# A Framework for Automated Independent-Biclique Cryptanalysis

Farzaneh Abed, Christian Forler, Eik List(✉),
Stefan Lucks, and Jakob Wenzel

Bauhaus-Universität Weimar, Weimar, Germany
{farzaneh.abed,christian.forler,eik.list,
stefan.lucks,jakob.wenzel}@uni-weimar.de

**Abstract.** In this paper we introduce Janus, a software framework – written in Java – which is built to provide assistance in finding independent-biclique attacks for a user-chosen set of parameters, e.g., the number of rounds and dimension of the biclique. Given a certain cipher, Janus not only finds an optimal bipartite graph (biclique), but also provides an all-round carefree package of finding an optimal matching-with-precomputation step, rendering the found biclique, and determining the computational complexity of the attack.

We have used the Janus framework to verify existing results on ARIA and the AES. Additionally, by using this framework, we could find the first full-round biclique attacks on all versions of the AES-like cipher BKSQ.

**Keywords:** Automated cryptanalysis · Biclique · BKSQ

## 1 Introduction

**Overview.** Biclique cryptanalysis was first introduced by Khovratovich *et al.* in 2011 [17] and presented at the FSE 2012 [18]. The authors used this approach to find preimages for reduced-round versions of the block cipher based hash functions Skein [12] and SHA-2 [21]. Bicliques represent an improvement of the splice-and-cut approach [4,22,23], which itself is a variant of meet-in-the-middle attacks. More detailed, biclique cryptanalysis uses a complete bipartite graph (biclique), which can be constructed over a part of a primitive, to extend an existing meet-in-the-middle or similar attack. While the splice-and-cut approach was intentionally designed to target hash functions, Wei *et al.* presented the first splice-and-cut attacks on the block cipher KTANTAN [28]. Bogdanov *et al.* then adapted biclique-based attacks on the AES [5]. Their work obtained a high level of attention, since they demonstrated the first single-key attacks on all full versions of the AES with a significant advantage over exhaustive search. Since then, biclique attacks have become a well-known technique and attacks on several further ciphers have been published in [1–3,7,8,13–15,20,24,26,27].

S. Moriai (Ed.): FSE 2013, LNCS 8424, pp. 561–581, 2014.
DOI: 10.1007/978-3-662-43933-3_29, © Springer-Verlag Berlin Heidelberg 2014

Finding good (independent) bicliques over a given number of rounds is a time-consuming task which requires in-depth knowledge of the investigated cipher to find well-suited differentials. Thus, it is adequate to think about using a computer to find such bicliques. Usually, implementations of common block cipher APIs are not designed to provide a sufficiently fine granularity, e.g., access to single steps and the basic operations of the cipher is not supported, but required to find good bicliques.

**Our Contribution.** A unified API is needed to reduce the effort of modifying a block-cipher implementation for the biclique search. In addition, such an API would allow applying one single biclique-searching framework that fits all. In this paper, we present such a framework, called `Janus`, which is open source and free to use[1]. The main feature of `Janus` is to find a complete and independent bipartite graph for a certain number of given rounds. In addition, it computes the corresponding step of matching with precomputations, and the overall complexity. Finally, it supports rendering a graphical illustration of the found biclique and the matching part.

`Janus` provides a highly modular and flexible API, i.e., it allows the user to determine parameters like the used cryptographic primitive, the starting/ending round, the dimension of the biclique, the starting difference, etc.

First, we used our framework to verify and validate published attacks on variants of the AES and ARIA (see Sect. 4). Thereby, we detected a flaw in the complexity computation of the attack on AES-192. Thus, we were not able to verify the claim made for this attack. But secondly, further analysis revealed that the authors just forgot to include one round during the matching-with-precomputation phase. This example points out the importance of an automated framework to validate claims for existing attacks.

Additionally, we used `Janus` to find the first full-round attacks on variants of the AES-like cipher BKSQ [10]. Results of our work can be found in Sect. 4 in Table 1.

**Related Work.** There are several published tools and frameworks which support certain cryptanalytic techniques. Though, these frameworks are mostly limited to a very specific area of application. For example, the work of Daemen and Van Assche[2] concentrates only on analyzing their SHA-3 winner Keccak [9]. They provide, among other things, a computation of linear and differential trails. Another framework was introduced by Leurent [19] to analyze ARX-based hash functions (like Skein or Blake) with the goal to assist in finding good differential trails. Further, Stankovski implemented an automated algebraic cryptanalysis framework [25], which uses the Maximum-Degree-Monomial (MDM) test to launch algebraic attacks against stream and block ciphers. Currently, it supports more than 20 stream and block ciphers, and provides a possibility to produce TeX code for graphs.

[1] https://github.com/janus-framework/janus

[2] http://keccak.noekeon.org/KeccakTools-doc/ [April 2013]

**Outline.** In Sect. 2 we will provide a brief introduction of biclique cryptanalysis. In Sect. 3 we introduce `Janus` – containing the search for bicliques, the matching phase, and the rendering option. We used our framework to verify existing attacks on the AES and ARIA, as well as to mount new attacks on BKSQ. Our results are shown in Sect. 4. Section 5 concludes the paper.

## 2 Independent-Biclique Cryptanalysis

In this section we review the basics of independent-biclique cryptanalysis following the work of [17]. A biclique is a complete bipartite graph which covers some steps of a given cipher. It connects every element in a set of starting states $\mathcal{S}$ with every element in a set of ending states $\mathcal{C}$. We enumerate the elements in $\mathcal{S}$ by $S_j$ and the elements in $\mathcal{C}$ by $C_i$, where a path from $S_j$ to $C_i$ represents the encryption under a key $K[i,j]$. More formally, the 3-tuple of sets $[\{S_j\}, \{C_i\}, \{K[i,j]\}]$ is called a *d-dimensional biclique*, if

$$\forall i, j \in \{0, \ldots, 2^d - 1\} : \quad S_j \xrightarrow[\mathcal{B}]{K[i,j]} C_i,$$

where $\mathcal{B}$ denotes the steps of the cipher covered by the biclique. The basic idea is to divide the key space into $2^{k-2d}$ groups of $2^{2d}$ keys, where $k$ denotes the length of the secret key and $d$ is the dimension of the biclique. A biclique can then be defined for one such group of keys $K[i,j]$, where the individual keys are represented relative to a so-called *base key* of the group, $K[0,0]$, and two differences $\Delta_i^K$ and $\nabla_j^K$:

$$K[i,j] = K[0,0] \oplus \Delta_i \oplus \nabla_j.$$

An adversary can construct a biclique over one part of a cipher and apply then a meet-in-the-middle or similar attack over the remaining parts.

### 2.1 Independent Bicliques

In [5,16,17], Khovratovich *et al.* proposed two different paradigms for biclique attacks: *bicliques from independent differential trails* (or *independent bicliques*) and *bicliques from interleaving differential trails* (or *long bicliques*). Independent bicliques allow the construction of bicliques from two sets of differentials:

1. In the beginning, the adversary chooses a so-called *base computation*, i.e., a 3-tupel $\{S_0, C_0, K[0,0]\}$, where the key $K[0,0]$ maps the internal state $S_0$ to the state $C_0$ over $\mathcal{B}$:
$$S_0 \xrightarrow[\mathcal{B}]{K[0,0]} C_0.$$
2. Then, it chooses $2^d$ differences $\Delta_i^K$, derives new keys $K[i,0] = K[0,0] \oplus \Delta_i^K$, perfoms $2^d$ computations from the state $S_0$ in forward direction and arrives at $2^d$ states $C_i$:
$$S_0 \xrightarrow[\mathcal{B}]{K[0,0] \oplus \Delta_i^K} C_0 \oplus \Delta_i = C_i \quad \forall\, i \in \{0, \ldots, 2^d - 1\}.$$
These are called the $\Delta_i$-differentials.

3. Similarly, it chooses $2^d$ further differences $\nabla_j^K$, again derives new keys $K[0,j] = K[0,0] \oplus \nabla_j^K$, computes $2^d$ times from the state $C_0$ in backward direction, and arrives at $2^d$ states $S_j$:
$$S_j = S_0 \oplus \nabla_j \xleftarrow[\mathcal{B}^{-1}]{K[0,0]\oplus\nabla_j^K} C_0 \quad \forall j \in \{0,\ldots,2^d-1\}.$$
These are called the $\nabla_j$-differentials.

If all $\Delta_i$-differentials *do not share any active non-linear operations* with the $\nabla_j$-differentials, then every state $S_j$ can be connected with every state $C_i$ by encrypting $S_j$ under the key $K[i,j] = K[0,0] \oplus \Delta_i^K \oplus \nabla_j^K$. Thus, one obtains a set of $2^{2d}$ *independent* $(\Delta_i,\nabla_j)$*-differential trails*:

$$S_0 \oplus \nabla_j \xrightarrow[\mathcal{B}]{K[0,0]\oplus\Delta_i^K\oplus\nabla_j^K} C_0 \oplus \Delta_i \quad \forall i,j \in \{0,\ldots,2^d-1\}.$$

The length of the biclique differentials is limited by two full diffusions of the cipher. An adversary can potentially create bicliques over more rounds by using the long-biclique approach. Though, the construction of long bicliques is quite sophisticated and requires a significantly higher computational effort. More importantly, the requirement for independent differentials is a very clear and well-understood criterion that allows us to test it by using an automated approach. Therefore, we focus on the independent-biclique approach in this work.

### 2.2 Matching-with-Precomputations

If a constructed biclique is quite short and the matching part needs to cover too many rounds, then a meet-in-the-middle attack may no longer be applicable. In such cases, [5] proposed an alternative procedure called *matching-with-precomputations.*

Assume an adversary is given a cipher $E$ which can be split into three parts $E = \mathcal{B} \circ E_2 \circ E_1$, where $E_1$ is the subcipher that maps a plaintext $P$ to an internal state $V$, $E_2$ maps $V$ to another internal state $S$, and $\mathcal{B}$ maps the state $S$ to the ciphertext $C$:
$$P \xrightarrow{E_1} V \xrightarrow{E_2} S \xrightarrow{\mathcal{B}} C.$$

After constructing a biclique over $\mathcal{B}$, the adversary is given $2^d$ states $C_i$, and obtains the corresponding plaintexts $P_i$ from a decryption oracle. Then, it performs $2^d$ forward computations from the plaintexts $P_i$ to $\overrightarrow{V_{i,0}}$,

$$P_i \xrightarrow[E_1]{K[i,0]} \overrightarrow{V_{i,0}},$$

and stores the $2^d$ values $\overrightarrow{V_{i,0}}$. Similarly, it performs $2^d$ backward computations from the states $S_j$ to $\overleftarrow{V_{0,j}}$,

$$\overleftarrow{V_{0,j}} \xleftarrow[E_2^{-1}]{K[0,j]} S_j,$$

and stores the $2^d$ values $\overleftarrow{V_{0,j}}$. These two steps are called the *precomputations.* In the following, the adversary re-uses the stored values for the remaining $2^{2d} - 2^d$ computations

$$P_i \xrightarrow[E_1]{K[i,j]} \overrightarrow{V_{i,j}}, \quad \text{and} \quad \overleftarrow{V_{i,j}} \xleftarrow[E_2^{-1}]{K[i,j]} S_j,$$

where it recomputes only those parts of the key schedule and the round transformation that differ from the stored values. By using this method, one can reduce the computational effort significantly even if no attacks are known to cover the remaining parts of the cipher. The recomputation costs can be further reduced by only matching in a part of $V$ (*partial matching*).

### 2.3 Complexity Calculation

For every biclique, the adversary tests $2^{2d}$ keys. Hence, it needs to construct $2^{k-2d}$ bicliques to cover the full key space. For the time complexity, [5] proposed the equation:

$$C_{full} = 2^{k-2d} \left( C_{biclique} + C_{decrypt} + C_{precomp} + C_{recomp} + C_{falsepos} \right), \qquad (1)$$

where

- $C_{biclique}$ denotes the costs for computing $2 \cdot 2^d$ trails over $\mathcal{B}$,
- $C_{decrypt}$ is the complexity of the oracle to decrypt $2^d$ ciphertexts,
- $C_{precomp}$ represents the effort for $2^d$ computations of $E_1$ to determine $\overleftarrow{V_{0,j}}$ and $2^d$ computations of $E_2^{-1}$ to determine $\overrightarrow{V_{i,0}}$,
- $C_{recomp}$ describes the costs of recomputing $2^{2d}$ values $\overleftarrow{V_{i,j}}$ and $\overrightarrow{V_{i,j}}$, and
- $C_{falsepos}$ is the complexity to eliminate false positives.

The full computational effort of the attack is dominated by the recomputations. The memory requirements are upper bounded by storing $2^d$ intermediate states $V_{i,j}$.

## 3 Framework Design

Our current implementation consists of four components:

1. The `biclique search` subsystem is responsible for searching for independent differential trails over some sub-cipher $\mathcal{B}$ of a given primitive $E$.
2. Given a found biclique, the `matching` subsystem analyzes the remaining parts of the cipher to find a matching which leads to an attack with a minimal computational effort.

3. The `rendering` subsystem can visualize bicliques as well as matching phase differentials in PDF format, using the community version 5.3.0 of the open-source library iText [6].
4. Moreover, the framework contains a number of common components, such as cipher implementations, serialization and utility classes, as well as cipher-dependent helper classes which generate and compare differentials.

In this work we concentrate on describing the two major components in detail.

### 3.1 Biclique Search

The task of finding independent bicliques can be transformed into the task of finding pairs of independent differentials $(\Delta^f, \nabla^b)$. In advance, the user needs to specify:

- a target cipher $E$,
- the round range of the sub-cipher $\mathcal{B}$,
- the dimension of bicliques $d$,
- a strategy to test the independency of differentials,
- and a strategy to define and generate round key differences.

The general biclique search follows the steps from Sect. 2.1. Assume that $\mathcal{B}$ covers the rounds $[r, s]$ with $1 \leq r \leq s \leq N_r$ of a given cipher $E$, where $N_r$ is the total number of rounds in $E$. We denote

- by $N_r^{\mathcal{B}} = s - r + 1$ the number of rounds covered by $\mathcal{B}$,
- by $T_i$ the state after Round $i$,
- by $U_i$ the intermediate state after the non-linear operation in Round $i$,
- and by $K_i$ the round key of Round $i$.

We further call the state of the cipher's key register, which contains the key for Round $r$, the starting key, and the state which contains the key for Round $s$ the ending key.

First, we fix $K[0,0]$ and $S_0$ to and derive $C_0$. This base computation is computed only once for a given cipher and round interval. We then create a trail $\Delta^f$ which will store all state values $T_i$, all intermediate state values $U_i$, as well as all round keys $K_i$ which are used in $\mathcal{B}$. At the beginning, we initialize them with all-zero values. Then, we choose a starting key difference $\Delta^{K^f}$ with $d$ bits set. In the following, we iterate over all $2^d$ possible values for the $d$ set bits in $\Delta^{K^f}$, and compute $2^d - 1$ differential trails

$$S_0 \xrightarrow[\mathcal{B}]{K[0,0] \oplus \Delta_i^{K^f}} C_i^f, \quad \forall\, i \in \{1, \ldots, 2^d - 1\}.$$

We denote by $\Delta_i^f$ the resulting differences between the corresponding states, intermediate states and round keys of the trail $\Delta^f$ and the base computation:

$$\Delta_i^f = \left( S_0 \xrightarrow[\mathcal{B}]{K[0,0]} C_0 \right) \oplus \left( S_0 \xrightarrow[\mathcal{B}]{K[0,0] \oplus \Delta_i^{K^f}} C_i^f \right).$$

Bits which are active in any of the $2^{d-1}$ differential trails $\Delta_i^f$ should remain active in the differential $\Delta^f$. Thus, the $\Delta_i^f$-trails are accumulated to $\Delta^f$ by applying the logical OR pair-wise to all corresponding state and round key differences of all differentials $\Delta_i^f$:

$$\Delta^f \leftarrow \bigvee_{i=1}^{2^d-1} \Delta_i^f.$$

This procedure is repeated for in total $N_d$ unique starting key differences $\Delta^{K^f}$, $\forall f \in \{1, \ldots, N_d\}$. All $N_d$ accumulated forward trails $\Delta^f$ are stored in a list. The $N_d$ backward trails $\nabla^b$ are computed similarly afterwards.

For every pair of differentials $(\Delta^f, \nabla^b)$, we check if any of their corresponding states or round keys share active parts in non-linear operations. If not, the current pair yields an independent biclique. Since any identified biclique can be used to mount an attack, we provide an option for the early abort as soon as the first such pair has been found. The time complexity of the biclique search process is given by

$$C_{time} = C_{forward} + C_{backward} + C_{testing},$$

where

- $C_{forward}$ is given by constructing $N_d$ $\Delta$-differentials,
- $C_{backward}$ denotes the effort of constructing $N_d$ $\nabla$-differentials,
- and $C_{testing}$ represents the costs for comparing $N_d^2$ pairs of differentials $(\Delta, \nabla)$.

The complexity is dominated by the effort for testing $N_d^2$ pairs of differentials. We have to store the states and round keys of $N_d$ forward differentials, where every differential holds $N_r^{\mathcal{B}} + 1$ (from $r-1$ to $s$) state differences, $N_r^{\mathcal{B}}$ (from $r$ to $s$) intermediate state differences, and a cipher-dependent number of $N_k$ round key differences, since $E$ may employ pre- and post-whitening keys. Hence, we need to store

$$C_{memory} = N_d \cdot (2N_r^{\mathcal{B}} + 1) \cdot n + N_k \cdot k$$

bits, where $n$ and $k$ denote the state and round-key size, respectively. In the case when the available memory is not sufficient to store all forward differentials, the biclique search is performed in iterations.

**Ciphers.** Throughout the framework we employ a unified interface for cipher implementations. Standard implementations allow the client to specify only the plaintext, the used key and, in some cases, a tweak. The implementations in our framework have to provide access also to internal values, such as intermediate states to allow the comparison of state differences.

In addition, they have to provide access to the values of round keys as well as to their internal key register. To obtain the longest possible independent bicliques, one should not minimize the number of active bits with respect to the secret key. Since the key schedule of most ciphers provides a significant diffusion, it would increase the number of affected bits in the round-key differences $\Delta_i^K$ or $\nabla_j^K$ and hence, would increase the number of active bits in the differential trails. Instead, one should choose key differences which have a minimum number of active bits in the round keys at the beginning (for $\Delta$-differentials) or at the end (for $\nabla$-differentials) of $\mathcal{B}$, respectively. This minimizes the number of active bits in non-linear operations of the differential trails through $\mathcal{B}$. Thus, the starting point for choosing key differences should be an intermediate state of the cipher's key register, from where one can derive the differences for all further round keys. The ciphers we are interested in utilize a key register which is updated in an iterated reversible procedure, with the consequence that the secret key can be reconstructed from any given register state. Our implementations specify if the key schedule of a cipher is reversible. In this case – which applies to most AES-like primitives and modern lightweight ciphers – they provide a method which allows to invert the key schedule given an arbitrary $k$-bit state of the key register at a certain number of iterations. In the opposite case, the starting key differences are injected in the secret key as a fallback solution.

**Starting Key Differences.** The number of tested differentials, $N_d$, depends on the dimension of the biclique $d$ and the size of the key register $k$. Given $k$ and $d$, one could potentially generate $N_d = \binom{k}{d}$ forward and backward differentials, which becomes infeasible for $k \geq 64$. Though, this effort can be reduced significantly for byte- and nibble-wise operating ciphers. In the following, we consider three strategies to generate key differences for such primitives, which are illustrated in Fig. 1.

1. Firstly, one can set only a minimum number of $d$ active bits in the starting key difference. Then, for byte-wise operating primitives, there are only $\frac{k}{\lceil d/8 \rceil}$

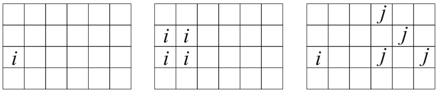

**Fig. 1.** Approaches to iterate over key differences for byte-wise/nibble-wise operating ciphers: iterate over a minimum number of active bytes/nibbles (left), over multiple bytes/nibbles with equal value (middle), or choose user-defined differences over a part of the key to cancel out results of the round transformation (right).

active bytes in the difference. As a consequence, for byte- and nibble-wise primitives the number of possible differences which can be tested reduces to

$$N_d = \binom{k/8}{\lceil d/8 \rceil} \quad \text{and} \quad N_d = \binom{k/4}{\lceil d/4 \rceil}$$

differentials, respectively. For bit-wise operating primitives, one can limit the number of generated key differences to a user-definable number.

2. Secondly, one can set the same difference for multiple nibbles/bytes in the starting key difference. At the first sight, these will produce additional active bytes in the state after a key injection, making it harder for the differential to be independent in a pair. At second sight, the additional active bytes may cancel out byte differences in the key schedule and/or the round transformation of AES-like ciphers, as we can learn from the attack on SQUARE by Mala [20]. Though, this strategy increases the number of tested keys to $N_d = 2^{k/8}$ for byte-wise and $N_d = 2^{k/4}$ for nibble-wise primitives, respectively.
3. Alternatively, one can employ custom rules to generate round-key differences. In their attack on AES-192, Bogdanov *et al.* employed the inverse result of a `MixColumns` operation as a part of the round key difference [5]. And in their attack on ARIA-256 [8], the authors used dedicated differences in which the right half of the 256-bit key canceled the difference injected by the left half. One can learn from those examples that cipher-specific key differences can result in longer bicliques for AES-like ciphers. Since testing all custom differences in the key space is infeasible, the task of choosing "good" custom starting key differentials can be left to the user.

### 3.2 Matching

A matching-with-precomputations step is supposed to be applied to the subciphers not covered by a given biclique (here $E_2 \circ E_1$). Our framework can help to identify a well-suited matching by investigating two aspects: first, it tests all possible rounds which can be used to locate $V$:

$$P \xrightarrow[E_1]{} V \xleftarrow[E_2^{-1}]{} S,$$

and second, it tests all possible nibbles or bytes in $V$ which can be used for a partial matching. For every round $r$ that can be used to locate $V$, we perform four steps:

1. First, we compute differentials from the start and the end of the matching part to the middle:

$$P \xrightarrow[E_1]{K[0,0] \oplus \nabla_j^K} V_r \oplus \nabla_j^V \qquad \text{and} \qquad V_r \oplus \Delta_i^V \xleftarrow[E_2^{-1}]{K[0,0] \oplus \Delta_i^K} S.$$

Note that these differential trails result from injecting differences in the round keys.

2. Then, for every nibble/byte in $V$, we create a new difference $\delta^V$ in which the bits that are used for a partial matching are set. We compute the differentials from $V$ to start and end:

$$P \oplus \delta^P \xleftarrow[E_1^{-1}]{K[0,0]} V_r \oplus \delta^V \qquad \text{and} \qquad V_r \oplus \delta^V \xrightarrow[E_2]{K[0,0]} S \oplus \delta^S.$$

   These active bits in these trails represent the parts of the states and round keys that have to be known in order to apply the partial matching.
3. For the recomputation effort of an attack, one has to consider only those parts of the states and round keys that are active in both differential trails: $0 \rightarrow \nabla_j^V$ and $\delta^P \leftarrow \delta^V$. Therefore, we apply the logical AND ($\wedge$) between the active bits/nibbles/bytes (depending on the cipher) of all corresponding states and round keys and obtain the accumulated differential $\varDelta_j^P$ by

$$\varDelta_j^P = (0 \rightarrow \delta^V) \wedge (\delta^P \leftarrow V_r).$$

   Similarly, we compute the accumulated differential $\nabla_i^S$

$$\nabla_i^S = (\delta V_r \leftarrow 0) \wedge (\delta_r^V \rightarrow \delta S).$$

4. As the final step, the number of active bits/nibbles/bytes in keys, states, and intermediate states is counted in both $\varDelta_j^P$ and $\nabla_i^S$ to have a single number which refers to the recomputational effort.

## 4 Applications

We used our implementation to validate existing biclique attacks on the AES and ARIA from [5,8], and to mount new attacks on the three versions of the cipher BKSQ. Table 1 summarizes our results and compares them with previous attacks.

### 4.1 Verifications

**AES.** In our experiments on the AES we could construct bicliques on up to three rounds for the 128-bit, and on up to four rounds for the 192-bit and 256-bit versions. Hence, our results confirm to the findings of Bogdanov *et al.* in terms of maximal biclique lengths. In their independent-biclique attacks, Bogdanov *et al.* pointed out that the round key differences are a linear function of the indices $i$ and $j$. Thus, the authors could neglect the effort for recomputing the S-boxes in the key schedule. We did not employ this optimization, since we searched for a more general approach in our implementation. Additionally, we detected a minor flaw in the complexity calculation for the independent-biclique attack on the 192-bit version. There, the authors forgot to consider either the round 6 or 7 with 16 active S-boxes which increases the number of SubByte operations from 2.8125 to 3.8125, and the total complexity from $2^{189.74}$ to $2^{190.16}$.

**Table 1.** Independent-biclique attacks constructed by automated search in comparison with previously published attacks. CP: chosen plaintexts, (∗): the computational complexity should be $2^{190.16}$ (cf. Sect. 4.1).

| Primitive | Rounds | Comp. complexity | Data complexity (CP) | Memory complexity | Ref. |
|---|---|---|---|---|---|
| *AES* | | | | | |
| AES-128 | 10 (full) | $2^{126.72}$ | $2^{72}$ | $2^{8}$ | This work |
| AES-128 | 10 (full) | $2^{126.18}$ | $2^{88}$ | $2^{8}$ | [5] |
| AES-192 | 12 (full) | $2^{190.28}$ | $2^{48}$ | $2^{8}$ | This work |
| AES-192 | 12 (full) | $2^{189.74}(*)$ | $2^{80}$ | $2^{8}$ | [5] |
| AES-256 | 14 (full) | $2^{254.53}$ | $2^{64}$ | $2^{8}$ | This work |
| AES-256 | 14 (full) | $2^{254.42}$ | $2^{40}$ | $2^{8}$ | [5] |
| *ARIA* | | | | | |
| ARIA-256 | 16 (full) | $2^{255.20}$ | $2^{80}$ | $2^{8}$ | [8] |
| *BKSQ* | | | | | |
| BKSQ-96 | 10 (full) | $2^{94.47}$ | $2^{80}$ | $2^{8}$ | This work |
| BKSQ-144 | 14 (full) | $2^{142.63}$ | $2^{96}$ | $2^{8}$ | This work |
| BKSQ-192 | 18 (full) | $2^{190.78}$ | $2^{96}$ | $2^{8}$ | This work |

**ARIA.** ARIA is a Korean variant of the AES. Its round transformation provides a significant diffusion, where every input byte is involved in the computation of seven output bytes. In the key schedule of ARIA, the input key is transformed in a four-round Feistel structure to create four intermediate key words $W_0, W_1, W_2, W_3$. All round keys are then extracted from these words using rotations and XORs. Chen and Xu [8] injected one-byte differences for the $\Delta_i$- and $\nabla_j$-differentials in the leftmost 128 bits of the key, and used the rightmost 128 bits to cancel the resulting seven-byte difference. We have implemented and verified the attack on ARIA-256. However, the Feistel preparation in the key schedule refused more efficient attacks.

### 4.2 Independent-Biclique Attack on the Full AES-128 and AES-192

While the time complexities of the previous works on the AES are better than our results for them, we could decrease the data complexity for the 128-bit and 192-bit versions. In the biclique for the 128-bit version, the ciphertexts $C_i$ differ in only 11 out of 16 bytes, as can be seen on the left side of Fig. 2 in Appendix A. The bytes $0, 8, 12$ (from left: the first, third and fourth byte in the uppermost row) are active in the ciphertexts only after the key injection in the final round. Due to the key schedule of the AES, these bytes in the final round key always have an equal difference. As a consequence, since the ciphertexts can only take $(2^8)^9$ values, the data complexity is upper bounded by $2^{72}$.

Similarly, in the biclique for the 192-bit version, the ciphertexts $C_i$ differ in only five out of 16 bytes before the final key addition, as illustrated on the right side of Fig. 2 in Appendix A.

Due to the key schedule, the bytes $1, 5, 9$ (from left: the first, third and fourth byte in the second row) in the round key for the final round always have an equal difference. The ciphertexts for this biclique can take only $(2^8)^6$ values. Thus, the data complexity of an attack using this biclique is upper bounded by $2^{48}$.

### 4.3 Specification of BKSQ

BKSQ is a substitution-permutation network that was proposed by Daemen and Rijmen in [10]. The cipher represents a generalization of Rijndael, in which the state has a rectangular $m \times n$-structure (cf. [11]). There are three different versions of BKSQ which all have a state size of 96 and individual key lengths of 96, 144, or 192 bits. The internal state is represented by a $3 \times 4$- and the secret key is represented as a $3 \times 4$-, $6 \times 4$-, or $9 \times 4$-byte matrix. The plaintext is transformed in 10/14/18 rounds using the four operations:

- *MixColumns*/$\theta$: The internal state is multiplied column-wise by a circulant MDS-matrix in the Galois-Field $GF(2^8)$.
- *SubBytes*/$\gamma$: Each byte in the internal state is replaced using an $8 \times 8$-bit S-box.
- *ShiftRows*/$\pi$: The $i$-th row of the internal state for $i \in \{0, 1, 2\}$ is rotated by $i$ bytes to the left.
- *AddRoundKey*/$\sigma[k_i]$: The internal state is XORed byte-wise with the subkey $k_i$ for round $i$.

Before the first round, an inverse $\theta$-operation is applied to the plaintext and an additional key $k_0$ is XORed with the state.

### 4.4 Independent-Biclique Attack on Full BKSQ-96

This subsection explains our independent-biclique attack on full BKSQ-96. The attack includes three steps: partitioning the key space, constructing a biclique, and matching over the remaining parts of the cipher. The complexity of the attack is described at the end.

**Key Space Partitioning.** We partition the key space in $2^{80}$ sets with respect to the round key for Round 8, $k_8$. The base keys $K[0,0]$ of the sets are the $2^{80}$ 12-byte values with two bytes fixed to zero, where the ten remaining bytes run over all possible values. The $2^{16}$ keys $K[i,j]$ in a set are defined by applying the key differences $\Delta_i^K$ and $\nabla_j^K$ to the base key, where $i, j \in \{0, \ldots, 255\}$.

$$K[0,0] = \begin{array}{|c|c|c|c|}\hline \phantom{0} & \phantom{0} & \phantom{0} & \phantom{0} \\ \hline & & & \\ \hline & & 0 & 0 \\ \hline\end{array} \qquad \Delta_i^K(k_8) = \begin{array}{|c|c|c|c|}\hline \phantom{0} & \phantom{0} & \phantom{0} & \phantom{0} \\ \hline & & & \\ \hline & & i & \\ \hline\end{array} \qquad \nabla_j^K(k_8) = \begin{array}{|c|c|c|c|}\hline \phantom{0} & \phantom{0} & \phantom{0} & \phantom{0} \\ \hline \vert & \vert & & \\ \hline & & & j \\ \hline\end{array}$$

Note that the key schedule of BKSQ-96 performs a bijective mapping where every value for the secret key is mapped uniquely to one value of each round key. Thus, our splitting of the key space covers the full secret key space.

**3-Round Biclique of Dimension 8.** We construct a biclique of dimension eight over the rounds 8-10. Fig. 3 in Appendix B shows the base computation as well as the $\Delta_i$- and $\nabla_j$-differentials. It can be seen from there that all $\Delta_i$- and $\nabla_j$-differentials are independent, i.e., their keys and states do not share active bytes which are used as inputs to the non-linear S-box. From Fig. 3 in Appendix B one can see that the $\Delta_i$-differentials affect the ciphertexts $C_i$ in only 10 bytes. By fixing $C_0$ for all bicliques, we can upper bound the data complexity of this attack by $2^{80}$ ciphertexts.

**Matching Over 7 Rounds.** The matching part covers the first seven rounds of the cipher, as illustrated in Fig. 4 in Appendix B. We choose the first byte of the state after Round 3 for the partial matching. The bytes which have to be recomputed are darkened in Fig. 4.

Similar to the attacks on the AES in [5], we have to be accurate concerning the recomption effort. In all attacks on BKSQ we follow the argumentation of [5] and focus on the number of S-boxes which require recomputation in order to have a single value which refers best to the total effort, since the number of S-box lookups is the dominant summand compared to the number of recomputed $\theta$- and $\sigma$- operations.

As we can see from Fig. 4 in Appendix B, we need to consider nine S-boxes in the first, three S-boxes in the second, and one additional S-box in the third round. Hence, we have $9 + 3 + 1 = 13$ S-boxes in the forward part of the round transformation. In backward direction (covering rounds 4 to 7) we need to consider $3 + 9 + 7 + 3 = 22$ S-boxes in the round transformation. Additionally, we have to take into account the S-boxes that require recomption in the key schedule. BKSQ uses the S-box for the rightmost column of each of its round keys. There are $3 + 3 + 3 + 3 + 1 + 1 + 0 + 1 = 15$ such active S-boxes in the last column of the round keys. These sum up to $13 + 22 + 15 = 50$ S-boxes for one group of keys.

**Complexity of the Attack.** In the full BKSQ-96, there are $10 \cdot 12 = 120$ S-boxes in all $\gamma$-operations of the full cipher and 30 S-boxes in the key schedule. Thus, for $2^{16}$ keys in one key group, $C_{recomp}$ is equivalent to $2^{16} \cdot \frac{50}{150} = 2^{14.42}$ full encryptions. In all of our attacks on BKSQ we use bicliques of dimension eight. Therefore, the decryption oracle needs $2^8$ decryptions per biclique. Since we match in eight bits in the state $v$, we can expect to have $2^{16-8}$ false positive key candidates per key group in average, which have to be tested in a brute-force stage.

For BKSQ-96, the effort to construct a biclique, $C_{biclique}$, is given by computing $2 \cdot 2^8$ times three out of 10 rounds, which is equal to $2^{7.26}$ full encryptions. The precomputations costs are given by computing $2^8$ times three rounds in forward direction from $P$ to $V$ and $2^8$ times four rounds in backward direction from $S$ to $V$. Hence, $C_{precomp}$ is equal to $2^{7.49}$ encryptions. The full computational complexity is given by

$$2^{80} \cdot (2^{7.26} + 2^8 + 2^{7.49} + 2^{14.42} + 2^8) = 2^{94.48}$$

encryptions. This attack requires $2^{80}$ chosen plaintexts, and memory to store $2^{8}$ 96-bit states at a time.

### 4.5 Independent-Biclique Attack on Full BKSQ-144

**Key Space Partitioning.** In the attack on the 144-bit version of BKSQ we partition the key space in $2^{128}$ sets with respect to the block $(k_{12}\|k_{13}^{L})$, which contains the full round key $k_{12}$ and the leftmost two columns of $k_{13}$. The base keys of the sets, $K[0,0]$, are the $2^{128}$ 18-byte values, where two bytes are fixed to zero and the remaining 16 bytes run over all possible values. The $2^{16}$ keys $K[i,j]$ in a set are defined by applying the key differences $\Delta_i^K$ and $\nabla_j^K$ to the base key, where $i, j \in \{0, \ldots, 255\}$.

$$K[0,0] = \begin{array}{|c|c|c|c|c|c|}\hline & & & & 0 & \\ \hline & & & & & \\ \hline & 0 & & & & \\ \hline\end{array} \qquad \Delta_i^K(k_{12}||k_{13}^L) = \begin{array}{|c|c|c|c|c|c|}\hline & & & & i & i \\ \hline & & & & & \\ \hline & & & & & \\ \hline\end{array} \qquad \nabla_j^K(k_{12}||k_{13}^L) = \begin{array}{|c|c|c|c|c|c|}\hline & & & & & \\ \hline & & & & & \\ \hline & j & j & & & \\ \hline\end{array}$$

Note that the key schedule of BKSQ-144 maps every value of the secret key uniquely to one value of each 18-byte block of the key register. Thus, our splitting of the key space with respect to $(k_{12}\|k_{13}^{L})$ covers the full secret-key space.

**4-Round Biclique of Dimension 8.** We construct a four-round biclique which covers the rounds 11 to 14, as shown in Fig. 5 in Appendix C. This time, the ciphertexts $C_i$ are affected in all bytes. Thus, the attack can potentially include the full codebook.

**Matching Over 9 Rounds.** We match in the first byte of the state after Round 3. Figure 6 in Appendix C shows the active bytes in the matching phase. We consider 9 + 3 + 1 = 13 S-boxes in forward and 3 + 9 + 12 + 12 + 12 + 6 + 2 = 56 active S-boxes in the backward part of the matching. Moreover, in the key schedule, we have to recompute one active S-box in each of the round keys $k_1$, $k_4$, $k_7$, and $k_{10}$. Hence, there are in total 13 + 56 + 4 = 73 active S-boxes in the matching phase.

**Complexity of the Attack.** In the full cipher, there are $14 \cdot 12 = 168$ S-boxes in the $\gamma$-operations and 27 S-boxes in the key schedule. Thus, for $2^{16}$ keys in one key group, $C_{recomp}$ is equivalent to $2^{16} \cdot \frac{73}{195} = 2^{14.58}$ full encryptions. $C_{biclique}$ is given by computing $2 \cdot 2^{8}$ times four out of 14 rounds, which is equivalent to $2^{7.19}$ full encryptions. Considering $C_{precomp}$, one has to compute $2^{8}$ times ten out of 14 rounds, which is equivalent to $2^{7.51}$ full computations. The total time complexity is given by

$$2^{128} \cdot (2^{7.19} + 2^{8} + 2^{7.51} + 2^{14.58} + 2^{8}) = 2^{142.63}$$

full encryptions. The data complexity of this attack is $2^{96}$, and we need memory to store $2^{8}$ states.

### 4.6 New Independent-Biclique Attack on Full BKSQ-192

**Key Space Partitioning.** For this attack we divide the key space into $2^{176}$ sets with respect to the block $(k_{16} \| k_{17})$, which contains the keys for rounds 16 and 17. The base keys $K[0,0]$ are the $2^{176}$ 24-byte values with two bytes fixed to zero, where all other bytes run over all possible values. The $2^{16}$ keys $K[i,j]$ in a set are defined by applying the key differences $\Delta_i^K$ and $\nabla_j^K$ to the base key, where $i, j \in \{0, \ldots, 255\}$.

$$K[0,0] = \begin{array}{|c|c|c|c|c|c|c|c|}\hline \; & \; & \; & \; & \; & \; & \; & \; \\ \hline & & & & & & 0 & \\ \hline & & & & & & 0 & \\ \hline \end{array} \quad \Delta_i^K(k_{16}||k_{17}) = \begin{array}{|c|c|c|c|c|c|c|c|}\hline \; & \; & \; & \; & \; & \; & \; & \; \\ \hline & & & & & & & \\ \hline & & & & & & i & i \\ \hline \end{array} \quad \nabla_j^K(k_{16}||k_{17}) = \begin{array}{|c|c|c|c|c|c|c|c|}\hline \; & \; & \; & \; & \; & \; & \; & \; \\ \hline & & & & & & j & \\ \hline & & & & & & & \\ \hline \end{array}$$

Note, that the key schedule of BKSQ-192 maps every value of the secret key uniquely to one value of each 24-byte block of the key register. Thus, our splitting of the key space with respect to $(k_{16} \| k_{17})$ covers the full secret-key space.

**5-Round Biclique of Dimension 8.** We construct a 5-round biclique which covers the rounds 14 to 18, as shown in Fig. 7 in Appendix D. For this attack, the $\Delta_i$-differentials affect all bytes in the ciphertexts $C_i$. Hence, this attack may require the full codebook.

**Matching Over 13 Rounds.** We match in the first byte of the state after Round 5, as shown in Fig. 8, Appendix D. There, an adversary should recompute $12 + 12 + 9 + 3 + 1 = 37$ S-boxes in the forward direction, $3 + 9 + 4 \cdot 12 + 6 + 2 = 68$ S-boxes in backward direction and six S-boxes in the key schedule. Hence, $37 + 68 + 6 = 111$ S-boxes need to be recomputed in total.

**Complexity of the Attack.** In BKSQ-192, there are $18 \cdot 12 = 216$ S-boxes in the $\gamma$-operations and 51 bytes in the key schedule. Thus, for $2^{16}$ keys in one key group, $C_{recomp}$ results in $2^{16} \cdot \frac{111}{267} = 2^{14.73}$ full encryptions. $C_{biclique}$ is given by computing $2 \cdot 2^8$ times five out of 18 rounds, which is equivalent to $2^{7.15}$ full encryptions. $C_{precomp}$ is given by computing $2^8$ times 13 out of 18 rounds or $2^{7.53}$ computations. The full time complexity is given by

$$2^{176} \cdot (2^{7.15} + 2^8 + 2^{7.53} + 2^{14.73} + 2^8) = 2^{190.78}$$

full encryptions. Again, the data complexity is $2^{96}$ and the memory complexity is $2^8$.

## 5 Conclusion and Outlook

With `Janus`, we have introduced a user-friendly, highly flexible, and expandable framework for cryptanalysts which supports automated biclique cryptanalysis of a user-specified cryptographic algorithm. With this framework, we found the first full-round attacks on BKSQ-96, BKSQ-144, and BKSQ-192. It is planned to increase the number of supported primitives, e.g., the AES and SHA-3 finalists to analyze the resistance against biclique attacks.

# A Bicliques from the Attack on Full AES-128 and AES-192

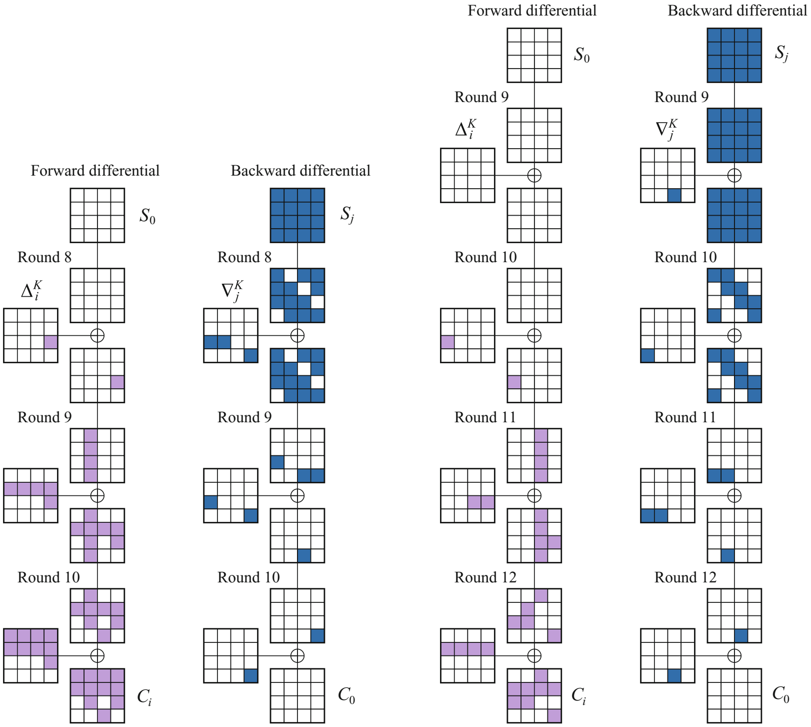

**Fig. 2.** $\Delta_i$- and $\nabla_j$-differentials of the bicliques for the AES-128 (left) over the rounds 8 - 10 and the AES-192 (right) over the rounds 9 - 12.

# B Independent-Biclique Attack on Full BKSQ-96

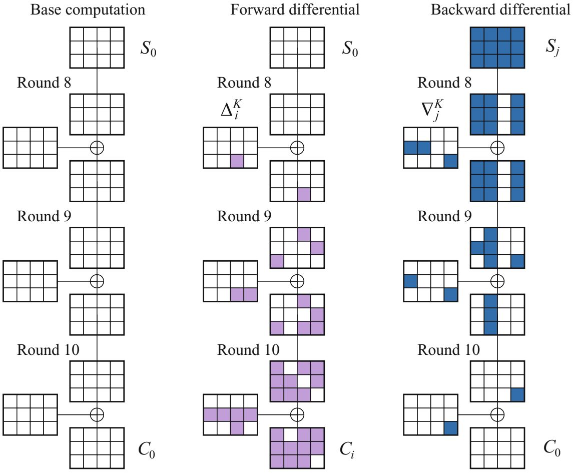

**Fig. 3.** Biclique for BKSQ-96 over the rounds 8 - 10 with $\Delta_i$- and $\nabla_j$-differentials.

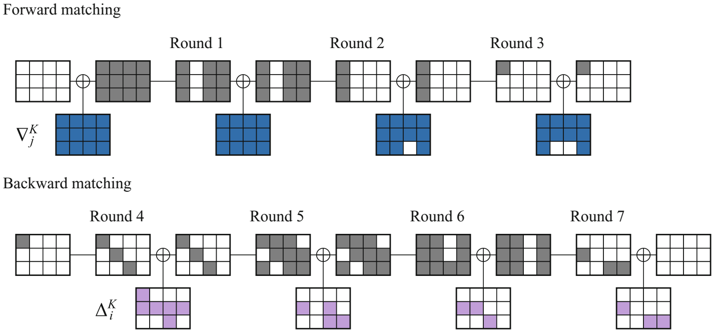

**Fig. 4.** Recomputations for BKSQ-96 in forward and backward direction.

# C Independent-Biclique Attack on Full BKSQ-144

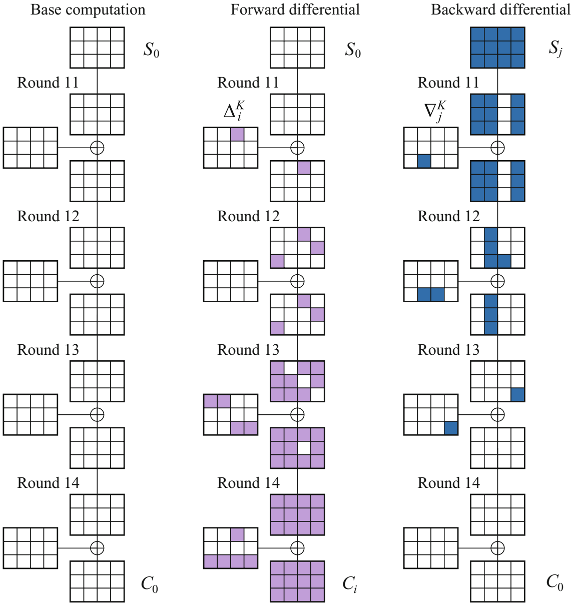

**Fig. 5.** Biclique for BKSQ-144 over the rounds 11 - 14 with $\Delta_i$- and $\nabla_j$-differentials.

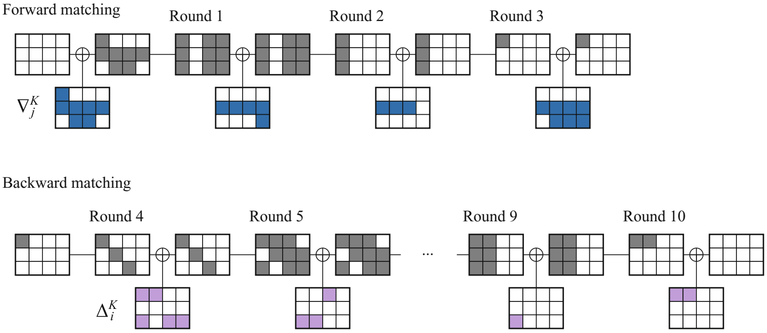

**Fig. 6.** Recomputations for BKSQ-144 in forward and backward direction.

# D Independent-Biclique Attack on Full BKSQ-192

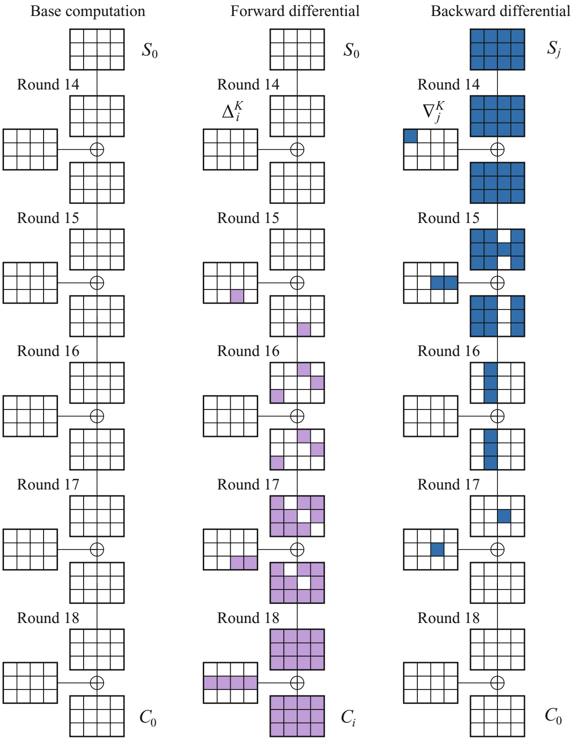

**Fig. 7.** Biclique for BKSQ-192 over the rounds 14 - 18 with $\Delta_i$- and $\nabla_j$-differentials.

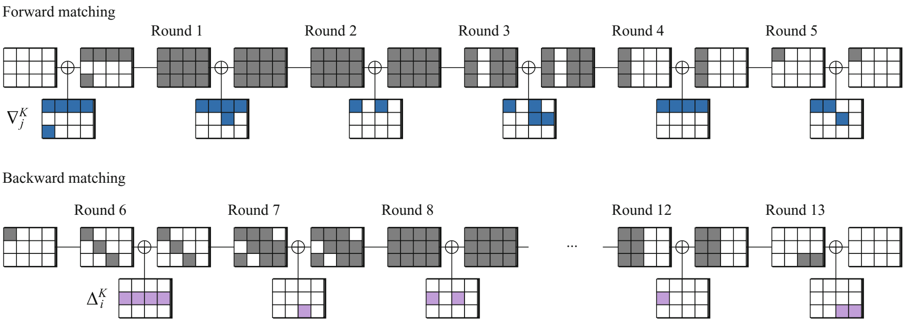

**Fig. 8.** Recomputations for BKSQ-192 in forward and backward direction.

## References

1. Abed, F., Forler, C., List, E., Lucks, S., Wenzel, J.: Biclique Cryptanalysis of the PRESENT and LED lightweight ciphers. Cryptology ePrint Archive, report 2012/591 (2012). http://eprint.iacr.org/
2. Abed, F., List, E., Lucks, S.: On the security of the core of prince against biclique and differential cryptanalysis. Cryptology ePrint Archive, report 2012/712 (2012). http://eprint.iacr.org/
3. Ahmadian, Z., Salmasizadeh, M., Aref, M.R.: Biclique Cryptanalysis of the Full-Round KLEIN block cipher. Cryptology ePrint Archive, report 2013/097 (2013). http://eprint.iacr.org/
4. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009)
5. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
6. 1T3XT BVBA. iText, a Free Java-PDF Library (2012). http://www.itextpdf.com/
7. Çoban, M., Karakoç, F., Boztaş, Ö.: Biclique cryptanalysis of TWINE. Cryptology ePrint Archive, report 2012/422 (2012). http://eprint.iacr.org/
8. Chen, S., Tianmin, X.: Biclique attack of the full ARIA-256. IACR Cryptology ePrint Archive, 2012:11 (2012)
9. Daemen, J., Van Assche, G.: Differential propagation analysis of keccak. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 422–441. Springer, Heidelberg (2012)
10. Daemen, J., Rijmen, V.: The block cipher BKSQ. In: Schneier, B., Quisquater, J.-J. (eds.) CARDIS 1998. LNCS, vol. 1820, pp. 236–245. Springer, Heidelberg (2000)
11. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Heidelberg (2002)
12. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The skein hash function family. Submission to NIST (Round 3) (2010)
13. Hong, D., Koo, B., Kwon, D.: Biclique attack on the full HIGHT. In: Kim, H. (ed.) ICISC 2011. LNCS, vol. 7259, pp. 365–374. Springer, Heidelberg (2012)
14. Jeong, K., Kang, H., Lee, C., Sung, J., Hong, S.: Biclique cryptanalysis of lightweight block ciphers PRESENT, piccolo and LED. Cryptology ePrint Archive, report 2012/621 (2012). http://eprint.iacr.org/
15. Khovratovich, D., Leurent, G., Rechberger, C.: Narrow-Bicliques: cryptanalysis of full IDEA. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 392–410. Springer, Heidelberg (2012)
16. Khovratovich, D., Rechberger, C.: A splice-and-cut cryptanalysis of the AES. IACR Cryptology ePrint Archive, 2011:274 (2011). http://eprint.iacr.org/2011/274
17. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: attacks on skein-512 and the SHA-2 family. Cryptology ePrint Archive, report 2011/286 (2011). http://eprint.iacr.org/
18. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: attacks on skein-512 and the SHA-2 family. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 244–263. Springer, Heidelberg (2012)
19. Leurent, G.: ARXtools: a toolkit for ARX analysis. University of Luxembourg, Technical report (2012)

20. Mala, H.: Biclique cryptanalysis of the block cipher SQUARE. Cryptology ePrint Archive, report 2011/500 (2011). http://eprint.iacr.org/
21. NIST National Institute of Standards and Technology. FIPS 180–2: Secure Hash Standard. April 1995. http://csrc.nist.gov
22. Sasaki, Y., Aoki, K.: Preimage attacks on step-reduced MD5. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 282–296. Springer, Heidelberg (2008)
23. Sasaki, Y., Wang, L., Aoki, K.: Preimage attacks on 41-step SHA-256 and 46-step SHA-512. Cryptology ePrint Archive, report 2009/479 (2009). http://eprint.iacr.org/
24. Shakiba, M., Dakhilalian, M., Mala, H.: Non-isomorphic biclique cryptanalysis and its application to full-round mCrypton. Cryptology ePrint Archive, report 2013/141 (2013). http://eprint.iacr.org/
25. Stankovski, P.: Automated algebraic cryptanalysis. Department of Electrical and Information Technology. Technical report, Department of Electrical and Information Technology, Lund University (2010)
26. Wang, Y., Wu, W., Yu, X.: Biclique cryptanalysis of reduced-round piccolo block cipher. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 337–352. Springer, Heidelberg (2012)
27. Wang, Y., Wu, W., Yu, X., Zhang, L.: Security on LBlock against biclique cryptanalysis. In: Lee, D.H., Yung, M. (eds.) WISA 2012. LNCS, vol. 7690, pp. 1–14. Springer, Heidelberg (2012)
28. Wei, L., Rechberger, C., Guo, J., Wu, H., Wang, H., Ling, S.: Improved meet-in-the-middle cryptanalysis of KTANTAN (poster). In: Parampalli, U., Hawkes, P. (eds.) ACISP 2011. LNCS, vol. 6812, pp. 433–438. Springer, Heidelberg (2011)