Revisiting the Biclique Attack on the AES

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

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

Abstract. *The AES Cipher is one of the most widely used block ciphers throughout the world for the better part of two decades now. Despite its relevancy, there has been no great progress in the attempts at finding exploitable flaws or cryptanalysis techniques that are able to find the secret key in less time than simple exhaustive search for its full version. The only exception is biclique cryptanalysis which was used more than once to recover the secret key in marginally less time than simple brute force. The last improvement happened 8 years ago. This paper finds the best results for all but one of the variations attempted on the AES, through the help of the concept of generator sets for related-key differentials, in terms of time complexity as well as a software that semi-automates tests on general word-based ciphers.*

1. Introduction

The AES Cipher is one of the most widely used block ciphers throughout the world for the better part of two decades now. Despite its relevancy, there has been no great progress in the attempts at finding exploitable flaws or cryptanalysis techniques that are able to find the secret key in less time than simple exhaustive search for its full version.

The only exception is biclique cryptanalysis which was used in 2011 to create the first attack faster than brute force on the full version of AES [Bogdanov et al. 2011]. At the time, this new method inspired its straightforward application in many ciphers [Çoban et al. 2012, Abed et al. 2012, de Carvalho and Kowada 2020], as well as the development of variations to apply in others which were not as simple of a task [Khovratovich et al. 2012, Chen and Xu 2014].

Many relevant improvements were introduced over the years. One that deserves mentioning is star-based bicliques, which deals with the biggest problem of biclique cryptanalysis: the amount of data necessary to perform the attack being too high in comparison to the small gain in time for searching the keys. This variation is able to reduce to only one plaintext-ciphertext pair, turning the method into a direct improvement over just brute force [Canteaut et al. 2013, Bogdanov et al. 2014]. However, it does become slower in comparison with other variations.

The most recent improvement was made in 2022 and is regarding the way in which the related-key differentials of the attack are chosen [de Carvalho et al. 2022]. The concept of generator sets enables the cryptanalyst to use more than one subkey as basis for the attack which in turn greatly increases the possible bicliques that can be used.

The process of finding bicliques, most of the time, is not simple. This is due to most of the processes involved being ad hoc, with few to no frameworks to help the cryptanalyst, as is true for most fields in cryptanalysis. Last decade there has been a movement towards the development of frameworks and software for this field in particular, from progress in automated proofs of security for sbox-based ciphers [Mouha et al. 2011] and

ARX ciphers [Liu et al. 2016, Sun et al. 2017], and most recently an automatic tool has been used to find related-key differentials for some instances of the block cipher Rijndael [Rouquette et al. 2022].

Nonetheless, the only existing tool for biclique cryptanalysis is a software that was created in 2011 [Abed et al. 2014] but was never updated with the new variations, therefore being archaic nowadays. Furthermore, the software was not able to find the best bicliques on account of it being only able to look for simple bicliques, such as the original one [Bogdanov et al. 2011].

1.1. Our contributions

Our work consists in turning the task of finding bicliques less demanding through a new design perspective and through our software that semi-automates it. The tool is capable of finding related-key differentials however way the user desires and can extrapolate a biclique from them.

To show them in practice, we improve on most of the time complexity results of biclique cryptanalysis over the AES using a 128 bit key (AES-128). This is done by combining the generator sets method from [de Carvalho et al. 2022] with our improved overview of the attack. Table 1 compiles the comparison between the results over AES found in literature and ours.

Table 1. Summary of the comparison between our results and the best found in literature for biclique cryptanalysis over the AES

identifier	time complexity	data complexity	our time complexity	our data complexity	authors
star-based	$2^{126.71}$	1	$2^{126.69}$	1	[Bogdanov et al. 2014]
uses whole codebook	$2^{125.64}$	2^{128}	$2^{125.64}$	2^{128}	[Bogdanov et al. 2014]
unbalanced	$2^{126.13}$	2^{72}	$2^{126.02}$	2^{88}	[Tao and Wu 2015]
dimension 16 balanced	$2^{126.02}$	2^{88}	$2^{125.90}$	2^{88}	[Tao and Wu 2015]

2. Background of independent bicliques

Here we informally define the core concepts needed to understanding biclique cryptanalysis as a whole. For all definition below, assume that: It is given a cipher C_{ipher} which has a secret key with k bits and m total key bits.

2.1. Generator sets for the key bits

A generator set is any set of key bits that is enough to generate all m key bits of C_{ipher} through some algorithm.

Prior to its creation in 2022 [de Carvalho et al. 2022], instead of generator sets, a base key was used for all the related-key differentials involved in the biclique attack, and they were always a single or consecutive subkeys of C_{ipher}, restricting the amount of bicliques that could be found. With generator sets, this is not the case. Each differential in the attack can be defined over a different generator set, if it is necessary.

2.2. Related-key differentials

A related-key differential Δ is defined by a related-key difference Δ^K, which can be defined, for the purposes of the biclique attack, as the pair (G, δ), where G is a generator set and δ is an assigned value for each bit of G. For instance, for the AES-128 cipher, if the secret key is chosen as G the following bitstring in hexadecimal represents the assignment of each bit of G (the rightmost being bit 0):

$$\delta = 0x00000000000000000000\text{FF}000000\text{FF}00$$

The related-key differential is then the propagation to all other key bits of the cipher as well as the propagation through the internal states of that cipher. Every state bit and key bit that is affected by this propagation is *active* and everyone that is not affected is not.

The propagation is done through two encryptions of a single plaintext P (or decryptions of a single ciphertext C): one by an arbitrary assignment of values a for G and the second one by $a \oplus \delta$. Every state or key bit whose value was different through the encrytion is active.

In our case, and many others, it is more useful to work with active words instead of bits, simply due to most operations in the cipher's encryption being word-based (all of them for AES).

2.3. Independent bicliques

A biclique covers a pre-defined number of consecutive states, thus, the initial state and the final state of the biclique must be pre-defined. The biclique is also formed by at least two families of related-key differentials, one whose propagation (inside the biclique) is done by encryption (Δ_i-differentials defined by Δ_i^K key differences) and other by decryption (∇_j-differentials defined by ∇_j^K key differences).

For computing the active words of the Δ_i-differentials, the propagation starts in the initial state and propagates to both ends of the cipher (encryption and decryption). The ∇_j-differentials, starts in the final state and propagates to both ends of the cipher (encryption and decryption). The biclique is independent if and only if the Δ_i-differentials and ∇_j-differentials share no non-linear components of the cipher (Sboxes).

$$P \xleftarrow{\Delta} B_{initial} \xrightarrow{\Delta} C \quad P \xleftarrow{\nabla} B_{final} \xrightarrow{\nabla} C$$

Considering the existence of the initial and final states, we can partition C_{ipher} into three subciphers, f, g and h, where h covers from the plaintext to the initial state, g covers the biclique and h covers from the final state till the final state, as shown below.

$$P \underset{h}{\rightarrow} B_{initial} \underset{g}{\rightarrow} B_{final} \underset{f}{\rightarrow} C.$$

The subcipher g can also cover the first state or the last state of the cipher. In that case, respectively, h and f are the identity cipher.

2.4. Key partitioning

The biclique attack can be seen as an optimization of an exhaustive search. This is due to the fact that every single key is tested, except that it is done in a way that is faster than just simply testing each possible key through the cipher.

For that to happen, it is necessary to partition the whole key space into groups in a way that each group is tested separately through the biclique. The improvement in speed comes from this choice.

Each group has a representative, called base key. The base key of the group has some bits fixed to 0 while all other vary from group to group. The bits that are fixed to 0 are the ones that go through every possible value we the key difference of each related-key differential in the biclique is applied to them.

For example, suppose a base key in which bytes 0 and 1 are fixed to 0. This means that there has to be a total of 2^{16} related-key differentials in the cipher, and every one of them influence one or both of those bytes in a way that no key is repeated nor is not tested.

Furthermore, the base key must also be a generator set (usually a subkey or consecutive subkeys), so that all the key bits can be generated to carry out the attack.

3. The Biclique Attack

In this section is presented the steps necessary to execute the biclique attack on a given cipher, as well as how the time, memory and data complexities are computed.

3.1. The steps of the attack

After choosing the related-key differentials and which internal states the biclique covers, the actual attack follows these steps below. We assume that the biclique contains only one of each related-key differential for simplicity, in which case, all keys inside a group are indexed in a $2^{d_1} \times 2^{d_2}$ matrix $K[i, j]$, where $0 \leq i < 2^{d_1}$, $0 \leq j < 2^{d_2}$ and d_1 and d_2 are the dimensions of the first and second related-key differentials, respectively.

1. **Building the biclique.** The chosen biclique results in a structure that satisfies the following condition
$$\forall i, j : S_j \xrightarrow[g]{K[i,j]} T_i,$$
where S_j are internal states of the cipher that are at the beginning of the biclique and T_i are internal states at the end of the biclique.
2. **Obtain data.** Since this is either a chosen plaintext or chosen ciphertext attack, we have at our disposal either an encryption or a decryption oracle, which is used to obtain the ciphertext C_i for each plaintext P_i or vice-versa.
$$\forall i : P_i \xrightarrow[C_{ipher}]{encryption\ oracle} C_i \text{ or } \forall i : C_i \xrightarrow[C^{-1}_{ipher}]{decryption\ oracle} P_i.$$
3. **Meet-in-the-Middle.** For each key $K[i, j]$ in the group it is tested either if
$$\exists i, j : P_i \xrightarrow[h]{K[i,j]} S_j \text{ or } \exists i, j : T_i \xrightarrow[f]{K[i,j]} C_j.$$

depending on which end of the cipher has been chosen for the propagation.
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. To do this faster than a simple meet-in-the-middle approach, there is a method called matching with precomputations.

3.2. Matching with precomputations

This method consists on the choosing of a variable v, that can be any amount of words in a internal state of the cipher between an edge and the biclique, such that the computing of v is done from both sides, depending on the biclique.

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

This is useful due to the fact that we are able to precompute and save in memory all internal states and subkeys of both related-key differentials for the keys $K[i,0]$ and $K[0,j]$ in the forward (encryption) and backward (decryption) directions.

What is left is to recompute those words of the cipher that are affected by both the related-key differentials. The other words do not need to be recomputed because need they can be read from memory.

3.3. 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. This is negligible for most almost all variations of biclique cryptanalysis.

The data complexity depends only on how many bits of C_i are affected by the Δ_i-differentials (or P_j are affected by ∇_j-differentials depending on the biclique), which depends essentially on the amount of rounds covered by the cipher, its positioning inside the cipher, as well as on the 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

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

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 and $C_{precomp}$ includes the biclique building step.

4. The AES-128 cipher

The AES cipher is one of the most well known ciphers in the world. Due to space constraints, we refer to the original paper for a description [Daemen and Rijmen 2013].

We keep the notations used in other relevant papers. In essence, for the first nine rounds, an even state is a pre-AddRoundKey state while the odd ones are pre-SubBytes states. The pre-ShiftRows and pre-MixColumns states are not enumerated or shown in the diagrams. Each one of the eleven subkeys are enumerated with a $ symbol, the secret key being subkey $0.

Finally, inside a state or subkey, the bytes are enumerated from 0 to 15 start at the leftmost column and uppermost row, descending each column before passing to the next one.

5. Our attacks

All of our attacks were implemented and tested by a developed software created by us. It is a Java console application capable of finding bicliques for ciphers given the parameters. The only two ciphers implemented are the AES-128 and Serpent, however any word-based cipher can be implemented by simply creating a new class, extending the "Cipher" class and implementing its abstract methods.

Another aspect of every attack in this paper is that no biclique has the same generator set for all its related-key differentials. This adds a small substep at the beginning of the biclique building step in which the base key must first generate all the subkeys of the cipher and only then the key differences are xored to compute the related-key differentials. This does not affect the overall complexity since it is done only once for each group and costs less than a full AES-128 computation. Besides that, the definition of the base key is also a little different, given that a single base key must be chosen to be used as the representative of each key group to be tested.

The complete framework as well as the material regarding the attacks that are not present in this work can be found at the github repository, accessible through the following address `https://github.com/MfMhj3uNy5gfp4Z/BicliqueFinder`.

5.1. Star

The star is a different kind of biclique, in which there are two or more related-key differentials of the same kind and is always constructed in one of the ends of the cipher. If it is constructed in the beginning, there are only Δs, and there are only ∇s if its in the end of the cipher.

This is the case because, when constructed this way, no words are active in the plaintext or ciphertext, making it so that only one pair of data is necessary to carry the attack.

Our attack uses an 8-dimension balanced biclique. This means that each family of related-key differentials, contains 2^8 different ones. For the AES, this means that a single value $0 \leq i_0 < 2^8$ is used to define the Δ_{i_0}-differentials and $0 \leq i_1 < 2^8$ is used to define the Δ_{i_1}-differentials. The generator set for Δ_{i_0} is subkey $1, where both bytes 8 and 12 are equal to i_0, and the one for Δ_{i_1} is subkey $0, where both bytes 0 and 4 are equal to i_1.

There are many available base keys to be chosen for this particular biclique. We chose subkey \$0, in which all bytes assume all values, except for bytes 4 and 8, which are fixed to 0. In the biclique building step they will assume all values through i_0 and i_1, respectively.

The recomputation phase is, for the time complexity of the attack, the most relevant part. We are interested in the amount of Sboxes that are active both in forward and backward directions towards the variable v. In the case of our attack, v is byte 3 of state $\#11$. Due to space constraints, both the propagation through the biclique and the recomputaion step are not shown here. We refer to `https://github.com/MfMhj3uNy5gfp4Z/BicliqueFinder`.

The data complexity of the attack is trivially just 1 pair of plaintext-ciphertext, due to being no active bytes on state $\#0$. The time complexity is given by the fraction of the total amount of Sboxes that must be recomputed in the attack divided by the total amount of Sboxes in the cipher. The relevant states are the pre-SubBytes ones, i.e., the odd numbered ones.

Therefore, in the forward direction, there are 16 in states $\#5$ and $\#7$ and 4 in state $\#9$, totaling 36 Sboxes to be recomputed. For the backward direction, state $\#19$ requires the recomputation of 6 bytes, while states $\#17$ and $\#15$ must have all 16 bytes recomputated. State $\#13$ needs only 4 and $\#11$ needs 1. This makes it 43 Sboxes. Since there are no Sboxes to be recomputed for the keys, than the total is 79 out of 200 Sboxes. They need to be recomputed for all possible values of Δs, thus $C_{recomp} = 2^{16} \times 79/200 = 2^{14.66}$.

Finally, $C_{precomp} \approx 2^{8.5}$ full computations of the AES cipher, because the precomputation only needs to be done till v, and $C_{falpos} = 2^8$ since there are 2^{16} tests and only 2^8 possible values for v. The complete time complexity is $2^{112} \times 2^{14.69} = 2^{126.69}$, a small improvement (one less Sbox) than the best in literature [Bogdanov et al. 2014].

5.2. No restrictions on the data

The peculiarity of this kind of biclique is that it can be constructed in the middle of the cipher instead of one of the edges. This creates more possible bicliques, but almost guarantees that all bytes of the last (or first) state are active, due to the diffusion properties of the AES-128.

We are not able at this time to find a better biclique than what is in the literature in terms of time complexity. What we managed to find were some bicliques that are as good as the literature and some that are worse, although they need to recompute less Sboxes in the internal states of the cipher. The problem is that for them it is necessary to also recompute some bytes of the subkeys. Furthermore, we decide to show one of the bicliques that manages the same time complexity as the best in literature [Bogdanov et al. 2014].

Similarly to our star-based attack, this one uses an 8-dimension balanced biclique, but this time, there is a Δ family and a ∇ family. Therefore, a single value $0 \leq i < 2^8$ is used to define the Δ_i-differentials and $0 \leq j < 2^8$ is used to define the ∇_j-differentials. The generator set for Δ_i is subkey \$2, where both bytes 8 and 12 are equal to i, and the one for ∇_j is subkey \$4, where only byte 0 is equal to j.

There are many available base keys to be chosen for this biclique as well. We

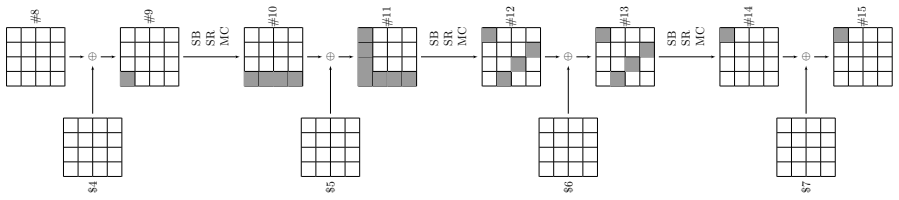
chose subkey \$3, in which all bytes assume all values, except for bytes 0 and 3, which are fixed to 0. In the biclique building step they will assume all values through i and j, respectively.

In the recomputation phase we are interested in the amount of Sboxes that are active both in forward and backward directions towards the variable v. In the case of our attack, v is byte 3 of state $\#15$. Due to space constraints, both the propagation through the biclique and the recomputaion step are not shown here. We refer to `https://github.com/MfMhj3uNy5gfp4Z/BicliqueFinder`.

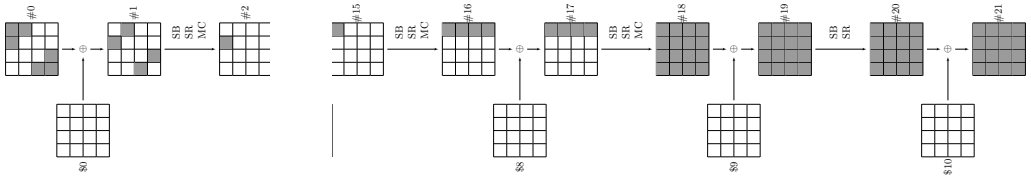
The data complexity of the attack is the whole codebook, due to all bytes in state $\#0$ being active, i.e. 2^{128} pairs of plaintext/ciphertext.

For the time complexity, in the forward direction, there is 1 byte to be recomputed in state $\#9$, 7 bytes in state $\#11$ and 4 more for state $\#13$. The total is 12 Sboxes. For the backward direction, state $\#19$ requires the recomputation of all 16 bytes, while state $\#17$ needs only 4 and $\#15$ must recompute only 1. However, since the biclique is not on any edge of the cipher, some states have to be recomputed until the edge. In this case, the backward needs to take into account the 4 active bytes of state $\#1$. This totals 25 Sboxes for the backward part. Since there are no Sboxes to be recomputed for the keys, than the total is 37 out of 200 Sboxes. They need to be recomputed for all possible values of Δ_i and ∇_j, thus $C_{recomp} = 2^{16} \times 37/200 = 2^{13.57}$.

Finally, exactly as is for the star-based attack, $C_{precomp} \approx 2^{8.5}$ full computations of the AES cipher, because the precomputation only needs to be done till v, and $C_{falpos} = 2^8$ since there are 2^{16} tests and only 2^8 possible values for v. The complete time complexity is $2^{112} \times 2^{13.64} = 2^{125.64}$. This is the same time complexity as shown previously in the literature [Bogdanov et al. 2014], with the difference being the use of multiple generator sets for defining the families of related-key differentials.



(a) Bytes that need to be recomputed in forward direction



(b) Bytes that need to be recomputed in backward direction

Figure 1. The figure shows the forward and backward recomputations for the attack with no data restrictions.

5.3. Unbalanced biclique

An unbalanced biclique is one that does not have the same dimension for Δ and ∇ or that has more Δ families than ∇ or vice-versa. Using more families diminishes time complexity because two Δ families (same for ∇) do not need to be independent from each other for the biclique to be independent. The independence is only between the Δ and ∇ families. On the other hand, this method increases the amount of memory necessary to store the precomputations of each family. Although, for this size, it is still negligible. For this kind of biclique, we are able to noticeably improve upon the previous best attack in literature [Tao and Wu 2015], in terms of time complexity.

Differently from our previous attacks, this one uses an (8,16)-dimension unbalanced biclique, which means that there is a Δ family and two ∇ families. Therefore, a single value $0 \leq i < 2^8$ is used to define the Δ_i-differentials, $0 \leq j_0 < 2^8$ is used to define the ∇_{j_0}-differentials and $0 \leq j_i < 2^8$ is used to define the ∇_{j_1}-differentials.

The generator set for Δ_i is subkey \$7, where byte 8 is equal to i, the one for ∇_{j_0} is subkey \$9, where both bytes 1 and 5 are equal to j_0 and the generator set for ∇_{j_1} is subkey \$9, where bytes 0 and 4 are both equal to j_1. Figure 2 shows their propagation through the biclique and is possible to see that the Δ_i-differentials are independent from both ∇-differentials.

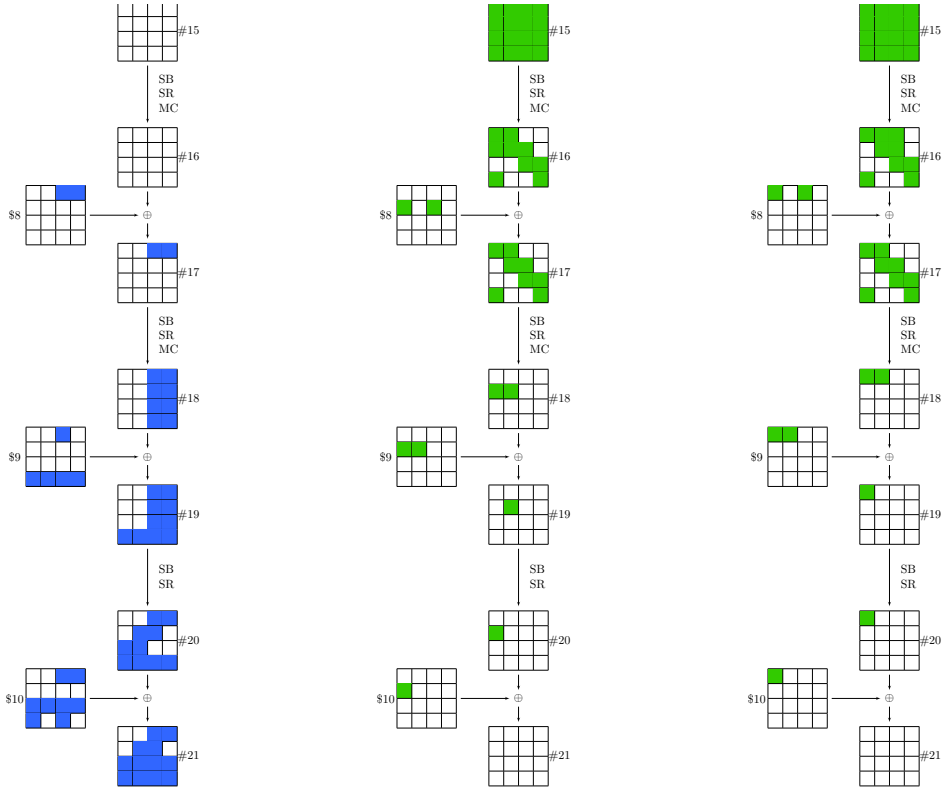
We know choose a base key between many available ones. We chose subkey \$8, in which bytes 0, 1 and 12 are fixed to 0 and the others go through all possibles values. In the biclique building step, byte 0 will assume all values through j_1, byte 1 will assume all values through j_0 and byte 12 assumes all values through i.

The recomputation phase is, for the time complexity of the attack, the most relevant part. We are interested in the amount of Sboxes that are active both in forward and backward directions towards the variable v. For this attack, v is byte 0 of state $\#5$. Figure 3 shows the bytes that are relevant for the recomputation in both directions.

The data complexity of the attack is given by the number of active bytes on state $\#21$. Since there are 12 bytes, the data complexity should be 2^{96}, but bytes 10 and 11 are always the same, turning the data complexity into 2^{88}.

For the time complexity, in the forward direction, there are only 5 bytes that need to be recomputed in state $\#1$ and state $\#3$ needs to recompute only 4, totaling 9 Sboxes to be recomputed. For the backward direction, state $\#13$ requires the recomputation of 4 bytes, while states $\#11$ and $\#9$ must have all 16 bytes recomputated. State $\#7$ recomputes only 4 and $\#5$ needs 1. This makes it 41 Sboxes. Since there are no Sboxes to be recomputed for the keys, than the total is 50 out of 200 Sboxes. They need to be recomputed for all possible values of Δs, thus $C_{recomp} = 2^{24} \times 50/200 = 2^{22.0}$.

Finally, $C_{precomp} \approx 2^{16}$ full computations of the AES cipher, because the precomputation only needs to be done till v an it is dominated by the precomputation of the ∇ families. On the other hand, $C_{falpos} = 2^{16}$ since there are 2^{24} tests and only 2^8 possible values for v. The complete time complexity is $2^{104} \times 2^{22.02} = 2^{126.02}$, a small improvement (one less Sbox) than the best in literature [Bogdanov et al. 2014].



(a) Δ_i-differentials (b) ∇_{j_0}-differentials (c) ∇_{j_1}-differentials

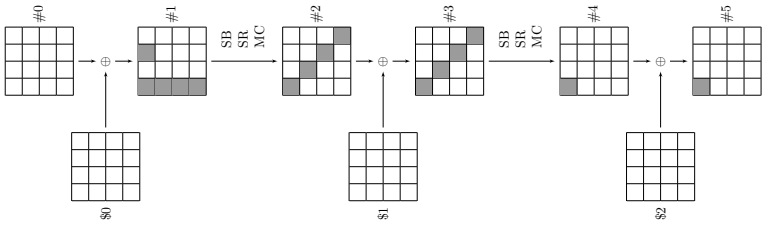
Figure 2. The figure shows both Δ_i, ∇_{j_0} and ∇_{j_1} inside the biclique for the attack with the (16,8)-dimension unbalanced biclique.

5.4. 16-dimension balanced biclique

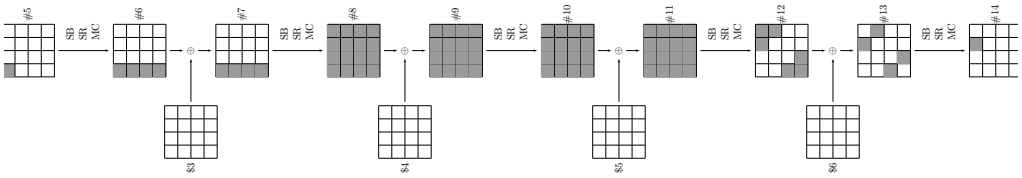
This biclique is a mix of the unbalanced one [Tao and Wu 2015] and the original balanced biclique [Bogdanov et al. 2011]. It is still positioned in the end of the cipher and both Δ and ∇ families have the same dimension as is in the first ever biclique attack. However, it uses the idea from the unbalanced biclique of having more than one Δ and ∇ families instead of just one for each, activating more bits. As is the case for the previous attack, using more families diminishes time complexity because two Δ families (same for ∇) do not need to be independent from each other for the biclique to be independent. The independence is only between the Δ and ∇ families. The amount of memory necessary to store the precomputations of each family does start to become concerning past this point, but it continues to be negligible for this biclique (less than 1 GB of memory). Similarly to the previous attack, we are able to noticeably improve upon the previous best attack in literature [Tao and Wu 2015], in terms of time complexity.

This one uses a 16-dimension balanced biclique, which means that there are two Δ families and two ∇ families. Therefore, $0 \leq i_0 < 2^8$ is used to define the Δ_{i_0}-differentials, $0 \leq i_1 < 2^8$ is used to define the Δ_{i_1}-differentials, $0 \leq j_0 < 2^8$ is used to define the ∇_{j_0}-differentials and $0 \leq j_i < 2^8$ is used to define the ∇_{j_1}-differentials.

The generator set for Δ_{i_0} is subkey \$8, where bytes 8 and 12 are equal to i_0, the generator set for Δ_{i_1} is subkey \$9 in which byte 7 is equal to i_1. The one for both ∇



(a) Bytes that need to be recomputed in forward direction



(b) Bytes that need to be recomputed in backward direction

Figure 3. The figure shows the forward and backward recomputations for the attack with the (16,8)-dimension unbalanced biclique..

is subkey \$10, where byte 0 is equal to j_0 for ∇_{j_0} and byte 4 is equal to j_1 for ∇_{j_1}. Figure 4 shows their propagation through the biclique and is possible to see that both Δ-differentials are independent from both ∇-differentials.

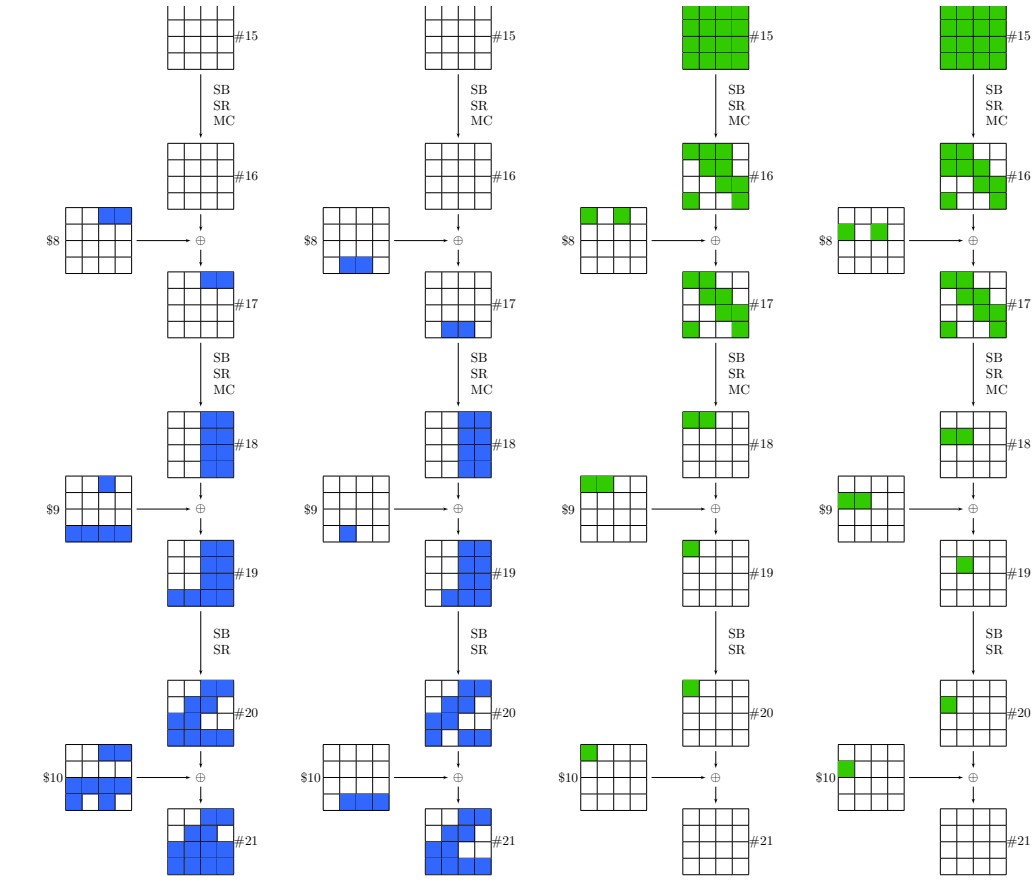
We chose subkey \$8 as base key, in which bytes 0, 1, 11 and 12 are fixed to 0 and the others go through all possibles values. In the biclique building step, byte 0 will assume all values through j_0, byte 1 will assume all values through j_1, byte 11 will assume all values through i_1 and byte 12 assumes all values through i_0.

We are interested in the amount of Sboxes that are active both in forward and backward directions towards the variable v, for the recomputation phase. In the same way as the last attack, v is byte 0 of state #5. Figure 5 shows the bytes that are relevant for the recomputation in both directions.

The data complexity of the attack is given by the number of active bytes on state #21. Exactly as happened in the last attack, there are 12 bytes, which makes the data complexity be 2^{96}. However, bytes 10 and 11 are always the same, turning the data complexity into 2^{88}. Then, there is the time complexity.

In the forward direction, there are only 5 bytes that need to be recomputed in state #1 and state #3 needs to recompute only 4, totaling 9 Sboxes to be recomputed. In the backward direction, state #13 requires no recomputation at all, while states #11 and #9 must have all 16 bytes recomputated. State #7 recomputes only 4 and #5 needs 1. This makes it 37 Sboxes. Since there are no Sboxes to be recomputed for the keys, than the total is 46 out of 200 Sboxes. They need to be recomputed for all possible values of Δ, thus $C_{recomp} = 2^{32} \times 46/200 = 2^{29.88}$.

Finally, $C_{precomp} \approx 2^{16.5}$ full computations of the AES cipher, because the pre-computation only needs to be done till v. On the other hand, $C_{falpos} = 2^{24}$ since



(a) Δ_{i_0}-differentials **(b)** Δ_{i_1}-differentials **(c)** ∇_{j_0}-differentials **(d)** ∇_{j_1}-differentials

Figure 4. The figure shows both Δ_{i_0} and Δ_{i_1}, as well as ∇_{j_0} and ∇_{j_1} inside the biclique for the attack with the 16-dimension balanced biclique.

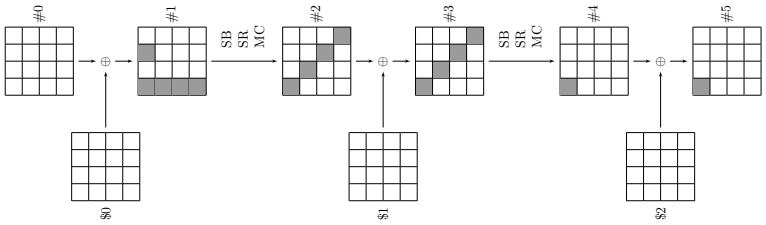
there are 2^{32} tests and only 2^8 possible values for v. The complete time complexity is $2^{96} \times 2^{29.90} = 2^{125.90}$. This is a big improvement from the previous result ($2^{126.02}$), being the first attack, that does not use the full codebook, to go below 2^{126}.

6. Conclusions

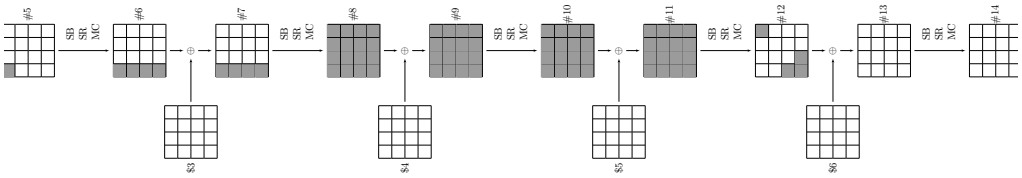
This paper presents the first progress on an attack over any full version of the AES since 2015, by revisiting a technique that has been rarely used for the better part of a decade now. The lack of use is due to some well established ideas regarding the biclique attack. Mainly, the fact that, although basically all block ciphers based on Sboxes can be affected by the biclique attack, this technique only marginally improves the time complexity when compared to an exhaustive search on account of it being an optimization of brute force instead of a shortcut attack.

The improvement we presented is attained through the combination of the concept of generator sets for bicliques with a semi-automated software that facilitates the work of the cryptanalyst when applying the biclique attack to a cipher.

Our results show that there was still room for enhancement in the biclique attack.



(a) Bytes that need to be recomputed in forward direction



(b) Bytes that need to be recomputed in backward direction

Figure 5. The figure shows the forward and backward recomputations for the attack with the 16-dimension balanced biclique.

Moreover, although there are only slight gains from our technique for finding bicliques on the AES-128, other ciphers may be more susceptible to it. The attack is trivially expanded for the other versions of the AES, being necessary only the implementation of these versions on the software we made available here: `https://github.com/MfMhj3uNy5gfp4Z/BicliqueFinder`.

Therefore, future work includes the application of this variant on other ciphers and, as a consequence, finding which properties a cipher must have to be more or less vulnerable to biclique cryptanalysis, under this variant.

Acknowledgments

This research was supported by fellowships from FAPERJ (project APQ1 n.211.666/2021) for Luis Kowada.

References

Abed, F., Forler, C., List, E., Lucks, S., and Wenzel, J. (2012). Biclique cryptanalysis of the PRESENT and LED lightweight ciphers. *IACR Cryptology ePrint Archive*, 2012:591.

Abed, F., Forler, C., List, E., Lucks, S., and Wenzel, J. (2014). A framework for automated independent-biclique cryptanalysis. In *Fast Software Encryption: 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers 20*, pages 561–581. 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.

Daemen, J. and Rijmen, V. (2013). *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media.

de Carvalho, G. et al. (2022). Generator sets for the selection of key differences in the biclique attack. In *Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 1–14. SBC.

de Carvalho, G. C. and Kowada, L. A. (2020). The first biclique cryptanalysis of serpent-256. In *Anais do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 29–42. SBC.

Khovratovich, D., Leurent, G., and Rechberger, C. (2012). Narrow-Bicliques: cryptanalysis of full IDEA. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 392–410. Springer.

Liu, Y., Wang, Q., and Rijmen, V. (2016). Automatic search of linear trails in arx with applications to speck and chaskey. In *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*, pages 485–499. Springer.

Mouha, N., Wang, Q., Gu, D., and Preneel, B. (2011). Differential and linear cryptanalysis using mixed-integer linear programming. In *International Conference on Information Security and Cryptology*, pages 57–76. Springer.

Rouquette, L., Gerault, D., Minier, M., and Solnon, C. (2022). And rijndael? automatic related-key differential analysis of rijndael. In *Progress in Cryptology-AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18–20, 2022, Proceedings*, pages 150–175. Springer.

Sun, L., Wang, W., and Wang, M. (2017). Automatic search of bit-based division property for arx ciphers and word-based division property. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 128–157. Springer.

Tao, B. and Wu, H. (2015). Improving the biclique cryptanalysis of AES. In *Australasian Conference on Information Security and Privacy*, pages 39–56. Springer.