Generator sets for the selection of key differences in the Biclique Attack

G. C. de Carvalho[1], **L. A. B. Kowada**[1]

[1]Instituto de Computação - Universidade Federal Fluminense - Niterói - RJ

Abstract. *Biclique cryptanalysis is the only known method faster than brute force for some block ciphers, such as the AES. The main phase of the attack is the Preparation phase in which the attacker selects the two families of key differences and chooses ways to partition the key space and the cipher. In the versions of biclique cryptanalysis used in literature, the selection of both families of key differences is done over the same set of key bits. We introduce our novel technique that involves a new concept of generator set of key differences where each key difference can be chosen from distinct sets of key bits. This also influences the way that the key space is partitioned, allowing for new possible bicliques with better time or data complexities. To demonstrate the power of this new technique, we present in this work an attack on the Serpent-256 that uses a negligible amount of data and remains close in terms of time complexity when compared to the best biclique attack applied to the cipher.*

1. Introduction

Biclique Cryptanalysis was created in 2011 by Bogdanov et al. [Bogdanov et al. 2011] The publishing of this method was really impactful given that it was applied to the AES at the time, one of the most used ciphers even nowadays. There had never been an attack faster than brute force for the full round AES until then. In the following years after its initial success, the method was applied to a variety of ciphers such as HIGHT [Hong et al. 2011], TWINE [Çoban et al. 2012], SQUARE [Mala 2014] and ARIA [Chen and Xu 2014]. However, besides Rijndael (the original name of the AES), Biclique Cryptanalysis was applied to no other AES finalist, until the year 2020 when de Carvalho and Kowada [de Carvalho and Kowada 2020] attacked Serpent-256.

The biclique attack is divided in four phases: Preparation, Biclique Building, Obtaining plaintexts and Meet-in-the-Middle phases. The Preparation phase, our main focus, is composed of three steps: the selection of the key differences, the key partitioning and the cipher decomposition. The selection of the key differences is the most important step, since it is on this step that the key differences Δ_i^K and ∇_j^K are chosen in a way that they are independent through some rounds of the cipher, meaning that the bits affected by their propagation through the cipher must not share non-linear transformations. The key partitioning is the transformation of the key space into groups of keys in which each group has a Base Key, which is a key in the group that, when XORed with the key differences, generates all the keys in the group. The cipher decomposition is the step in which the cipher E is arranged as a composition of three subciphers $E = f \circ g \circ h$: the biclique structure is built in either f or h and the meet-in-the-middle is done over g and the other.

In practice there are no rules for which step should be done first in the Preparation phase. One way is to choose a set of key bits (usually a round key) and try to

build independent key differences propagating from it. The part of the cipher in which the differentials are independent becomes the subcipher where the biclique will be built, whereas to the rest of the cipher will be applied the meet-in-the-middle. Another way to do so is to first choose the key differences in the best way possible and then choose the set of bits to partition the key space. We show in this work that the latter broadens the possibilities and we demonstrate it, by applying the process to the Serpent cipher.

The Serpent cipher is, along with MARS, RC6, Twofish and Rijndael, one of the AES process finalists [Nechvatal et al. 2001] and has not had, since its proposal, its full round versions attacked. It is a Substitution Permutation Network (SPN) with 32 rounds, 128 bit block size and accepts keys of sizes 128, 192 and 256 bits. It has been targeted by several cryptanalysis [Kelsey et al. 2000, Biham et al. 2001b, Biham et al. 2001a, Biham et al. 2003, Collard et al. 2007b, Collard et al. 2007a, Nguyen et al. 2011] since the year 2000, but no one was able to get results on the full cipher, until the year 2020 when de Carvalho and Kowada [de Carvalho and Kowada 2020] attacked the Serpent-256 using the Biclique Cryptanalysis.

Our Contributions. It is standard in literature that both Δ_i^K and ∇_j^K key differences are chosen from the same subkeys on the cipher. We argue this approach limits the possible bicliques that can be searched, thus making it more likely that more data is used than necessary or that the biclique is not as long as is could have been.

Here, we also define the concept of generator set of key differences and use it to show the limitations of using the same generator set for both Δ_i^K and ∇_j^K key differences. We show that using this same generator set as base key for identifying the key groups in the key partitioning phase also limits the possible bicliques that can be found for a given cipher.

Finally, we create an attack on the cipher Serpent-256 with the intent to show the potential of using multiple generator sets. This yields an attack with with time complexity of $2^{255.34}$ Serpent full computations while using only 16 pairs of texts, which compares favourably to the previously found attack on literature from de Carvalho and Kowada [de Carvalho and Kowada 2020] since it needs 2^{88} pairs of texts and is only slightly faster at $2^{255.21}$ Serpent full computations.

Paper structure. Section 2 gives an overview of the biclique cryptanalysis while Section 3 details each important step involved on the biclique attack. This includes our main theoretical contributions on Section 3.2. Section 4.1 describes the Serpent cipher and the notations used in the paper for handling it. Section 4 describes the attack we created to demonstrate the proposed technique. Finally, Section 5 concludes the article.

2. Overview of the Biclique Cryptanalysis

The original attack, as presented by Bogdanov *et al.* [Bogdanov et al. 2011], is a $d-$dimension biclique. Nowadays there are many other variations such as the Star-based Bicliques [Canteaut et al. 2013, Bogdanov et al. 2014] and LDC Biclique [Ahmadi et al. 2014], but the overview of the attack is still the same. It is divided into: Preparation, Biclique Building, Obtaining texts and Meet-in-the-Middle phases. We now look at each of these steps:

- **Preparation phase.** An adversary partitions the key space into groups of keys.

Each key group is associated with a matrix K, where each element $K[i, j]$ represents a key in the group. Let E be the attacked cipher, which is a composition of the subciphers f, g and h, $E = f \circ g \circ h$. The three steps below are then applied for each key group.

1. **Biclique Building phase.** A biclique structure is built either over the subcipher f or h. Assume it is constructed over f. This results in a structure that satisfies the following condition
$$\forall i, j : S_j \xrightarrow[f]{K[i,j]} C_i,$$
where S_j are internal states of the cipher and C_i are ciphertexts. We call internal state any block of the cipher after applied an operation. Section 3.1 explains that the biclique is a structure that satisfies this condition.
2. **Obtaining plaintexts phase.** Since this is a chosen ciphertext attack, we have at our disposal a decryption oracle, which is used to obtain the plaintext P_i for each ciphertext C_i.
$$\forall i : C_i \xrightarrow[E^{-1}]{decryption\ oracle} P_i.$$
3. **Meet-in-the-Middle phase.** For each key $K[i, j]$ in the group it is tested if
$$\exists i, j : P_i \xrightarrow[g \circ h]{K[i,j]} S_j.$$
If one of the $K[i, j]$ is the secret key, then the above condition is satisfied. Therefore, every key that satisfies it is a candidate to the secret key. Section 3.3 shows a way, created by Bogdanov *et al.* [Bogdanov et al. 2011], to do this faster than a simple meet-in-the-middle approach, called Matching with Precomputations.

3. The steps of the Biclique Attack

3.1. Bicliques

First presented by Bogdanov *et al.* [Bogdanov et al. 2011], we now look at the balanced biclique structure applied to the cipher $E = f \circ g \circ h$. Let f be the subcipher that maps an internal state S to the ciphertext C using the key K (*i.e.* $f_K(S) = C$). A *dimension* d *biclique over* f is the triple $(\{S_j\}, \{C_i\}, \{K[i, j]\})$, where $0 \leq i, j < 2^d$ and

$$\forall i, j : f_{K[i,j]}(S_j) = C_i.$$

One way to achieve this condition is using related-key differentials. It is important to highlight the fact that this is a single key attack. The related-key model is used only within the key groups.

Let $K[0, 0]$ be the *base key*, *i.e.* the key that maps the internal state S_0 to the ciphertext C_0. This is called the *base computation*

$$S_0 \xrightarrow[f]{K[0,0]} C_0.$$

The next step is defining the Δ_i*-differentials* and ∇_j*-differentials* using related-key differentials.

Δ_i-*differentials* map the input difference 0 to the output difference Δ_i, using the key difference Δ_i^K, where $\Delta_0 = \Delta_0^K = 0$ and $0 \leq i < 2^d$,

$$0 \xrightarrow[f]{\Delta_i^K} \Delta_i,\ \Delta_0 = \Delta_0^K = 0.$$

In contrast, ∇_j-*differentials* map the input difference ∇_j to the output difference 0, using the key difference ∇_j^K, where $\nabla_0 = \nabla_0^K = 0$ and $0 \leq j < 2^d$,

$$\nabla_j \xrightarrow[f]{\nabla_j^K} 0,\ \nabla_0 = \nabla_0^K = 0.$$

If both sets of differentials are independent (do not share non-linear components, such as S-boxes), then it is possible to combine them into (Δ_i, ∇_j)-*differentials*

$$\nabla_j \xrightarrow[f]{\nabla_j^K \oplus \Delta_i^K} \Delta_i.$$

Being independent means that an internal state or subkey of f is affected by ∇_j-*differentials* if and only if it is not affected by Δ_i-*differentials*.

By definition, the base computation conforms to both sets of differentials and hence, it is possible to substitute it to the combined differentials

$$S_0 \oplus \nabla_j \xrightarrow[f]{K[0,0] \oplus \nabla_j^K \oplus \Delta_i^K} \Delta_i \oplus C_0.$$

By letting

$$S_j = S_0 \oplus \nabla_j,$$
$$C_i = \Delta_i \oplus C_0 \text{ and}$$
$$K[i,j] = K[0,0] \oplus \nabla_j^K \oplus \Delta_i^K$$

we have the definition of a dimension d biclique over f.

Building a biclique this way costs only 2^{d+1} computations of f, since it is possible to choose the key differences and base computation, and then, independently, compute the Δ_i-*differentials* and ∇_j-*differentials*.

3.2. The Preparation Phase

This phase is when the theoretical steps of the attack are described. Once this phase is complete, the other phases can be easily implemented.

3.2.1. Defining the key differences

As seen in section 3.1, there are two families of key differences that must be chosen, Δ^K and ∇^K. They must be chosen in a way that the families of differentials generated from the propagation of their bits (namely Δ and ∇), through the key schedule and internal states of the cipher, are independent in part of the cipher, either in the beginning or the

end in most cases. We call this part of the cipher as subcipher f. This means that those differentials do not share non-linear components with each other inside of f, thus making it possible to combine them creating an independent biclique over f.

Each family of key differences is defined over a set of key bits (usually one or more round keys) that is sufficient to generate all key bits of the key schedule. We call that a *generator set* of the key schedule. For example, any round key of the AES-128 cipher is a set of key bits capable of generating all others.

To define the family of key differences, one must choose a generator set and set all bits of it to 0, except for the active bits i.e. the bits that we want to propagate throughout the cipher. Those bits are unique to each key difference in the family. The active bits of a key difference can then be concatenated to form a unique integer that in turn, identifies it. In the context of the biclique attack, we have two families, Δ^K and ∇^K, and each of their elements is addressed as Δ_i^K and ∇_j^K.

These families are the backbone of the independent bicliques as seen in Section 3.1, as they both enable the independence of the bicliques and define the dimension of it. The latter is true because the log_2 of the cardinality of the families defines the dimensions of the biclique. If both families have the same cardinality d then the biclique is balanced ($d-$dimensional biclique), else the biclique is unbalanced.

The standard in literature is that the chosen generator set is the same to both Δ^K and ∇^K. We argue that this is not going to necessarily yield the best results because it limits the possible bicliques that can be searched, making it more likely that more data is used than necessary or that the biclique is not as long as is could have been.

Choosing two different generator sets, one for each of Δ^K and ∇^K, can generate longer bicliques for the balanced biclique approach (the same approach used on the original attack of Bogdanov et al. [Bogdanov et al. 2011]) when applied to certain ciphers. This can be done by choosing a generator set more distanced from the edge of f (either beginning or end of the cipher depending on where the biclique is constructed) for the Δ^K family and a closer to the edge of f for the ∇^K family. If they are still independent, it makes so that the Δ_i- and ∇_j- differentials activate the most possible bits in the biclique, which does not affect the time complexity of the attack since the biclique building phase has a quadratic gain over the biclique dimension. The work of de Carvalho and Kowada [de Carvalho and Kowada 2020] already uses this concept, although not to its fullest potential, using only slightly different generator sets.

On the low data complexity method, choosing different generator sets can be used to improve the time complexity of the recomputation phase, while maintaining the data complexity really low, since there are more possibilities to find key differences capable of activating fewer bits on the rest of the cipher. First, we choose the generator set of the Δ^K near the edge of f. This is required to keep the data complexity low. The generator set of ∇^K is then chosen to be the most distanced as possible from the edge in order to activate the most bits of f, and to propagate more slowly to the rest of the cipher.

As an extreme example, we could define the generator set of the Δ^K family as being the round key $\$10$ of the AES, while the generator set of ∇^K could be $\$0$. The only condition to be satisfied is that inside of either f or h, both families are able to build an independent biclique.

3.2.2. The key partitioning

The key partitioning phase is where the key groups are defined, that is, the key space is partitioned into groups of keys that are disjoint. During the attack, each of these groups is tested separately to find secret key candidates. This is what enables a parallel approach.

For every group, there is one key that is capable of generating all others through the XOR with the families of key differences Δ^K and ∇^K, and therefore, it identifies the group. This key is called *base key*. The group is represented by the matrix K, where the base key is the element $K[0][0]$. All other keys are the elements $K[i][j] = K[0][0] \oplus \Delta_i^K \oplus \nabla_j^K$.

The matrix K of the current group being tested is the one used both in the building of the biclique and the matching with precomputations thus being of uttermost importance.

Up to this day, within what we could find in literature, all attacks implicitly assume that two assertions are true:

1. The base keys are defined as one or more round keys of the cipher;
2. The base keys and the generator sets of the key differences have to be defined over the same round keys.

The first assertion is not true because all that is needed is a set of bits capable of generating all key bits of the cipher, in turn being able to encipher a plaintext into the corresponding ciphertext. On the other hand, it is usually easier to use one or more round keys to define this set.

The second one is false, for all that is needed is that:

1. the key groups are disjoint and their union forms the whole key space and
2. each element of each group is $K[i][j] = K[0][0] \oplus \Delta_i^K \oplus \nabla_j^K$ for the base key $K[0][0]$.

That means that it is possible to, for instance, use the subkey $\$10$ of the AES to define the family of key differences Δ^K, $\$0$ to define ∇^K and $\$8$ to define the key groups, if it is capable of generating an independent biclique over f or h and satisfy the above conditions.

The above example, would create a biclique attack that needs a small amount of data (since it depends only on the bits activated by Δ^K of the ciphertext/plaintext) and that is more efficient time-wise in the recomputation step because the recomputation of the text up to the intermediate variable depends heavily on the amount of bits activated by ∇^K (More information about the recomputation step in the Section 3.3).

3.2.3. The cipher decomposition

The cipher decomposition is in which the attacked cipher E is partitioned, first into two sub ciphers $E = F \circ H$ and then into three subciphers $E = f \circ g \circ h$. This happens because first one must decide if the biclique is going to be built in the beginning or the end or the cipher. If it is the beginning, then $F = f \circ g$ and $H = h$, else $F = f$ and $H = g \circ h$.

This decision takes into account the properties of the cipher E which may facilitate the independence of the differentials in one end or the other. For example, SPN ciphers, such as AES and Serpent, use a XOR with another subkey in the last round instead of applying a linear transformation, which makes it more advantageous to build the biclique over f than over h.

Assuming that we want to build the biclique over f, it is always advantageous to try and make f as long as possible, while maintaining the independence of the biclique. This means that the more internal states of the cipher there are in f, the less states will need to be recomputed on the Matching Phase. This makes the attack faster, because the recomputation step is the most time consuming of the entire attack.

The second partitioning, the one that encompasses g, depends on the choice of the intermediate value v to be used on the Matching with Precomputaions phase, since v is either where h ends and g begins (if the biclique is in f) or where g ends and f begins.

3.3. Matching with Precomputations

This technique [Bogdanov et al. 2011] uses the knowledge that only parts of the cipher are affected by the differentials of the biclique to do the meet-in-the-middle faster.

This can be further exploited if instead of meeting an entire internal state, we meet in only a part of the state, namely v. This way we only look at the parts affected by the differentials and that affect v.

Let $E = f \circ g \circ h$, where the biclique was built over f and that the size of Δ^K is d_1 and of ∇^K is d_2. An adversary then computes and stores $2^{d_1} + 2^{d_2}$ full computations of the cipher up to the variable v: 2^{d_1} computations of h and 2^{d_2} computations of g^{-1}

$$\forall i : P_i \xrightarrow[h]{K[i,0]} v_i^1 \text{ and } \forall j : v_j^2 \xleftarrow[g^{-1}]{K[0,j]} S_j.$$

This means that every internal state of and subkeys used in g and h up to v have to be stored. This is the *precomputation phase*.

Then comes the *recomputation phase*, where the parts that differ from the stored values must be recomputed. These parts are the ones influenced by the key bits activated by ∇^K on h and the key bits activated by Δ_K on g^{-1}. This is true because the parts that are not affected by neither of them were already stored in the precomputation phase, as well as the parts affected only by Δ^K on h and the ones affected only by ∇^K on g^{-1}.

The time cost of this method is heavily dependant on the recomputation phase, since this step is done for all keys in the group, meaning that it is done $2^{d_1+d_2}$ times, while the precomputation is done $2^{d_1} + 2^{d_2}$ times. Therefore, the most important measures to diminish the time complexity of the whole attack are the ones that reduce the amount of computation done in this step. This can be done through longer bicliques and choosing key differences that activate the least possible amount of bits in g and h. The choice of v also influences this cost, as the bigger the bit size of v, the more bits of the internal states are necessary to compute it. On the other hand, the smaller the bit size of v the more false positives are generated in average, which means that more key candidates have to be tested in each group.

The memory complexity of the method is dominated by the precomputation phase being equal to the amount of bytes necessary to store all internal states and subkeys used in the computation of both g and h for all $2^{d_1} + 2^{d_2}$ tests. This usually ends up in the order of magnitude of megabytes, which is irrelevant nowadays.

3.4. Complexities

This attack can be seen as an improved exhaustive search, since every key will be tested, but not the whole cipher will be computed in each step. Three types of complexities are of interest: memory, data and time.

The memory complexity is dominated by the Precomputation Phase of the Matching with precomputations method due to requiring the storage of whole states of many rounds of the cipher.

The data complexity depends only on how many bits of C_i are affected by the Δ_i-differentials, which depends essentially on the amount of rounds covered by the cipher as well as on the dimension and diffusion properties of the cipher.

Finally, the time complexity is where most of the analysis is necessary. It is basically the number of key groups times the time complexity of each iteration. Each iteration builds the biclique and then does the matching with precomputations, which is divided into precomputation phase and recomputation phase. If there are 2^{d_1} Δ_i^K key differences and 2^{d_2} ∇_j^K key differences we have

$$C_{time} = 2^{k-d_1-d_2}(C_{biclique} + C_{precomp} + C_{recomp} + C_{falsepos}).$$

The false positives are the keys that pass on the test in the recomputation phase, meaning that they are secret key candidates. Thus it is necessary to check if they are the secret key.

4. Applying the new method to the Serpent Cipher

4.1. The Serpent Cipher

To show the potential of this attack, we apply it to the AES finalist Serpent. Serpent is a 32-round SPN Cipher. Each internal state and each subkey has 128 bits, divided into 4 words of 32 bits each, while the secret key may have 128, 192 or 256 bits. We call Serpent-128 the version of this cipher with 128-bit key and similarly for the other key sizes. The description of the cipher can be found in the seminal paper [Biham et al. 1998] and will not be addressed here due to space constraints.

4.2. Defining the key differences

First, we must define the families of key differences Δ^K and ∇^K. As stated in Section 3.2.1, we can choose any two generator sets of the key schedule, one for each family, that best suits our initial conditions.

These conditions refer to time complexity or data complexity, meaning that we can minimize either one or the other. It is also possible to minimize one while restricting the other. For our attack, the objective is to find a biclique with smallest time complexity while maintaining the data complexity in 2^4, since the Serpent Cipher works with nibbles.

To carry out the attack, we built a Java program that searches all possible Δ^K and ∇^K such that their generator sets are built from two consecutive keys in the cipher and a 4-dimensional independent bicliques is created from them.

There were 18056 possible bicliques. To choose the one with the smallest time complexity, it was necessary to define a nibble of some internal state of the cipher as the variable v for the matching with precomputations phase of the attack. Depending on the chosen v, one or other biclique could be the best.

By choosing v as the nibble 31 of state #75, the best time complexity is associated to the following families of key differences:

- Δ^K: The keys K^{31} and K^{32} with the nibble 6 of K^{31} activated, form the generator set of Δ^K.
- ∇^K: The keys K^{18} and K^{19} with the nibble 11 of K^{18} activated, form the generator set of ∇^K.

4.3. Key Partitioning

The key partitioning is the second step of the preparation phase. In this step, the main objective is to define a **base key**, which is a set of key bits capable of creating all keys groups such that:

1. the key groups are disjoint and their union forms the whole key space and
2. each element of each group is $K[i][j] = K[0][0] \oplus \Delta_i^K \oplus \nabla_j^K$ for the base key $K[0][0]$.

To conform to these conditions for the Serpent cipher, it is enough to choose two consecutive subkeys, in which all nibbles, except for two, vary through all possible values. The two nibbles that are exceptions are the ones affected only by Δ_i^K and ∇_j^K. One being affected by only Δ_i^K and the other only by ∇_j^K.

To find candidates for the base key, we tested all nibbles affected by the Δ_i^K key differentials and ∇_j^K key differentials, but not by both at the same time, to find all nibbles that go through all possible values only once. If both nibbles exist in two consecutive subkeys, than we can choose them as base key.

The program that made the tests outputs all possible nibbles that can be chosen as base key. Since any of them can be chosen, we chose subkeys K^{16} and K^{17} as the base key. Nibbles 9 and 19 of K^{16} are fixed to zero, since those are the ones that go through all possibilities only when the XOR with Δ_i^K and ∇_j^K is done. The values of all other nibbles define which key group is being tested at any given time.

4.4. The cipher decomposition

The cipher must be decomposed into $E = f \circ g \circ h$ where, in our case, the biclique is built on the subcipher f. It is the computation from state $\#91$ up to the end of the cipher, state $\#96$. It covers the last two rounds of the cipher excluding the operation AK_{30}. In other words, it is two rounds minus one application of L.

The subcipher h covers the computation from the plaintext up to $\#75$, while g is the computation from $\#75$ up to $\#91$. This is the case because the variable v of the matching phase is set on state $\#75$.

4.5. The Definition of the Biclique

First, an adversary sets $C_0 = 0$ and calculates $S_0 = f^{-1}_{K[0,0]}(C_0)$, where $K[0,0]$ is the base key of the group. Then, the Δ_i-differentials are computed using the key differences Δ_i^K on the subkeys K^{31} and K^{32}. The ∇_j-differentials are computed using the key differentials ∇_j^K on the subkeys K^{18} and K^{19}. As stated in Section 4.2, the key differentials were chosen in such a way that the Δ_i-differentials and ∇_j-differentials are independent inside of f.

Figure 1 shows the Δ_i-differentials and ∇_j-differentials generated from the key differences over f. It is easy to see that they are independent, i.e. they share no non-linear components of the cipher (Sboxes).

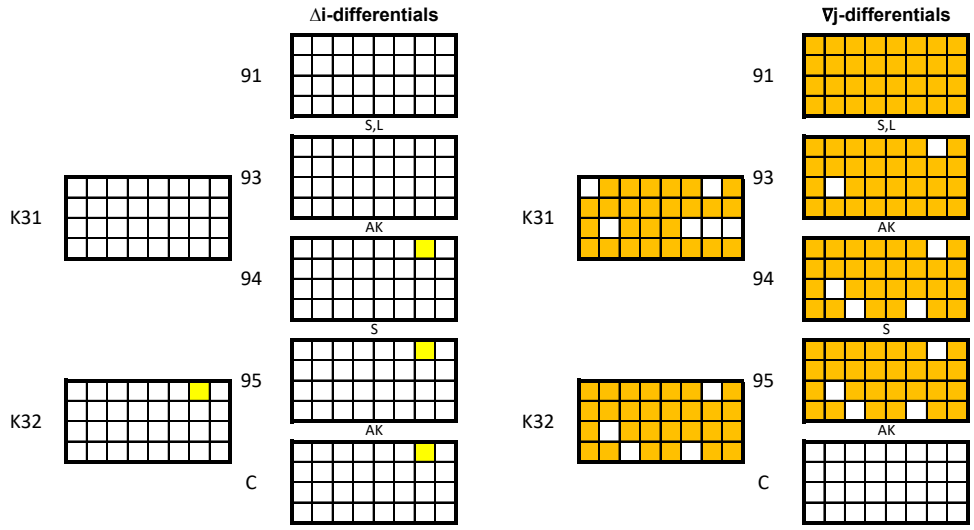

Figure 1. Δ_i-differentials and ∇_j-differentials in the $4-$dimension biclique.

The nibbles of the ciphertext C affected by the Δ_i-differentials determine the amount of possible ciphertexts, and respective plaintexts, are necessary to carry the attack. Since there is only one nibble, the data complexity of this attack is only $2^4 = 16$ chosen ciphertexts.

4.6. Matching with Precomputations

In this part of the attack, we check if the secret key belongs to the group $\{K[i,j]\}$, *i.e.* if $K_{secret} \in \{K[i,j]\}$. First we precompute 2^4 values of $v^1_{i,0}$ from P_i and precompute 2^4 values of $v^2_{0,j}$ from S_j. We define v as being the nibble $\#75_{31}$. Then, they are saved, together with all internal states and subkeys involved in these precomputations. Now we have

$$P_i \xrightarrow[h]{K[i,j]} v^1_{i,j} \text{ and } v^2_{i,j} \xleftarrow[g]{K[i,j]} S_j$$

for each i and j, recomputing only those parts that differ from the ones saved in memory. If $v^1_{i,j} = v^2_{i,j}$, then $K[i,j]$ is a key candidate.

4.6.1. Forward Recomputation

Here we observe the difference between the computation of $P_i \xrightarrow{K[i,j]} v$ and the precomputed values of $P_i \xrightarrow{K[i,0]} v$, given by the influence of ∇_j^K on all the subkeys from K^0 up to K^{24} on the variable v, to calculate how many Sbox recomputations must be done. All observations of this section can be seen on Figure 2.

All nibbles influenced by the keys K^0 to K^{22} affect the computation of v and, since we apply Sboxes to all key nibbles, they have to be counted, resulting in 213 Sbox recomputations. On the other hand, not all nibbles of subkeys K^{23} and K^{24} need to be recomputed, only those that affect v. So, although there are 20 and 26 active nibbles of K^{23} and K^{24} respectively, only 13 nibbles of K^{23} and 5 nibble of K^{24} affect v. Totalling $213 + 18 = 231$ Sbox recomputations.

In addition to the recomputations on the subkeys, there are the recomputations of the active nibbles of the cipher. Only the states $\#T$ where $T \equiv 1 \mod 3$ are relevant due to they being the states that precede the operation S. State $\#1$ is influenced only by the nibbles of subkey K^0 and therefore has 14 active nibbles. All other 22 relevant states from $\#4$ up to $\#67$ have all of their nibbles being active. On states $\#70$ and $\#73$, only a few nibbles affect the computation of v, so, only these matter. For $\#70$ there is a total of 22 active nibbles and $\#73$ has only 5. The total is $14 + 33 * 22 + 5 = 745$ Sbox recomputations.

Hence, the total cost of the forward recomputations is $231 + 745 = 976$ Sbox recomputations.

4.6.2. Backward Recomputation

Similarly to the forward recomputation, we look at the difference between $v \xleftarrow{K[i,j]} S_j$ and the precomputed $v \xleftarrow{K[0,j]} S_j$, given by the influence of Δ_i^K in the subkeys between K^{25} and K^{30}. All observations of this section can be seen on Figure 2.

We begin by looking at the active nibbles of the subkeys K^{30}, K^{29} and K^{28} since they are relevant to the computation of v. Only 1, 5 and 4 nibbles of the keys K^{30}, K^{29} and K^{28}, respectively, are active. There are 4 nibbles of key K^{27} that affect v, 2 nibbles of K^{26} and none of K^{25}. This sums up to 16 Sbox recomputations.

Now we focus on the active nibbles of the internal states. Only states of the type $\#T$ where $T \equiv 2 \mod 3$ are relevant, since they come right after the application of the operation S (we could also have used $\#T$ where $T \equiv 0 \mod 3$). State $\#88$ has 5 active nibbles, while $\#85$ has 25 and $\#82$ has 22. Finally, $\#79$ has 5 active nibbles while $\#76$ has only one. In total we have 58 Sbox recomputations.

In total, the sum of the recomputations of the subkeys and internal states we have $16 + 58 = 74$ Sbox recomputations.

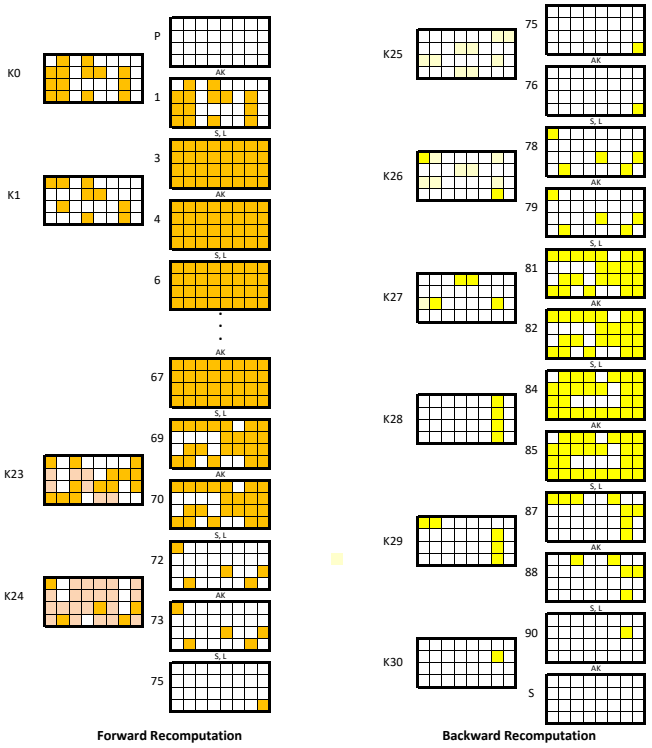

Figure 2. Forward recomputation and backward recomputation of the $4-$dimension biclique. The lighter nibbles are the ones that are affected by the key differences, but does not affect the recomputation.

4.7. Complexities

Firstly we have

$$C_{total} = 2^{k-2d}(C_{biclique} + C_{precomp} + C_{recomp} + C_{falpos}).$$

On average, there will be 2^4 false positives per iteration ($C_{falpos} = 2^{2d}/2^{|v|} = 2^8/2^4$). We also know that $C_{biclique} = (2^4 + 2^4) * (5/96) = 2^{0.736965594166206}$ and $C_{precomp} = 2^4 * (91/96) = 2^{3.922832139477540}$. It remains to find C_{recomp}. For that, we shall count each S-box computation done in the recomputation phase and compare the total number of S-box computations involved in the computation of the whole cipher. In this way, we know which percentage of the cipher is computed in each iteration, observing only the number of S-boxes, due to the fact that this is the most expensive operation of the cipher.

Once it is not necessary to observe the nibbles that do not influence the recomputation (due to the fact that the key schedule only computes the S-boxes after calculating all words of the subkeys), the total of S-box computations done by the forward recomputation is 976, while the backward recomputation only does 74, as seen in subsections 4.6.1 and 4.6.2 respectively, totalling 1050 S-box computations. Since the entire cipher computes 2080 S-boxes, then $C_{recomp} = 2^8 \cdot (1050/2080) = 2^{7.013805799525030}$.

In the end, we obtain approximately

$$C_{attack} = 2^{248} \cdot (2^{0.74} + 2^{3.92} + 2^{7.01} + 2^4) = 2^{255.34}$$

Serpent-256 computations.

In terms of memory, the attack is upper limited by 2^4 computations of h and 2^4 computations g, since h and g together are much bigger than f and thus, much more memory is necessary to store all of their internal states and subkeys than storing the 2^8 states necessary for the biclique. The full computation of h consists of 75 internal states and 24 subkeys while g has 16 internal states and 6 subkeys, with 16 bytes each. Therefore the memory complexity is $(2^4 \cdot (75+24) + 2^4 \cdot (16+6)) \cdot 16 = 2^{14.92}$ bytes approximately.

The most important aspect of this attack is the fact that only $2^4 = 16$ pairs of plaintexts/ciphertexts are necessary.

5. Conclusions

Our work focuses on expanding the boundaries of what is possible inside the biclique method. Specifically, the focal points are the selection of the families of key differences Δ^K and ∇^K and the key partitioning step of the Preparation Phase, where the base keys are selected as the representative of the key groups. We show that, through our new concept of generator set of key differences, it is possible to choose distinct generator sets for each family of key differences, as well as for the base keys. This broadens the spectre of possible bicliques that can be built.

To demonstrate it, we presented here an attack to the full round Serpent-256 using this new concept. It was capable of using far less data than the previous attacks to the cipher from de Carvalho & Kowada [de Carvalho and Kowada 2020] while being slightly slower in comparison. Our attack uses only 16 pairs of plaintext/ciphertext and has a time complexity of $2^{255.34}$. This is favourably compared to both attacks in [de Carvalho and Kowada 2020]. Their slowest attack has a data complexity of 2^{60} and time complexity of $2^{255.45}$, while the fastest attack has 2^{60} and $2^{255.21}$ respectively. This means that our attack is only slightly slower than the fastest one while requiring a really small amount of data, compared to the huge amount necessary for both the attacks.

Future work involves identifying in which types of ciphers this technique is most useful and which it is not. For instance, our current method has the data complexity lower bounded by the smallest word used in the cipher on the non-linear step. In the case of the Serpent, the nibble is the smallest word size on the last round. It may also be possible to create faster attacks using the whole space of data, by choosing appropriate generator sets for the key differences. Although this creates unfeasible attacks, it may be useful to compare this technique to the classic ones applied to other ciphers, such as the AES, PRESENT, HIGHT and many others.

References

Ahmadi, S., Ahmadian, Z., Mohajeri, J., and Aref, M. R. (2014). Low Data Complexity Biclique Cryptanalysis of Block Ciphers with Application to Piccolo and HIGHT. *IEEE Trans. Information Forensics and Security*, 9(10):1641–1652.

Biham, E., Anderson, R., and Knudsen, L. (1998). Serpent: A new block cipher proposal. In *International Workshop on Fast Software Encryption*, pages 222–238. Springer.

Biham, E., Dunkelman, O., and Keller, N. (2001a). Linear cryptanalysis of reduced round serpent. In *International Workshop on Fast Software Encryption*, pages 16–27. Springer.

Biham, E., Dunkelman, O., and Keller, N. (2001b). The rectangle attack—rectangling the serpent. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 340–357. Springer.

Biham, E., Dunkelman, O., and Keller, N. (2003). Differential-linear cryptanalysis of serpent. In *International Workshop on Fast Software Encryption*, pages 9–21. Springer.

Bogdanov, A., Chang, D., Ghosh, M., and Sanadhya, S. K. (2014). Bicliques with minimal data and time complexity for aes. In *International Conference on Information Security and Cryptology*, pages 160–174. Springer.

Bogdanov, A., Khovratovich, D., and Rechberger, C. (2011). Biclique cryptanalysis of the full AES. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 344–371. Springer.

Canteaut, A., Naya-Plasencia, M., and Vayssiere, B. (2013). Sieve-in-the-middle: Improved MITM attacks (Full Version). Cryptology ePrint Archive, Report 2013/324. `https://eprint.iacr.org/2013/324`.

Chen, S.-z. and Xu, T.-m. (2014). Biclique key recovery for ARIA-256. *IET Information Security*, 8(5):259–264.

Çoban, M., Karakoç, F., and Boztaş, Ö. (2012). Biclique cryptanalysis of TWINE. In *International Conference on Cryptology and Network Security*, pages 43–55. Springer.

Collard, B., Standaert, F.-X., and Quisquater, J.-J. (2007a). Improved and multiple linear cryptanalysis of reduced round serpent. In *International Conference on Information Security and Cryptology*, pages 51–65. Springer.

Collard, B., Standaert, F.-X., and Quisquater, J.-J. (2007b). Improving the time complexity of matsui's linear cryptanalysis. In *International Conference on Information Security and Cryptology*, pages 77–88. Springer.

de Carvalho, G. C. and Kowada, L. A. (2020). The first biclique cryptanalysis of Serpent-256. In *SBSEG*. SBC. `http://sbseg.sbc.org.br/2020/pdfs/criptografia_mencao_honrosa.pdf`.

Hong, D., Koo, B., and Kwon, D. (2011). Biclique attack on the full HIGHT. In *International Conference on Information Security and Cryptology*, pages 365–374. Springer.

Kelsey, J., Kohno, T., and Schneier, B. (2000). Amplified boomerang attacks against reduced-round mars and serpent. In *International Workshop on Fast Software Encryption*, pages 75–93. Springer.

Mala, H. (2014). Biclique-based cryptanalysis of the block cipher SQUARE. *IET Information Security*, 8(3):207–212.

Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., and Roback, E. (2001). Report on the development of the Advanced Encryption Standard (AES). *Journal of Research of the National Institute of Standards and Technology*, 106(3):511.

Nguyen, P. H., Wu, H., and Wang, H. (2011). Improving the algorithm 2 in multidimensional linear cryptanalysis. In *Australasian Conference on Information Security and Privacy*, pages 61–74. Springer.