \rho, \pi$ and their inverse permutations are linear we can divide states $A^1$ and $C^0$ into two halves $(A^u, A^v)$, corresponding to $(C^u, C^v)$ and write the previous expressions as

$$\begin{aligned} C^1_{*,0,*} &= f_1(A^u) + g_1(A^v) \\ A^0_{*,0,*} &= f_2(C^u) + g_2(C^v) \end{aligned}$$

More precisely, we define $A^u$ (resp. $C^u$) as the first four slices (with index $0 \leq z \leq 3$) of $A^1$ (resp. $C^1$).

We can now describe the core of our attack. We guess the front half state $A^u$. For each slice, the 5 bits at position $y = 0$ are already known, which leaves 20 bits to guess per slice, and a total of 80 bits to guess from $A^u$. We denote our guess by $i \in \{0,1\}^{80}$. Then, we can compute the resulting value of $C^u$, by applying $\chi^{-1}$. We then define

$$f(A^u) = (f_1(A^u), f_2(C^u)).$$

Similarly, by guessing 80 bits of the back half of the state $A^v$, we can compute

$$g(A^v) = (g_1(A^v), g_2(C^v)).$$

Our attack then consists in applying the divide-and-conquer strategy described above to $A^u$, $A^v$ and $t = (C^1_{*,0,*}, A^0_{*,0,*})$. We have $|\mathcal{U}| = |\mathcal{V}| = 2^{80}$ and $c = 80$, therefore the time complexity of our attack is $2^{80}$ evaluations of $f$ and $g$ for the divide-and-conquer phase. Then, the remaining $2^{80}$ candidates can be searched exhaustively, for a cost of $2^{80}$.

The general idea is summarized on Figure 4.

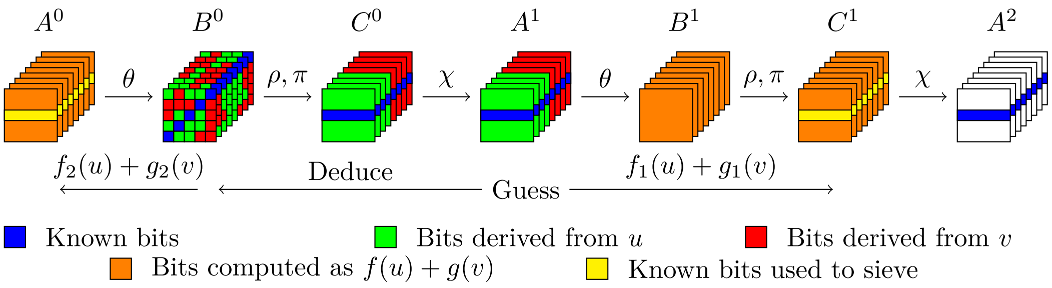

**Figure 4:** Summary of our basic state-recovery attack

## 3.2 An attack against Ketje Jr v2 with rate 40 bits

### 3.2.1 Sieving with nonlinear relations

We now show how to modify the previous attack to apply it to Ketje Jr with the twisted permutation. By studying every step of the attack, we can notice that the only point that prevents it from being directly applied is that we can no longer compute 40 bits of $C^1$ from known bits of $A^2$, as the output bit positions no longer cover outputs of the same Sbox. We overcome this problem by sieving using $A^2$ instead of $C^1$ in our divide-and-conquer attack.

To this end, we need to modify our divide-and-conquer algorithm. Indeed, our previous attack consists in expressing known bits as the sum of two values computed from independent guesses $u$ and $v$. One limitation of this strategy is that it cannot be applied as soon as such an expression involves a nonlinear combination of $u$ and $v$. We now demonstrate how to modify our attack to encompass more cases without a significant increase in the time complexity of the algorithm. After that, we show how to use our new algorithm to improve the attack against Ketje Jr.

**Divide-and-conquer with nonlinear interactions.** We now modify the divide-and-conquer framework (that was previously limited to linearly independent expressions in $u$ and $v$) to encompass cases that will help attacking Ketje Jr with the twisted permutation.

We define the following problem as a system of equations built from three subsystems with respective sizes $\alpha$, $\beta$ and $\gamma$. We consider the following boolean functions :

- $2\alpha$ functions $f_i : \mathcal{U} \to \{0,1\}$ and $g_i, : \mathcal{V} \to \{0,1\}$, for $0 \leq i \leq \alpha - 1$ ;
- $3\beta$ functions $f'_i : \mathcal{U} \to \{0,1\}$ and $g_i, g'_i : \mathcal{V} \to \{0,1\}$, for $\alpha \leq i \leq \alpha + \beta - 1$ ;
- $3\gamma$ functions $f_i, f'_i : \mathcal{U} \to \{0,1\}$ and $g'_i : \mathcal{V} \to \{0,1\}$, for $\alpha + \beta \leq i \leq \alpha + \beta + \gamma - 1$ ;

The problem we try to solve then consists in finding all $(u, v) \in \mathcal{U} \times \mathcal{V}$ such that :

$$f_i(u) + g_i(v) = 0 \text{ for } 0 \leq i \leq \alpha - 1 \tag{1}$$
$$f'_i(u)g'_i(v) + g_i(v) = 0 \text{ for } \alpha \leq i \leq \alpha + \beta - 1 \tag{2}$$
$$f_i(u) + f'_i(u)g'_i(v) = 0 \text{ for } \alpha + \beta \leq i \leq \alpha + \beta + \gamma - 1 \tag{3}$$

In summary, we consider a system of $c = \alpha + \beta + \gamma$ equations and partially remove the condition on linearly independent contributions from $u$ and $v$ to all equations. More precisely, we can deal with equations involving one product of $u$ and $v$ dependent values, provided there is only one other term involving either $u$ or $v$ (but not both). A similar problem was treated in [LN15].

**Link with Ketje.** The nonlinear layer $\chi$ of KETJE involves only one product, namely $(a, b, c) \to a + bc$. We focus on a very specific case of divide-and-conquer attacks, involving a $\chi$-layer in which:

- Every input bit can be fully computed by guessing either $u$ or $v$;
- Some output bits are known to the adversary.

Depending on which half of the state input bits $a$, $b$ and $c$ are computed from, the knowledge of the output bit $a + bc$ can be expressed as an equation of type 1, 2 or 3.

**Our modified divide-and-conquer strategy.** As in the case of the initial problem of Section 3.1, our solution consists in building two (sorted) lists of values $L_{\mathcal{U}}$ and $L_{\mathcal{V}}$ such that each solution $(u, v)$ to our problem leads to a match between both lists. We also want to be able to recover $u$ and $v$ from the matching elements, and require that our algorithm avoid false positives, i.e., matches between lists that do not lead to values $u$ and $v$ that verify the system of equations.

We now describe how we build the list $L_{\mathcal{U}}$. Each element of this list is a couple $(F, u)$, where $F$ is a $c$-bit value. The bit at each position $i$ of $F$ is meant to be compared with the same bit of elements $(G, v)$ of $L_{\mathcal{V}}$ to verify the equation involving $f_i, f'_i, g_i, g'_i$. We append $u$ to the elements of the list so as to recover a solution $(u, v)$ from a match.

**Dealing with equations independently.** We now describe how we deal with each equation. For each value $u$ and $v$ and for each equation, we aim at defining sets $R_i(u)$ and $S_i(v)$ such that there is a collision between $R_i(u)$ and $S_i(v)$ if and only if $(u, v)$ is a solution to the $i$-th equation. We now study the three possible cases.

1. Case of eq.1. This is the usual case of divide and conquer attacks, with independent contributions from $u$ and $v$. The equation is satisfied if $f_i(u) = g_i(v)$. We therefore define $R_i(u) = \{f_i(u)\}$ and $S_i(v) = \{g_i(v)\}$.

2. Case of eq.2. Our goal is to verify such an equation by comparing $f'_i(u)$ with some $v$-dependent values. Therefore, we define $R_i(u) = \{f'_i(u)\}$. We handle $v$ in a more complicated manner. $S_i(v)$ is build from values $g_i(v), g'_i(v)$ only, but different cases occur. First, one computes $g'_i(v)$. If $g'_i(v) = 1$, equation 2 becomes $f'_i(u) = g_i(v)$ and therefore we define $S_i(v) = \{g_i(v)\}$. If $g'_i(v) = 0$, equation 2 becomes $g_i(v) = 0$. If $g_i(v) = 1$, the $i$-th equation cannot be satisfied. Therefore $S_i(v) = \emptyset$. Finally, if $g_i(v) = 0$, the $i$-th equation holds independently of $u$. We need to have a match between any $R_i(u)$ and $S_i(v)$, therefore we define $S_i(v) = \{0, 1\}$.

3. Case of eq.3. Equation 3 is equivalent to equation 2 by exchanging the roles of $u$ and $v$. We therefore handle these equations as in case 2, exchanging $u$ and $v$.

In each case, one can check that there is a collision between $R_i(u)$ and $S_i(v)$ if and only if $(u, v)$ is a solution to the $i$-th equation.

**Solving all equations simultaneously.** Let us now consider the cartesian products $R(u) = R_0(u) \times \ldots \times R_{\alpha+\beta+\gamma-1}(u)$ and $S(v) = S_0(v) \times \ldots \times S_{\alpha+\beta+\gamma-1}(v)$. If $(u, v)$ is a solution to all the $\alpha + \beta + \gamma$ equations, there is a collision between elements of sets $R_i(u)$ and $S_i(v)$ for all $i$, and thus there is a collision between elements of the cartesian products $R(u)$ and $S(v)$. Conversely, if there is a collision between elements of $R(u)$ and $S(v)$, each coordinate of this collision gives a collision between $R_i(u)$ and $S_i(v)$, and thus $(u, v)$ is a solution of all the equations.

**Solving the nonlinear divide-and-conquer problem.** Our algorithm then consists in building $L_{\mathcal{U}}$ by enumerating all the elements $(F, u)$ for all elements in $R(u)$ and all $u$ in $\mathcal{U}$. Elements are sorted according to the value of $F$ (in lexicographic order for instance). A similar operation is done to compute $L_{\mathcal{V}}$. Then, the solutions $(u, v)$ to our problem can be retrieved by searching collisions in the lists. We proceed in two steps to build $L_{\mathcal{U}}$. First, we show in Algorithm 1 how to build the first $\alpha$ bits of all values in $R(u)$, for any guess $u$, and store them in a list denoted $L_u$. A similar list $L_v$ can be computed similarly by replacing loops from $\alpha + \beta$ to $\alpha + \beta + \gamma - 1$ by loops from $\alpha$ to $\alpha + \beta - 1$. Then, in Algorithm 2, we show how to build the full list $L_{\mathcal{U}}$, by completing all $\gamma$-bit elements of each list $L_u$ to a $c$-bit value and merging all these lists. Again, the list $L_{\mathcal{V}}$ can be computed similarly.

**Size of the lists.** Now, we have to evaluate the size of the lists $L_{\mathcal{U}}$ and $L_{\mathcal{V}}$ that are merged during our algorithm. To do this, we compute the expected size of $L_u$, which is the expected number of entries that are added to the list $L_{\mathcal{U}}$ for each value of $u$. We focus on the joint values of $(f_i(u), f'_i(u))$ for all $\alpha + \beta \leq i < \alpha + \beta + \gamma$. First, according to Algorithm 1, entries are added to the list only if none of these values is $(1, 0)$, which happens with probability $(3/4)^\alpha$. We denote by $\omega$ this event, and by $N(u)$ the number of entries added to the list. We have $N(u) = 2^j$, where $j$ is the number of positions such that $(f_i(u), f'_i(u)) = (0, 0)$. We can now compute:

$$\Pr\left[N(u) = 2^j | \omega\right] = \binom{\alpha}{j} (1/3)^j (2/3)^{\alpha-j} .$$

The expected value of $N(u)$ is then:

$$\begin{array}{rcl} E(N(u)) & = & \sum_{j=0}^{\alpha} \left(\Pr\left[N(u) = 2^j | \omega\right] \Pr[\omega] \times 2^j\right) \\ & = & (3/4)^\alpha \times \sum_{j=0}^{\alpha} \binom{\alpha}{j} (1/3)^j (2/3)^{\alpha-j} 2^j \\ & = & (3/4)^\alpha \times (2/3 + 2/3)^\alpha \\ & = & 1. \end{array}$$

**Algorithm 1** Computing the list $L_u$ of elements $Z$ such that equations $\alpha+\beta$ to $\alpha+\beta+\gamma-1$ are satisfied for any $(u, v)$ such that $Z = (g'_{\alpha+\beta}(v), \dots, g'_{\alpha+\beta+\gamma-1}(v))$.

```
Input: $u \in \mathcal{U}$
Output: a list $L_u$ containing all elements of $R_{\alpha+\beta}(u) \times \dots \times R_{\alpha+\beta+\gamma-1}(u)$
  $L_u \leftarrow (\epsilon)$ // Initialize $L_u$ with the empty string
  for $i = \alpha + \beta$ to $\alpha + \beta + \gamma - 1$ do
    // Check whether the list is empty
    if $f_i(u) = 1$ and $f'_i(u) = 0$ then
      return ()
  for $i = \alpha + \beta$ to $\alpha + \beta + \gamma - 1$ do
    if $f'_i(u) = 1$ then
      for $Z \in L_u$ do
        append $f_i(u)$ to $Z$
    else
      //$f_i(u) = 0$ and $f'_i(u) = 0$
      for $Z \in L_u$ do
        replace $Z$ with two elements $Z||0$ and $Z||1$ in $L_u$
  return $L_u$
```

**Algorithm 2** Enumerating all values in $L_{\mathcal{U}}$.

```
Input: $\mathcal{U}$, functions $f_i$, $0 \leq i < \alpha$ and $\alpha+\beta \leq i < \alpha+\beta+\gamma$, functions $f'_i$, $\alpha \leq i < \alpha+\beta+\gamma$
Output: The list $L_{\mathcal{U}}$
  $L_{\mathcal{U}} \leftarrow ()$
  for $u \in \mathcal{U}$ do
    Compute $X = f_0(u)|| \dots ||f_{\alpha-1}(u)$
    Compute $Y = f'_\alpha(u)|| \dots ||f'_{\alpha+\beta-1}(u)$
    Compute $L_u$ using Algorithm 1
    for $Z \in L_u$ do
      Insert $(X||Y||Z, u)$ in $L_{\mathcal{U}}$
  return $L_{\mathcal{U}}$
```

The average size of the list resulting from the first step of our algorithm is then $|\mathcal{U}|$. Similarly, the number of values tried during the second step is $|\mathcal{V}|$.

**Complexity analysis.** For the sake of simplicity, we did not try to completely optimize the algorithm we use to build lists $L_{\mathcal{U}}$ and $L_{\mathcal{V}}$. One of the reasons for it is that for our attack against Ketje Jr, we will precompute these lists, and the time required for that will not be the bottleneck of the attack. For each value of $u$, we need to compute at most $\alpha+\beta+2\gamma$ functions $f_i$ or $f'_i$, and for each of the last $\gamma$ equations, we need two comparisons to determine which values need to be added to the list. Each value added to the list also requires at most $\gamma$ modifications on the list $L_u$, and one insertion in the global list $L_{\mathcal{U}}$. The total complexity is therefore bounded by $2c(|\mathcal{U}|+|\mathcal{V}|)$ computations and comparisons and $c(|\mathcal{U}|+|\mathcal{V}|)$ insertions in lists.

In the sieving step, we search for matches between two lists of respective (average) sizes $|\mathcal{U}|$ and $|\mathcal{V}|$, which can be achieved in $|\mathcal{U}|+|\mathcal{V}|$ operations. This is equivalent to the complexity of the divide-and-conquer without nonlinear sieving algorithm.

### 3.2.2 An improved attack against Ketje Jr v2

**Adapting guesses on parts of the state.** We can now come back to Ketje Jr.

The state $A^1$ is divided in two halves $A^u$ and $A^v$, increasing the size of the lists by adding parity bits of 5 columns. More precisely, $A^u$ is defined as the first four slices ($0 \leq z \leq 3$) with the 5 parity bits of the columns of $A^1_{i,*,7}$, for $i \in \{0,1,2,3,4\}$, and $A^v$ corresponds to the last four slices ($4 \leq z \leq 7$) with the 5 parity bits of the columns of $A^1_{i,*,3}$, for $i \in \{0,1,2,3,4\}$. By adding those 5 parity bits, the application $\theta$ becomes transparent: $B^u$ and $B^v$ (first four slices and last four slices of $B^1$) are immediately derived from $A^u$ and $A^v$ independently.
For guessing the first half augmented state $A^u$, we know that for each slice the 5 bits at position $y=0$ are already known (from $A^1_{*,0,*}$), but also there are 5 known bits of information from $B^1$, which leaves a total of $4\times5\times5+5-4\times5-4\times5=65$ bits.
As the 5 additional bits from each list correspond to parity bits from the other list, there exist 10 extra linear equations (in addition to the 40 bit conditions given by the known values from $A^0$) that can sieve the number of possible combinations between the two lists (factor $2^{-10-40}$ when considering all the linear relations). In other words there exist $f_1$ and $g_1 : \{0,1\}^{65} \to \{0,1\}^{10}$ such that

$$0 = f_1(A^u) + g_1(A^v)$$

In the same way as in 3.1, there exist two functions $f_2$ and $g_2$ such that

$$A^0_{*,0,*} = f_2(A^u) + g_2(A^v)$$

**Removing the condition on the rate.** Our first attack from Section 3.1 could only work if the keystream blocks extracted from the state cover full Sboxes, so that one can invert the Sbox layer to prepare the divide-and-conquer attack. As we no longer need to invert a $\chi$ layer, this condition disappears. We can then apply our strategy to both versions of Ketje Jr even if the rate is smaller than 40 bits. The complexity of our attack however highly depends on the rate.

**Saving time through pre-computation.** Our algorithm involves the construction of lists $L_{\mathcal{U}}$ and $L_{\mathcal{V}}$, which implies a time complexity linear in the number of equations, due to iterations of Algorithm 1. However, in the case of Ketje Jr, we can improve our algorithm by computing $L_u$ for all possible values of $u$ in a pre-computation step. Indeed, let us

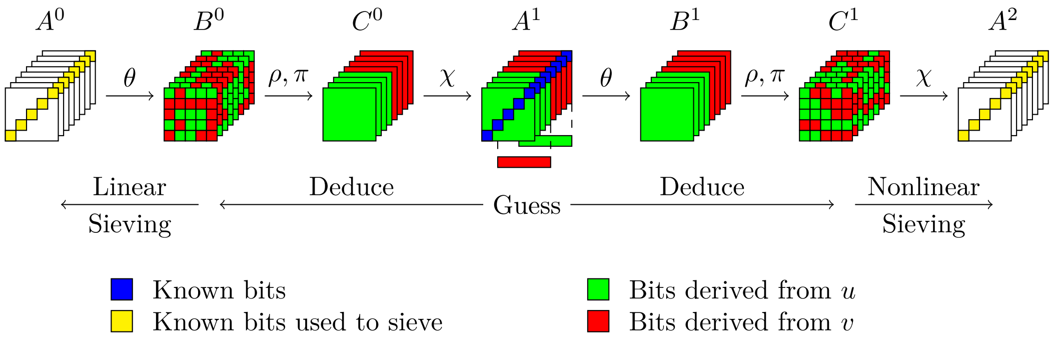

**Figure 5:** Summary of our advanced divide-and-conquer attack

take a deeper look at $C^1$ and $A^2$ and look for nonlinear sieving relations (see Equations 2 and 3), i.e., known bits of $A^2$ which expression involves the product of two bits of $C^1$, one computed from $A^u$ and the other from $A^v$. We can count that for Ketje Jr v1, there are at most 20 such relations, leading to $\beta \leq 10$ and $\gamma \leq 10$. Overall, (at most) 20 bits of $C^1$ depending on $u$ are involved in the $\gamma$ case 3 equations, and similarly (at most) 20 bits depending on $v$ are involved in the $\gamma$ case 3 other relations. As a consequence, all the $2^{20}$ possible lists $L_u$ (and similarly $L_v$) can be precomputed before starting Algorithm 2, for a (negligible) complexity of about $4 \times 10 \times 2^{20}$, and a memory of about $10 \times 2^{20}$ bits per list. Please note that we consider Ketje Jr v1 here because if the rate is below 40, one can no longer invert the $\chi$ layer between $C^1$ and $A^2$ and thus, one uses $A^2$ to sieve nonlinearly. For Ketje Jr v2, we can compute that $\alpha \leq 9$ and $\beta \leq 9$, leading to even lower complexities.

Therefore, partial lists $L_u$ and $L_v$ can be pre-computed using Algorithm 1 and stored, as only (at most) $2^{20}$ cases can occur for both $u$ and $v$. When computing $L_{\mathcal{U}}$ (resp. $L_{\mathcal{V}}$), one does not need to run Algorithm 1 and to recompute $L_u$ (resp. $L_v$) for each value of $u$ (resp. $v$), but only search for it in a precomputed list and insert the corresponding values in $L_{\mathcal{U}}$ (resp. $L_{\mathcal{V}}$).

**Complexity analysis.** Each guess $u$ or $v$ consists of 4 slices of 25 bits and 5 parity bits from $A^1$, but $r/2$ of these bits are known. Therefore, the complexity required to generate each list is

$$T_{\text{list}} = 2^{105-r/2} .$$

The number of solutions given by our divide-and-conquer algorithm depends on the number of sieving relations. We have $r$ sieving relations from the value of $A^2$, $r$ sieving relations from the value of $A^0$, and 10 sieving relations from the parity bits. Therefore, the number of remaining values for the full state $A^1$ is

$$\begin{aligned} T_{\text{search}} &= \left(2^{105-r/2}\right)^2 \times 2^{-10-2r} \\ &= 2^{200-3r} . \end{aligned}$$

When the rate is less than 40, the cost of the exhaustive search on the remaining solutions dominates the complexity of the attack. When the rate is 40, the complexity mainly comes from the computations of the values of $u$ and $v$. We can however improve this complexity by noticing that some bits of $A^2$ might fully be computed from $u$ or $v$, thus reducing the size of the lists (but not the number of solutions left after the divide-and-conquer part). By looking carefully at the details of $\rho$ and $\pi$, we find that in the case of

KETJE JR v2, 4 bits can be recovered from $u$ and 4 bits from $v$. Thus, the complexity of searching all the values of $u$ and $v$ drops from $2^{85}$ to $2^{81}$, leading to an overall complexity of $2 \times 2^{81} + 2^{80} \approx 2^{82}$.

The case of KETJE JR v1 is studied in Section 3.3. We show that the list of $2^{80}$ possible values of the internal state can be recovered with a complexity of $2^{66}$ operations, therefore the exhaustive search dominates the time complexity of the attack in that case.

Our results are summarized in Table 1

**Table 1:** Complexities of our divide-and-conquer attack with nonlinear sieving

| Cipher | Version | Rate | Complexity |
|---|---|---|---|
| KETJE JR | v1 and v2 | 16 | $2^{152}$ |
| KETJE JR | v1 and v2 | 24 | $2^{128}$ |
| KETJE JR | v1 and v2 | 32 | $2^{104}$ |
| KETJE JR | v2 | 40 | $2^{82}$ |
| KETJE JR | v1 | 40 | $2^{80}$ |

## 3.3 Application to the initial KETJE JR with rate 40 bits.

We now study the specific case of the initial KETJE JR permutation with an increased rate of 40 bits. In that case, the adversary can compute backwards the partial $\chi$ layer between states $A^2$ and $C^1$, and therefore knows 40 bits of state $C^1$. As $\pi$ and $\rho$ only shuffle bit positions, the adversary knows 40 bits of $B^1$. Moreover, one can easily notice that these bits are located on 5 lanes. Therefore, the adversary knows 5 bits on each slice.

In our advanced attack represented on Figure 5, we can see that the adversary can deduce the value of 4 full slices of $B^1$ from his guess (both the green phase and the red phase), as they are computed linearly (through a $\theta$ layer) from the guessed bits. Putting it together, he gets 20 linear relations on the 85 guessed bits in each phase of the attack. Taking account of these relations, he only needs to guess 65 bits. This case is represented on Figure 6. One can also notice that nonlinear sieving relations are not used in that specific case.

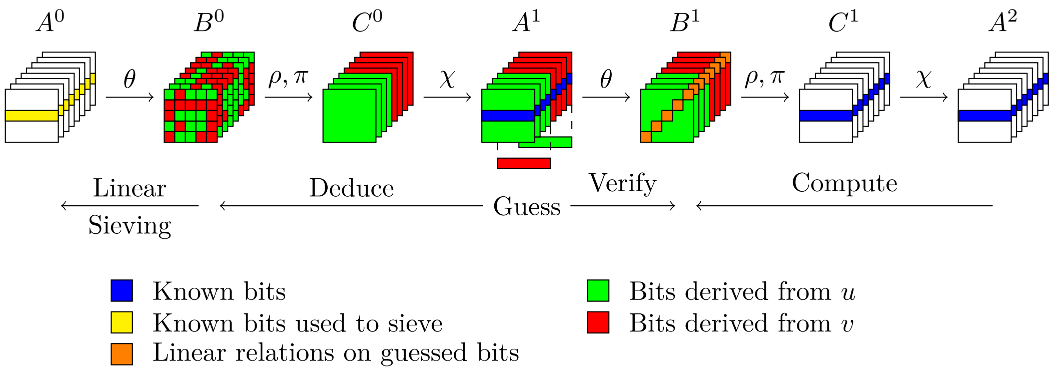

**Figure 6:** Divide-and-conquer attack against initial KETJE JR with rate 40 bits with guessing of parity bits.

Unsurprisingly, the number of remaining candidates after the divide-and-conquer phase of the attack is left unchanged. The number of pairs before sieving is divided by

$(2^{20})^2 = 2^{40}$, however the 40 known bits of $A^2$ have already been used and do not provide useful information for the sieving phase.

## 4 An attack using 4 output blocks with a rate of 40 bits on Ketje Jr

We describe here a more performant attack that uses 4 consecutive output blocks (*i.e.* 3 rounds) of the non-twisted version of Ketje Jr and we consider a rate of 40 bits. Hence, this produces a smaller number of solutions of $2^{200-4\times r} = 2^{40}$. The principle of the attack is similar, but we have in addition a last non-linear sieving using the keystream extracted from $A^3$ that is the most complicated part of the attack, and we will describe in detail in the next sections how to efficiently perform it. In fact, we aim at sieving with $B^2$ instead of $A^3$ because $\chi$ can be inverted on the full plane $A^3_{*,2,*}$ and the $\rho\pi$ application can also be inverted such that we move the known parts to the lanes $B^2_{i,i,*}$. For this we propose two different methods. The first performs a few initial guesses on some bits in the state $C^1$ in order to be able to partially compute through the non-linear relations (that are coming from the $\chi$-layer between $C^1$ and $A^2$) and sieve linearly. The second uses the merging lists algorithm from [Nay11], refined in [CNV13] with the ideas from [DDKS12]: the instant matching algorithm and the parallel matching without memory.

Our attack considers 4 consecutive output blocks of Ketje Jr v1, covering then three Keccak rounds. We use the same notations as in 3.1 and in figure 7. The idea relies also on guessing separately both halves of the state that complete the known part of $A^1$, and merging these lists by considering the sieve given by the information from $A^0$, $A^2$ and $A^3$. As in 3.1, we can compute the planes $C^2_{*,0,*}$, $C^1_{*,0,*}$ and $C^0_{*,0,*}$ through the inverse of $\chi$, as the whole corresponding rows are well known. Moreover, we also compute the $(\rho\pi)^{-1}$ application and get the full lanes $B^0_{i,i,*}$, $B^1_{i,i,*}$ and $B^2_{i,i,*}$ for $i \in \{0, 1, 2, 3, 4\}$ as it is shown in figure 7.

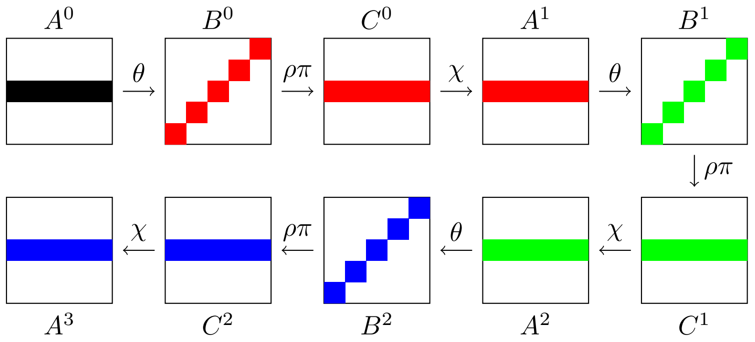

**Figure 7:** Representation of 3 rounds of Ketje. Each colored part corresponds to lanes that can be directly computed known from the 4 known output blocks.

As previously described in section 3.2.2, the state $A^1$ is divided in two halves $A^u$ and $A^v$, increasing the size of the lists by adding parity bits of 5 columns. Each list has a size of $2^{4\times5\times5+5-4\times5-4\times5} = 2^{65}$ elements.

There exist 10 extra linear equations (in addition to the 40 bit conditions given by the known values from $A^0$) that can sieve the number of possible combinations between the two lists. In other words there exist $f_1$ and $g_1 : \{0,1\}^{65} \to \{0,1\}^{10}$ such that

$$0 = f_1(A^u) + g_1(A^v)$$

In the same way as in 3.1, there exist two functions $f_2$ and $g_2$ such that

$$A^0_{*,0,*} = f_2(A^u) + g_2(A^v)$$

## 4.1 First method for sieving with $B^2$: preliminary guessing

As we focus on guessing slices independently, we focus on the application of $\rho\pi$ to the slices. More precisely, $B$ denotes the state before $\rho\pi$, hence each bit of the lane $B_{0,4,*}$ will stay in the same slice, each bit of $B_{1,4,*}$ will be shifted by 2,... Those shift values are called the $\rho$-shift offsets and are displayed in figure 8.

| | | | | |
|---|---|---|---|---|
| 0 | 2 | 1 | 6 | 1 |
| 5 | 5 | 4 | 4 | 3 |
| 5 | 6 | 0 | 4 | 3 |
| 1 | 2 | 6 | 7 | 7 |
| 7 | 0 | 3 | 4 | 2 |

**Figure 8:** $\rho$-shift offsets, that is the positions in each lane where bits of slice $z = 0$ before the application of $\rho\pi$ are relocated.

Eventually, we look where the bits of $A^u$ and $A^v$ are found in the state $C^1$ after the $\rho\pi$ application. However, one needs to pass from $C^1$ on to $A^2$ through $\chi$ and then to $B^2$ through $\theta$, in order to sieve with known bits of $B^2$ (colored in blue in 7). To do so, we fix bits of the state such that some rows in $C^1$ are fully determined by the value $A^u$ (resp. $A^v$). The bits that are guessed this way are given in figure 9. However, the attack has to be done $2^\ell$ times if $\ell$ denotes the number of guessed bits. But guessing those bits allows us to compute entire rows on the state $A^2$ with only bits from $A^u$ (resp. $A^v$). However, the application $\theta$ after $A^2$ remains, so we have a clever choice of guesses to make in order to be able to sieve with the known part of the state $B^2$. To do so we fix bits such that we can compute at least two consecutive slices of $A^2$, hence there will be a linear sieving. The details of the state $C^1$ and the choices of bits to guess are given in figure 9.

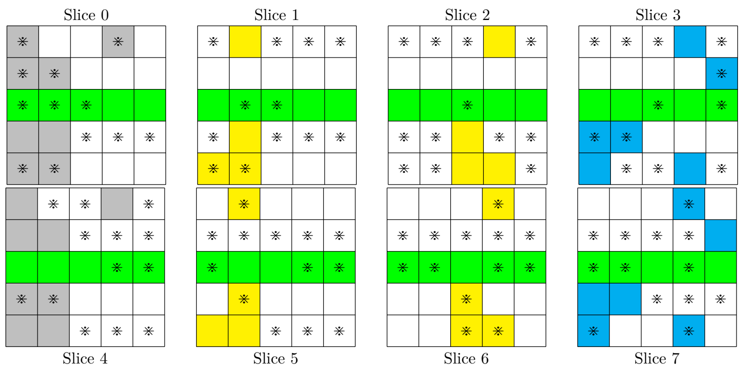

**Figure 9:** $C^1$: the green part is known, the bits with ⋇ correspond to the half state $A^u$, the other ones to $A^v$. Bits colored in yellow, grey or blue correspond to the best choice of guesses such that an entire slice is known after $\chi$.

Moreover, guessing bits decreases the size of the two lists, and allows us to sieve more bits, by using the known information from $B^2$ (that is immediately obtained from $A^3_{0,*,0}$). On the other hand, the number of guess will increase our time complexity.

To be more precise, let us explain one choice of guess: we fix all bits in yellow (that are on the slices 1, 2, 5 and 6). There are 16 of them. 8 are coming from $A^u$ and 8 are coming from $A^v$, then the size of the lists becomes $2^{65-8} = 2^{57}$. Hence, we can compute $5 \times 4$ rows of the state $A^2$, there exists then two functions $f_3$ and $g_3 : \{0,1\}^{57} \to \{0,1\}^{10}$ such that

$$(B^2_{0,0,2}, .., B^2_{4,4,2}, B^2_{0,0,6}, ..., B^2_{4,4,6}) = f_3(A^u) + g_3(A^v)$$

Eventually, we then define $f$ and $g$: $\{0,1\}^{57} \to \{0,1\}^{60}$:

$$f(A^u) = (f_1(A^u), f_2(A^u), f_3(A^u))$$

$$g(A^v) = (g_1(A^v), g_2(A^v), g_3(A^v))$$

Our attack consists now in applying the strategy that merges lists using linear sieving relations described in section 3 to $u = A^u$, $v = A^v$ and the known values 0,...,0, $A^0_{*,0,*}$, $B^2_{0,0,2}$,..,$B^2_{4,4,2}$, $B^2_{0,0,6}$,...,$B^2_{4,4,6}$ where the 0's come from the 10 parity bits of columns $A^1_{i,*,3}$ for $i \in \{0,1,2,3,4\}$. We have $|\mathcal{U}| = |\mathcal{V}| = 2^{57}$ and $c = 10 + 40 + 10 = 60$, however we have to do the attack $2^{16}$ times (for all possible values of guesses), therefore the time complexity is $2^{16} \times 2^{57} = 2^{73}$ evaluations of $f$ and $g$ for this phase. Then, the number of remaining candidates is $2^{57}2^{57}2^{-10}2^{-40}2^{-10}2^{16} = 2^{70}$. Those candidates can be searched exhaustively for a cost of $2^{70}$.

This choice of guess is the best one we can do regarding the total time complexity of the attack, that is $2^{73}$. The other reasonable choices of guesses are described in Table 2.

**Table 2:** Attack complexity for different choices of guessed slices. Guess is the number of bits the adversary has to guess and Rel. is the number of new sieving relations he gets. $T_{\text{search}}$ is the complexity of the exhaustive search on the remaining state values after the sieving.

| Choice | $\lvert\mathcal{U}\rvert$ | $\lvert\mathcal{V}\rvert$ | Rel. | Guess | $T_{\text{search}}$ | $N_{eval}(f,g)$ | Complexity |
|---|---|---|---|---|---|---|---|
| | $2^{65}$ | $2^{65}$ | 0 | 0 | $2^{80}$ | $2^{65}$ | $2^{80}$ |
| $\{1,2\}$ | $2^{63}$ | $2^{59}$ | 5 | 8 | $2^{75}$ | $2^{71}$ | $2^{75}$ |
| $\{1,2,5,6\}$ | $2^{57}$ | $2^{57}$ | 10 | 16 | $2^{70}$ | $2^{73}$ | $2^{73}$ |
| $\{1,2,3,5,6\}$ | $2^{54}$ | $2^{54}$ | 15 | 22 | $2^{65}$ | $2^{76}$ | $2^{76}$ |
| $\{1,2,3,5,6,7\}$ | $2^{51}$ | $2^{51}$ | 20 | 28 | $2^{60}$ | $2^{79}$ | $2^{79}$ |
| $\{1,2,3,4,5,6\}$ | $2^{52}$ | $2^{48}$ | 25 | 30 | $2^{55}$ | $2^{82}$ | $2^{82}$ |
| $\{1,2,3,4,5,6,7\}$ | $2^{49}$ | $2^{45}$ | 30 | 36 | $2^{50}$ | $2^{85}$ | $2^{85}$ |
| $\{0,...,7\}$ | $2^{43}$ | $2^{43}$ | 40 | 44 | $2^{40}$ | $2^{87}$ | $2^{87}$ |

## 4.2 Second method for sieving with $B^2$: list merging

Here we have two lists $L_1$ (which corresponds to $A^u$) and $L_2$ (which corresponds to $A^v$) of size $2^{65}$ each as explained in the beginning of section 4. We also have 50 linear relations coming from the 40 known bits of $A^0$ and the 10 parity bits of columns that should be satisfied by any candidate pair of elements. Each list can be partitioned in $2^{50}$ sublists of average size $2^{15}$. All the elements in each sublist are associated to the same value of the corresponding half of the state associated to the 50 linear relations. In order to separate each list in $2^{50}$ sublists we compute and store the associated values to the linear relations $(f_1(A^u), f_2(A^u))$ from the elements in $L_1$ and $(g_1(A^v), g_2(A^v))$ from the elements in $L_2$. Each one of the different possible values that $(f_1(A^u), f_2(A^u))$ could take defines a sublist. Because of the linear relations, there is only one possible value that is a match for $(g_1(A^v), g_2(A^v))$ from $L_2$ (which also defines another sublist). Each sublist, associated to a different value of $(f_1(A^u), f_2(A^u))$, contains $2^{65-50} = 2^{15}$ elements from $L_1$, and the same goes for the elements in the sublists from $L_2$ associated to each value of $(g_1(A^v), g_2(A^v))$. Then we have that, for each one of the $2^{50}$ possible combinations to compute the linear relations, we can merge the 2 associated sublists (one from each list) of average size $2^{15}$ that meet this sharing.

We propose then an algorithm that, for each one of the $2^{50}$ different sublists associated to $(f_1(A^u), f_2(A^u))$, considers the only correct sublist $(g_1(A^v), g_2(A^v))$, and efficiently merges next the two remaining sublists of size $2^{15}$. The final cost of the algorithm will be $2^{50}$ times the cost of merging the two lists of $2^{15}$. Those lists are denoted by $L'_1$ and $L'_2$.

Let us point out here that for each one of the values in the sublists of size $2^{15}$, we are able to compute the yellow (resp. purple) bits depicted in figure 10 in $C^1$ by computing through $\rho$ and $\pi$. We can also compute the yellow and purple bits in $A^2$ as all the corresponding inputs belong to the same list. We can also deduce from each list the values that, xored with a value given by the other half, will determine the bits marked with $L$ ($L$ means that the associated bits depend on linear relations between both lists).

In order to reduce the cost of this merge from the trivial $2^{30}$ (given by trying all the elements in one sublist with all the elements in the other), we have to consider the relations imposed by the output known in $A^3$, that we can trace up to $B^2$.

**Merging the two lists $L'_1$ and $L'_2$ of size $2^{15}$ through parallel matching without memory.** We will recall here how this algorithm detailed in [CNV13] works, but first we have to determine the relations that we will consider for the sieving. In Figure 11, we can see some information regarding the equations to satisfy certain of the known bits of $B^2$. More precisely, we focus on the known bits $e_0$, $b_1$, $d_1$, $a_2$, $d_2$, $e_2$, $b_5$, $d_5$, $a_6$, $d_6$, $e_6$ and $c_7$. As can be seen in equations from (16) to (25), those bits (or linear combinations of them) have a small number of known variables that intervene in a non-linear way.

For the sake of readability, $x_{ijk}$ denotes the bit $C^1_{i,j,k}$ if it belongs to $L'_1$ and $y_{ijk}$ if it belongs to $L'_2$.
In the following, we explain as an example how we get the equation of $b_5$. Through $\theta$, we get that

$$b_5 = A^2_{3,3,5} + \sum_{i=0}^{4} A^2_{2,i,5} + A^2_{4,i,4} \,.$$

However, in this equation only $A^2_{4,4,4}$ has a non-linear combination of variables from one variable of each list. In other words there exist two linear functions $\ell_1$ and $\ell_2$ such that

$$b_5 = \ell_1(A^u) + \ell_2(A^v) + A^2_{4,4,4} = \ell_1(A^u) + \ell_2(A^v) + C^1_{4,4,4} + (C^1_{0,4,4} + 1)C^1_{1,4,4} \,.$$

$$C^1_{4,4,4} = x_{444} \,,\ C^1_{0,4,4} = y_{044} \text{ and } C^1_{1,4,4} = x_{144} \,.$$

We can then define two linear functions $\ell^x_{b_5}$ and $\ell^y_{b_5}$ such that

$$b_5 = \ell^x_{b_5}(A^u) + \ell^y_{b_5}(A^v) + y_{044}x_{144} \,.$$

By doing the same for the other chosen bits of $B^2$ we get the following equations.

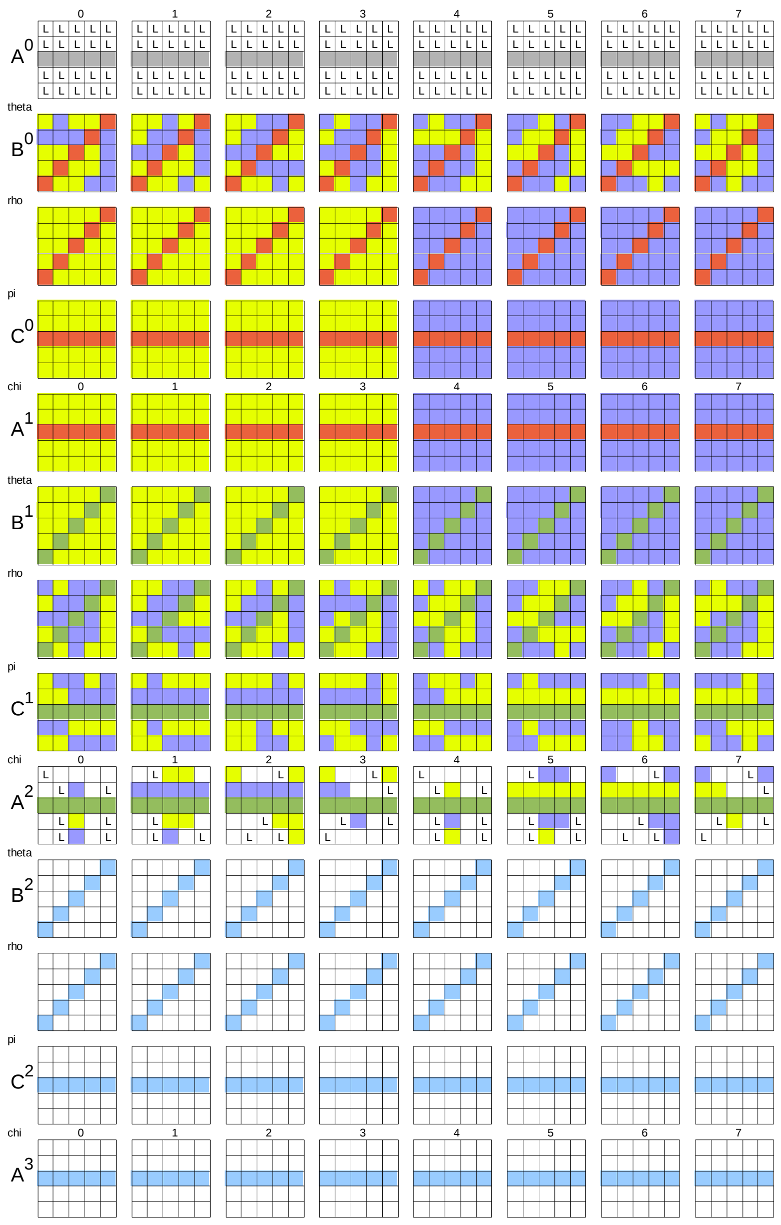

**Figure 10:** Representation of the 3-round attack. Each $5 \times 5$ square represents a slice, each small square is a bit, and each line of squares represents the full state (the 8 slices) at a certain instant. The first state that outputs a value is the one on the top, and the last the one at the bottom. The bits colored in grey, green, red and blue are known bits from the outputs. The bits colored in yellow are known bits from $A^u$ ($L_1$). Colored bits in purple are known bits from $A^v$ ($L_2$). Bits with an $L$ represent bits that can be computed as a linear combination of bits from values obtained from $A^u$ and from $A^v$. The ones in white represent a quadratic combination of both lists.

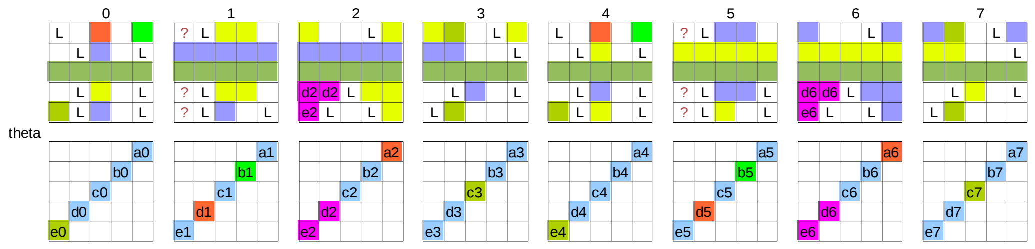

**Figure 11:** Scheme on exploited quadratic relations between the last output and the unknown bits for the final sieving

$$
\begin{aligned}
b_5 &= \ell^x_{b_5}(A^u) + \ell^y_{b_5}(A^v) + y_{044}x_{144} && (4)\\
b_1 &= \ell^x_{b_1}(A^u) + \ell^y_{b_1}(A^v) + x_{040}y_{140} && (5)\\
c_7 &= \ell^x_{c_7}(A^u) + \ell^y_{c_7}(A^v) + y_{247}x_{347} + y_{207}x_{307} && (6)\\
e_0 &= \ell^x_{e_0}(A^u) + \ell^y_{e_0}(A^v) + x_{100}y_{200} + x_{040}y_{140} + y_{247}x_{347} + y_{207}x_{307} && (7)\\
a_2 &= \ell^x_{a_2}(A^u) + \ell^y_{a_2}(A^v) + y_{141}x_{241} + y_{111}x_{211} + x_{101}y_{201} && (8)\\
d_1 &= \ell^x_{d_1}(A^u) + \ell^y_{d_1}(A^v) + y_{141}x_{241} + y_{111}x_{211} + x_{101}y_{201} + x_{340}y_{440} && (9)\\
a_6 &= \ell^x_{a_6}(A^u) + \ell^y_{a_6}(A^v) + x_{145}y_{245} + x_{115}y_{215} + y_{105}x_{205} && (10)\\
d_5 &= \ell^x_{d_5}(A^u) + \ell^y_{d_5}(A^v) + x_{145}y_{245} + x_{115}y_{215} + y_{105}x_{205} + y_{344}x_{444} && (11)\\
e_2 &= \ell^x_{e_2}(A^u) + \ell^y_{e_2}(A^v) + x_{102}y_{202} && (12)\\
d_2 &= \ell^x_{d_2}(A^u) + \ell^y_{d_2}(A^v) + y_{212}x_{312} + x_{102}y_{202} + x_{112}y_{212} && (13)\\
e_6 &= \ell^x_{e_6}(A^u) + \ell^y_{e_6}(A^v) + y_{106}x_{206} && (14)\\
d_6 &= \ell^x_{d_6}(A^u) + \ell^y_{d_6}(A^v) + x_{216}y_{316} + y_{106}x_{206} + y_{116}x_{216} && (15)
\end{aligned}
$$

By looking at the equations, we see that there are a lot of variables that appear several times in different equations. For instance the equations (5), (6) and (7) only involve 14 variables, 7 from $L'_1$ and 7 from $L'_2$. Moreover we can linearly combine the equations, for example:

$$a_2 + d_1 = x_{340}y_{440} + \ell^x_{a_2}(A^u) + \ell^y_{a_2}(A^v) + \ell^x_{d_1}(A^u) + \ell^y_{d_1}(A^v)$$

Eventually, we can also factorize the terms $y_{212}$ and $x_{216}$ in the equations (10) and (12), which gives us the following system of 10 equations that totals 21 variables of $L'_1$ and 21 variables of $L'_2$. As there are more variables with 2 quadratic terms, the complexity would only increase if we considered more. We will explain later that our obtained complexity is optimal as far as no more relations depending on one variable exist.

$$\begin{aligned}
b_1 &= x_{040}y_{140} + \ell^x_{b_1}(A^u) + \ell^y_{b_1}(A^v) && (16)\\
c_7 &= y_{247}x_{347} + y_{207}x_{307} + \ell^x_{c_7}(A^u) + \ell^y_{c_7}(A^v) && (17)\\
e_0 &= x_{100}y_{200} + x_{040}y_{140} + y_{247}x_{347} + y_{207}x_{307} + \ell^x_{e_0}(A^u) + \ell^y_{e_0}(A^v) && (18)\\
a_2 + d_1 &= x_{340}y_{440} + \ell^x_{a_2+d_1}(A^u) + \ell^y_{a_2+d_1}(A^v) && (19)\\
a_6 + d_5 &= y_{344}x_{444} + \ell^x_{a_6+d_5}(A^u) + \ell^y_{a_6+d_5}(A^v) && (20)\\
b_5 &= y_{044}x_{144} + \ell^x_{b_5}(A^u) + \ell^y_{b_5}(A^v) && (21)\\
e_2 &= x_{102}y_{202} + \ell^x_{e_2}(A^u) + \ell^y_{e_2}(A^v) && (22)\\
e_6 &= y_{106}x_{206} + \ell^x_{e_6}(A^u) + \ell^y_{e_6}(A^v) && (23)\\
e_2 + d_2 &= y_{212}(x_{312} + x_{112}) + \ell^x_{e_2+d_2}(A^u) + \ell^y_{e_2+d_2}(A^v) && (24)\\
e_6 + d_6 &= x_{216}(y_{316} + y_{116}) + \ell^x_{e_6+d_6}(A^u) + \ell^y_{e_6+d_6}(A^v) && (25)
\end{aligned}$$

**General intuition of the algorithm.** Given two lists $L'_1$ and $L'_2$, the idea of the algorithm is to find all the pairs of elements, one from each list, satisfying a certain number $(n_{aux} + n_{rem})$ of bit-relations by testing in parallel $n_{aux}$ relations and the $n_{rem}$ remaining ones. The aimed complexity is better than the naive one of testing each element with all the elements in the other list. As the considered relations are not linear, if for each elements we check all the possible matches regarding a certain number of relations, the complexity might soon become higher than the naive limit. For this we will first consider $n_{aux}$ relations, and classify the first list regarding the $v_{aux1}$ involved values. For each value, we consider all the possible matches through the $n_{aux}$ for the value of the $v_{aux2}$ involved variables from the second lists. With all the elements from the second list that satisfy these values in the $v_{aux2}$ variables, we build an *auxiliary list*, smaller than the second one, that we order by the values of the $v_{rem2}$ variables involved in the $n_{rem}$ remaining relations. This has to be done for each different value of the $v_{aux1}$ variables. We can now go back to the first sublist, of elements associated to a certain value for the $v_{aux1}$ variables. We know that these elements and the ones from the auxiliary list already satisfy the $n_{aux}$ first relations. If we now go through the different values for the $v_{rem1}$ variables involved from the first list in the $n_{rem}$ relations and check in the auxiliary list if the possible values of the $v_{rem2}$ variables appear or not, when we find a match we will find a pair of elements that satisfies both the $n_{aux}$ relations and the $n_{rem}$ relations. The intuition of the gain can be explained by imagining that the number of each group of relations and of the variables are balanced. The overhead of trying all the possible matches for one element in the other list is reduced by a square root (for similar relations). In the particular case we are dealing with here, $v_{aux1} = v_{aux2} = v_{aux}$ and $v_{rem1} = v_{rem2} = v_{rem}$. We want to point out here that the final complexity cannot be better than $|L'_1| \times |L'_2| \times 2^{-n_{aux}-n_{rem}}$, as this is the number of solutions obtained.

We want to merge $L'_1$ and $L'_2$. In order to apply the parallel matching algorithm, we have to separate these relations in two. A number of $n_{aux}$ relations (involving $v_{aux}$ variables from each list) will be considered for building the *auxiliary lists* of elements from $L'_2$. For each value of the $v_{aux}$ variables from the elements in $L'_1$, this auxiliary list will contain only the elements from $L'_2$ satisfying these $n_{aux}$ relations. The remaining $n_{rem}$ relations, involving $v_{rem}$ variables from each list, will be checked later. The parallel matching algorithm, for each one of the $2^{v_{aux}}$ different values of the $v_{aux}$ variables associated to the first $n_{aux}$ relations in $L'_1$ (that produces a sublist of $L'_1$), will perform the following:

1. Use the $n_{aux}$ relations to build the auxiliary list with the elements from $L'_2$ that are already a match with respect to these relations. About $2^{v_{aux}-n_{aux}}$ values for the

variables from the second list will be a match. Each will be composed of $|L_2'|/2^{n_{aux}}$ elements

2. Next, we reorder this auxiliary list, with respect to the $v_{rem}$ variables involved in the $n_{rem}$ remaining relations.

3. For each one of the elements in the generated sublist from $L_1'$ check, for all possible matches with respect to the $n_{rem}$ equations, if they belong to the auxiliary list. If yes, we keep the candidate and go to the next possible match, if not, go to the next possible match.

4. The final complexity cannot be smaller than the expected number of solutions.

The complexity of this algorithm will be:

$$2^{v_{aux}} \times [2^{(v_{aux}-n_{aux})+(15-v_{aux})} + 2^{15-v_{aux}} \times 2^{v_{rem}-n_{rem}}(\max(1, 2^{\frac{15-n_{aux}}{v_{rem}}}))] =$$

$$2^{15+v_{aux}-n_{aux}} + \max(2^{15+v_{rem}-n_{rem}}, 2^{30-n_{aux}-n_{rem}}).$$

This corresponds to: $2^{v_{aux}}$ is the number of sublists of $L_1'$ that we consider and therefore of auxiliary lists from the elements of $L_2'$ we have to build. For each sublist, $2^{v_{aux}-n_{aux}}$ possible values in $L_2'$ have to be checked, and $2^{15-v_{aux}}$ elements from the second list will be associated to each matched value. This leads to the first term on the complexity $2^{15+v_{aux}-n_{aux}}$, that corresponds finally to the size of the auxiliary lists, $2^{15-n_{aux}}$ that have to be built $2^{v_{aux}}$ times. Next, for each element in the first list associated to a certain value out of the $2^{v_{aux}}$ possible ones, we have $2^{15-v_{aux}}$ elements to test with respect to the $n_{rem}$ equations. The values to try in the auxiliary list are $2^{v_{rem}-n_{rem}}$. The cost for this part will be the max between $2^{15+v_{rem}-n_{rem}}$ (corresponding to when we do not always find a solution), and $2^{30-n_{aux}-n_{rem}}$ corresponding to the final number of solutions.

Let us choose for determining $v_{aux}$ and $n_{aux}$ the equations associated to $b_1, e_0, c_7$ (with 7 variables from $L_1'$ and 7 from $L_2'$) plus the ones associated to $a_2$ and $a_6$ (with 4 additional variables from each list). In this case, $n_{aux}$ is 5, $v_{aux} = 11$, and the size of the auxiliary list (that we need to build for each one of the different values of the $v_{aux}$ variables from $L_1'$) will be $2^{11-5+4} = 2^{10}$. The remaining equations filter through $n_{rem} = 5$ conditions and involve $v_{rem} = n_{rem} \times 2 = 10$ variables from each list.

The final complexity is then

$$2^{15+11-5} + \max(2^{15+10-5}, 2^{30-10}) = 2^{21} + 2^{20} = 2^{21.5}$$

**Complexities of the full attack.** As previously said, the time complexity of the full attack is given by repeating the parallel list merging algorithm for each one of the $2^{50}$ different values that determine the linear relations. We obtain then:

$$2^{50+21.5} = 2^{71.5} \text{ computations.}$$

Let us point out that each one of this computation is much smaller than a full round, so there should be a reduction factor when comparing with exhaustive search, but we leave it this way as it is the worst case for the attacker.

The memory complexity is given by the two lists of size $2^{65}$, as the auxiliary list is smaller. We do not need to add a logarithmic factor when ordering and searching the lists, as this can be done using hash tables.

In order to find one only solution, we should have one more output, otherwise we recover $2^{40}$ candidates for the internal state.

## 5 Improved attack using 4 output blocks with a rate of 32 bits

In this section we describe an improved version of the previous divide-and-conquer attacks described in section 4 that allows us to build an attack when considering a rate of $r = 32$ bits. The number of expected remaining candidates is equal to $2^{200-4\times r} = 2^{72}$ which is still lower than $2^{96}$. Moreover, similar techniques as the ones used in section 4 can be applied to mount an attack of complexity $2^{92}$.

**New ideas used for the attack.** Our attack on initial Ketje Jr with rate 32 relies on two new ideas. First, please notice that the 32 bits that are known to the adversary from each state $A^0, \ldots, A^3$ are 4 of the 5 output bits of the 8 Sboxes from the sheet $x = 0$. We show that we can partially invert such Sboxes layers and derive useful information on state $B^2$ from the known bits of $A^3$. Our second idea is that we can reduce the number of nonlinear interactions between guesses $u$ and $v$ in $B^2$ by guessing some bits of $C^1$ before starting applying divide-and-conquer algorithm.

### 5.1 Using known information from $A^0$, $A^1$ and $A^2$.

If we consider the red bits from Figure 10, that correspond to known bits of $A^1$, there is nothing to change from the previous attack with a rate $r = 40$, but the size of the two lists increases by a factor $2^4$ for both lists : half of the 8 bits that are now unknown are located in parts of the state $A^u$ and $A^v$, and need to be guessed while building our lists.

The 32 known bits from $A^0$ can be computed as linear relations between the bits of the two lists. Hence there is no longer 40 linear relations that are sieving in the merging lists algorithm, but only 32.

In our attacks with a rate 40, we need to recover 40 bits of $B^1$ from the 40 known bits of $A^2$. To transpose this information to a rate of 32, we guess the 8 missing bits before starting our attack and apply the same strategy, such that the green part can sieve the same information on the 2 lists. However, this increases the complexity of the attack by a factor $2^8$ but reduces by $2^4$ the size of both lists.

### 5.2 Linear relations derived from $A^3$.

The problem that we aim at solving is that we can no longer invert the application of $\chi$ between $C^2$ and $A^3$. Moreover, guessing 8 more bits will cost us too much. However, as only 1 known bit is missing for each row we want to propagate backwards through $\chi$ that we want to invert, there are exactly 32 linear equations between the bits in $C^2$ and the known bits of $A^3$. Hence, we do not know each bit in blue from state $B^2$ on Figure 10 independently, but we know exactly 32 independent linear equations. It is important to know that those equations depend on the value of the observed value in $A^3$. In other words, as there is only 1 bit missing per slice (per $\chi$), each bit taken as input of $\chi$ depends linearly from those missing values. Considering a rate of 32 and the specifications of Ketje Jr, the known bits of $A^3$ are the following ones:

$$\{A^3_{2,2,*}, A^3_{3,2,*}, A^3_{4,2,*}, A^3_{0,2,*}\}$$

All the bits $A^3_{1,2,*}$ are unknown. For sake of readability, we note $A^3_{1,2,i} = x_i$, for all $0 \le i \le 7$, and all the known bits of $A^3$ will be denoted by $\alpha_j$ for $0 \le j \le 31$. We note $(\alpha||X) = (\alpha_0, ..., \alpha_{31}, x_0, ..., x_7)^\top$ and we note $\beta$ the vector that corresponds to the 40 bits (blue) in $C^2$. Then there exists a matrix $M$ of size $40 \times 8$ and a constant vector $\beta_0$ such that $\beta = MX + \beta_0$. It is important to notice that $M$ depends on the values $\alpha_j$ for $0 \le j \le 31$, but these values are known to the adversary. Moreover, the applications

$\pi$ and $\rho$ are bit transpositions, hence, by using the same notations as in figure 11, we can assume that there exists a $40 \times 8$ matrix $M'$ and a constant vector $\beta_0'$, such that $(a_0, ..., e_0, a_1, ..., e_6, a_7, ..., e_7)^\top = M'X + \beta_0'$.

## 5.3 Guessing 20 bits from $C^1$

However, the bits $a_0, ..., e_0, a_1, ..., e_6, a_7, ..., e_7$ cannot be expressed as a the sums of linearly independent expressions of bits in $A^u$ and bits in $A^v$. To solve this issue, we guess 20 bits (10 in each list), such that we can get some linear equations. More precisely, we fix all bits that correspond to $C^1_{0,5,0}$, $C^1_{3,5,0}$, $C^1_{1,5,1}$, $C^1_{1,1,1}$, $C^1_{3,5,2}$, $C^1_{2,1,2}$, $C^1_{4,5,3}$, $C^1_{5,4,3}$, $C^1_{0,0,3}$, $C^1_{3,0,3}$, $C^1_{0,5,0}$, $C^1_{3,5,0}$, $C^1_{1,5,1}$, $C^1_{1,1,1}$, $C^1_{3,5,2}$, $C^1_{2,1,2}$, $C^1_{4,5,3}$, $C^1_{5,4,3}$, $C^1_{0,0,3}$, $C^1_{3,0,3}$. We guess all those bits because for each of those bits $C^1_{x,y,z}$, the bits $C^1_{x-1,y,z}$ and $C^1_{x+1,y,z}$ belong to the same list. Hence guessing those bits drastically decreases the number of quadratic variables on the equations we consider for the bits $a_0, ..., e_0, a_1, ..., e_6, a_7, ..., e_7$. The details of this part is depicted in figure 12.

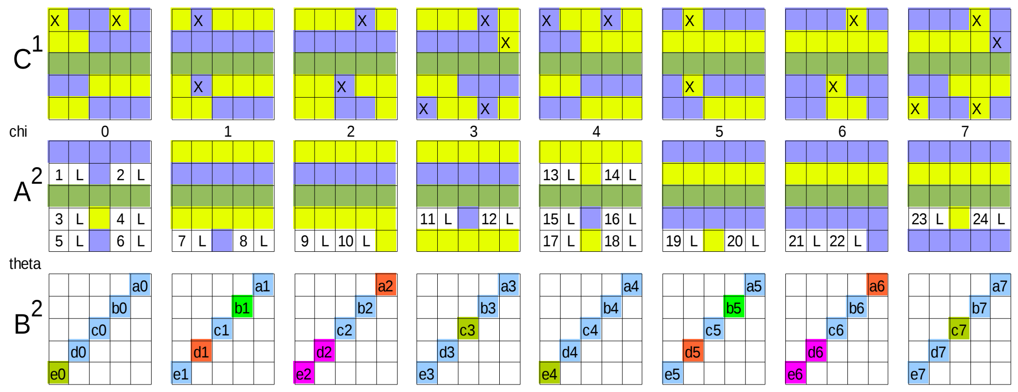

**Figure 12:** The bits with X in $C^1$ are the 20 bits we guess, hence only 24 quadratic variables remain (denoted with 1, 2,...,24).

Hence, we get 40 equations for the bits $a_0, ..., e_0, a_1, ..., e_6, a_7, ..., e_7$, where 24 quadratic variables occur. Each quadratic variable appears during the $\chi$ function between $C^1$ and $A^2$: as soon as two adjacent bits on the same row belong to the two different sets of our divide-and-conquer division of the state, $A^2$ involves the product of these bits, which can be viewed as a new variable (referred to as quadratic). Hence, we can derive at least 16 linearly independent expressions that do not imply quadratic variables, but in fact there is exactly 20 of them. In the following we note $q_i$, for $1 \leq i \leq 24$ the quadratic variables depicted in figure 12. Hence the following equations hold.

$$\begin{array}{ll}
a_0 = q_2 + q_4 + q_6 + q_{23} + l_{a_0}(L_1, L_2) & a_1 = q_1 + q_3 + q_5 + q_8 + l_{a_1}(L_1, L_2) \\
b_0 = q_2 + l_{b_0}(L_1, L_2) & b_1 = l_{b_1}(L_1, L_2) \\
c_0 = q_{24} + l_{c_0}(L_1, L_2) & c_1 = q_2 + q_4 + q_6 + l_{c_1}(L_1, L_2) \\
d_0 = q_1 + q_3 + q_5 + l_{d_0}(L_1, L_2) & d_1 = q_7 + l_{d_1}(L_1, L_2) \\
e_0 = q_5 + l_{e_0}(L_1, L_2) & e_1 = q_7 + l_{e_1}(L_1, L_2)
\end{array}$$

$$\begin{array}{ll}
a_2 = q_7 + l_{a_2}(L_1, L_2) & a_3 = q_9 + q_{12} + l_{a_3}(L_1, L_2) \\
b_2 = q_{10} + l_{b_2}(L_1, L_2) & b_3 = l_{b_3}(L_1, L_2) \\
c_2 = q_8 + l_{c_2}(L_1, L_2) & c_3 = l_{c_3}(L_1, L_2) \\
d_2 = q_9 + l_{d_2}(L_1, L_2) & d_3 = q_{10} + q_{11} + l_{d_3}(L_1, L_2) \\
e_2 = q_9 + l_{e_2}(L_1, L_2) & e_3 = l_{e_3}(L_1, L_2)
\end{array}$$

$$\begin{array}{ll}
a_4 = q_{11} + q_{14} + q_{16} + q_{18} + l_{a_4}(L_1, L_2) & a_5 = q_{13} + q_{15} + q_{17} + q_{20} + l_{a_5}(L_1, L_2) \\
b_4 = q_{14} + l_{b_4}(L_1, L_2) & b_5 = l_{b_5}(L_1, L_2) \\
c_4 = q_{12} + l_{c_4}(L_1, L_2) & c_5 = q_{14} + q_{16} + q_{18} + l_{c_5}(L_1, L_2) \\
d_4 = q_{13} + q_{15} + q_{17} + l_{d_4}(L_1, L_2) & d_5 = q_{19} + l_{d_5}(L_1, L_2) \\
e_4 = q_{17} + l_{e_4}(L_1, L_2) & e_5 = q_{19} + l_{e_5}(L_1, L_2)
\end{array}$$

$$\begin{array}{ll}
a_6 = q_{19} + l_{a_6}(L_1, L_2) & a_7 = q_{21} + q_{24} + l_{a_7}(L_1, L_2) \\
b_6 = q_{22} + l_{b_6}(L_1, L_2) & b_7 = l_{b_7}(L_1, L_2) \\
c_6 = q_{20} + l_{c_6}(L_1, L_2) & c_7 = l_{c_7}(L_1, L_2) \\
d_6 = q_{21} + l_{d_6}(L_1, L_2) & d_7 = q_{22} + q_{23} + l_{d_7}(L_1, L_2) \\
e_6 = q_{21} + l_{e_6}(L_1, L_2) & e_7 = l_{e_7}(L_1, L_2)
\end{array}$$

The 20 linear equations that we get correspond then to $a_0 + c_1 + b_6 + d_7$, $c_0 + a_7 + e_6$, $d_0 + a_1 + c_2$, $b_1$, $d_1 + e_1$, $d_1 + a_2$, $d_2 + e_2$, $b_2 + d_3 + a_4 + c_5$, $d_2 + a_3 + c_4$, $b_3$, $c_3$, $e_3$, $d_4 + a_5 + c_6$, $b_5$, $d_5 + e_5$, $d_5 + a_6$, $d_6 + e_6$, $b_7$, $c_7$, $e_7$.

**Sieving with 12 linear equations.** We have exactly 20 independent linear equations between the two lists and the vector space of dimension 40 that corresponds to $a_0, b_0, ..., e_7$. We note $\beta' = (a_0, ..., e_0, a_1, ..., e_6, a_7, ..., e_7)^\top$. Moreover, we know that there exist a matrix $M'$ and a vector $\beta_0$ such that $\beta' = M'X + \beta_0$. Hence, there exist $(l_i)_{1 \leq i \leq 20}$ and $(l'_i)_{0 \leq i \leq 20}$ 40 linear equations such that $l_i(L_1 + L_2) = l'_i(\beta') = l'_i \circ M'X + l'_i(\beta_0)$, for all $1 \leq i \leq 20$. Eventually, we apply a Gauss pivot on the 8 first equations such that we can eliminate the unknown values of $x_0, ..., x_7$, then we get at least 12 linear equations between the 2 lists.

**Guessing 4 more bits.** Now we have guessed 28 bits in total. The size of the lists becomes now $2^{100} \times 2^{-16} \times 2^5 \times 2^{-20} \times 2^{-10} = 2^{59}$. $2^{100}$ is half of the state, $2^{-16}$ corresponds to the known bits in $A^1$, $2^5$ corresponds to the parity bits of the columns, $2^{-20}$ corresponds to the known bits in $A^2$ (8 are guessed) and $2^{-10}$ corresponds to half of the bits we guess to get linear equations. The final cost of the algorithm is then $(2^{28} \times 2^{59}) + 2^{28} \times 2^{2\times 59} \times 2^{-32} \times 2^{-10} \times 2^{-12}$, which is $2^{87} + 2^{92}$. Both lists are of size $2^{59}$ and we have 54 linear relations, hence it is of interest to guess few bits more such that both terms become equal.

## 6 Conclusion

These attacks do not pose a threat to Ketje Jr instantiated with the recommended parameters. In particular, the attacks against 4 consecutive output blocks do not work with the twisted permutation, and therefore, the tweak seems to make the primitive more resistant to divide-and-conquer attacks. However, our attacks provide us with a new non-trivial limit on the rate we can output.

## Acknowledgements

We would like to thank Joan Daemen for his many valuable comments that have considerably improved this paper, and also Henri Gilbert, Anne Canteaut, Christina Boura and Colin Chaigneaux for many useful discussions. This work was partially supported by the French Agence Nationale de la recherche through the BRUTUS project under contract ANR-14-CE28-0015. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement no. 714294 - acronym QUASYModo).

## References

[BCC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of keccak and *Luffa*. In Joux [Jou11], pages 252–269.

[BDP$^+$df] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Ketje v1. *Submission to Caesar competition*, 2014. https://competitions.cr.yp.to/round1/ketjev1.pdf.

[BDP$^+$df] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Ketje v2. *Submission to Caesar competition*, 2016. http://competitions.cr.yp.to/round3/ketjev2.pdf.

[BDPA08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

[BDPA12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. In *DIAC*, 2012.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.

[BGS11] Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors. *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*. Springer, 2011.

[BR10] Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In Biryukov et al. [BGS11], pages 229–240.

[CNV13] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-middle: Improved MITM attacks. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 222–240. Springer, 2013.

[DDKS12] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 719–740. Springer, 2012.

[DGPW12] Alexandre Duc, Jian Guo, Thomas Peyrin, and Lei Wei. Unaligned rebound attack: Application to keccak. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 402–421. Springer, 2012.

[DLWQ17] Xiaoyang Dong, Zheng Li, Xiaoyun Wang, and Ling Qin. Cube-like attack on round-reduced initialization of ketje sr. *IACR Trans. Symmetric Cryptol.*, 2017(1):259–280, 2017.

[DSP07] Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved meet-in-the-middle attacks on reduced-round DES. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 249–274, 2016.

[JN15] Jérémy Jean and Ivica Nikolic. Internal differential boomerangs: Practical analysis of the round-reduced keccak- f f permutation. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 537–556. Springer, 2015.

[Jou11] Antoine Joux, editor. *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*. Springer, 2011.

[KNRS10] Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of *Luffa* v2 components. In Biryukov et al. [BGS11], pages 388–409.

[LMR$^+$09] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound distinguishers: Results on the full whirlpool compression function. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2009.

[LN15] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of full sprout. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 663–682. Springer, 2015.

[Nay11] María Naya-Plasencia. How to improve rebound attacks. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 188–205. Springer, 2011.

[Sas11] Yu Sasaki. Meet-in-the-middle preimage attacks on AES hashing modes and an application to whirlpool. In Joux [Jou11], pages 378–396.