Alternative Key Schedules for the AES

Christina Boura[1], Patrick Derbez[2], and Margot Funk[1]

[1] Université Paris-Saclay, UVSQ, CNRS, Laboratoire de mathématiques de Versailles, 78000, Versailles, France
{christina.boura,margot.funk}@uvsq.fr
[2] Univ Rennes, Inria, CNRS, IRISA, France
patrick.derbez@irisa.fr

Abstract. The AES block cipher is today the most important and analyzed symmetric algorithm. While all versions of the AES are known to be secure in the single-key setting, this is not the case in the related-key scenario. In this article we try to answer the question whether the AES would resist better differential-like related-key attacks if the key schedule was different. For this, we search for alternative permutation-based key schedules by extending the work of Khoo et al. at ToSC 2017 and Derbez et al. at SAC 2018. We first show that the model of Derbez et al. was flawed. Then, we develop different approaches together with MILP-based tools to find good permutations that could be used as the key schedule for AES-128, AES-192 and AES-256. Our methods permitted to find permutations that outperform the permutation exhibited by Khoo et al. for AES-128. Moreover, our new approach based on two MILP models that call one another allowed us to handle a larger search space and thus to search for alternative key schedules for the two bigger versions of AES. This method permitted us to find permutations for AES-192 and AES-256 that provide better resistance to related-key differential attacks. Most importantly, we showed that these variants can resist full-round boomerang attacks.

Keywords: AES · key schedule · MILP · related-key attacks · differential cryptanalysis

1 Introduction

The Rijndael family of block ciphers was designed by Joan Daemen and Vincent Rijmen in the late 90's. In 2000, the National Institute of Standards and Technology (NIST) selected three members of this family of ciphers to replace the DES and to form what is known today as the Advanced Encryption Standard (AES) [10]. In the standardized version, the block size is equal to 128 bits and

All authors were partially supported by the French Agence Nationale de la Recherche through the OREO project under Contract ANR-22-CE39-0015.

the key size can be 128, 192 or 256 bits. The AES is considered today as the most important and widely deployed symmetric primitive and its elegant design inspired several others through the years.

After almost 25 years of intense analysis and scrutiny, all three versions, i.e. AES-128, AES-192 and AES-256, are still considered secure in the single-key scenario. However, in the related-key setting, the two bigger variants of the AES were shown to be much weaker. In 2009, Biryuvov et al. discovered full-round related-key boomerang attacks, with respectively $2^{99.5}$ time and data complexity for AES-256 and 2^{123} data and 2^{176} time complexity for AES-192 [2, 3]. More attacks on AES-192 or AES-256, breaking all rounds or a high number of them, were described later, notably boomerang attacks [12, 8], differential meet-in-the-middle attacks [5] or attacks exploiting other properties [11].

The design of the AES round function, including its S-box as well as the MixColumns operation, was done with concrete criteria in mind and was based on solid mathematical arguments borrowed from the theory of Boolean functions and error-correcting codes. On the other hand, the design of the key schedule was much more ad-hoc with much less precise and formal security arguments employed. Mainly, the authors wanted the key schedule to be "different enough" from the round function. It is today considered that this component is responsible for the weaknesses discovered on the biggest versions in the related-key setting.

Even if the AES was not designed with related-key security in mind, the importance of this design necessitates that its security is analyzed even in weaker scenarios in which the adversary has access to data encrypted through related keys. In parallel, a natural question that is often asked for such important targets, is whether replacing a particular component of the cipher would make it more resistant to attacks the original version is not so strong against. As the AES is weaker than expected against related-key attacks of differential nature, i.e. attacks exploiting the existence of high-probability differential characteristics, it is therefore interesting to see whether the level of security would increase against such attacks if the original key schedule of AES was replaced by an alternative one.

This question was first investigated by Nikolic in [18] just after the attacks on the full AES-192 and the full AES-256 in the related-key setting got published. In this paper, Nikolic proposed to tweak the original key schedule of all three versions of the AES by adding more rotations and some additional S-box applications but keeping a global structure quite close to the original key schedule. Much later, Khoo et al. focused only on the smallest AES version and presented an alternative key schedule for AES-128 that could ensure pure differential truncated characteristics with more active S-boxes in the related-key setting than the original key schedule [15]. A very interesting approach in this paper is that the proposed key schedule consisted of a simple byte permutation of the 16 bytes of the key state and offered for this reason excellent performances in both software and hardware. This work was further extended by Derbez et al. who automated the search for good permutations to replace the key schedule of AES-128 and used a constraint programming (CP) model to evaluate the minimum number

of active S-boxes of an AES-128 cipher with a modified key schedule [9]. This permitted them to find the first permutation reaching at least 16 active S-boxes for 5 rounds of AES-128 and a different permutation that could reach at least 20 active S-boxes for 6 rounds, thus improving the results of [15].

Our Contributions In this paper we focus on the design of alternative key schedules for all three versions of the AES. Similarly to what was done in [15] and [9], we only analyzed key schedules that are built as a byte permutation of the key state, as these key schedules have excellent implementation properties. We first prove that the CP-model used in [9] is flawed and thus all the results obtained in this paper are wrong. Then, we build our own MILP model to compute the minimum number of active S-boxes for a modified AES and investigate several strategies to search for good permutation-based key schedules. Our first strategy improves the method used in [9] which consisted in searching for good permutations by decomposing them into disjoint cycles. The idea is to build a permutation cycle by cycle and early abort when we are sure a partially formed permutation cannot be extended to a strong one. This method works well for AES-128 and permitted us to obtain many different permutations that could reach 15 active S-boxes for 5 rounds and 20 active S-boxes for 6 rounds, leading thus to better permutations than the one designed by Khoo et al. in [15]. However, this method scales badly for the other two variants. For this reason, we propose a different strategy based on two MILP models that call each other. The first model starts to search for a key schedule by having as its only constraint that this key schedule should be a permutation. This model then calls a second model that computes the minimum number of active S-boxes of any characteristic of the AES with key schedule the one found by the first model. If a characteristic activating less S-boxes than the desired bound is found, then this second model calls again the first one by adding to it extra constraints for the key schedule to prevent that such a weak characteristic reappears. This is done until a good permutation is found or until the problem has no solution. This method is efficient, as each time the first model is called, extra constraints are added on the top of the previous ones, restricting the search space more and more. This strategy permitted us to find strong permutations that can be used as the key schedule for all three AES variants. In particular, we show in the last part of this article, that the key schedules we propose would permit AES-192 and AES-256 to resist full-round boomerang attacks in the related-key setting.

The rest of the paper is organized as follows. Section 2 provides a brief description of the AES, introduces some preliminary notions on differential characteristics and introduces the Mixed Integer Linear Programming (MILP) principle. In Section 3 we recall previous results on alternative key schedules for the AES and prove that the results of [9] are wrong. Then, in Section 4 we describe our first method based on the decomposition of a permutation into cycles for AES-128. Section 5 presents our new method based on the two MILP models that call each other. Finally, our results for all versions of the AES are summarized and discussed in Section 6.

Our code is available at:

`https://github.com/pderbez/acns2024/`

2 Background

2.1 Description of the AES

The `AES` is a Substitution Permutation Network that processes data blocks of 128 bits, using keys of 128, 192 or 256 bits. The number of rounds N_r depends on the key size. It is $N_r = 10$ for `AES-128`, $N_r = 12$ for `AES-192` and $N_r = 14$ for `AES-256`. From the initial master key, $N_r + 1$ round subkeys of 128 bits are generated with a key schedule algorithm that is composed of XORs and the application of an 8-bit S-box (the same as in the round function). We refer the reader to [10] for the detailed specification of the key schedule.

Both the internal block state and each 128-bit subkey can be represented by an array of 4×4 bytes. We will use the numbering below to refer to the bytes of such a state.

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

After an initial subkey addition, the state is transformed by iterating a round function composed of four byte-oriented transformations as depicted in Figure 1. SubBytes (SB) applies to each byte of the state the same non-linear bijection called S-box. ShiftRows (SR) shifts the i-th row by i bytes to the left. MixColumns (MC) transforms each column of the state by multiplying it by an MDS (Maximum Distance Separable) matrix. Finally, AddRoundKey (ARK) XORs the state with the round subkey. For the last round, the MixColumns operation is omitted.

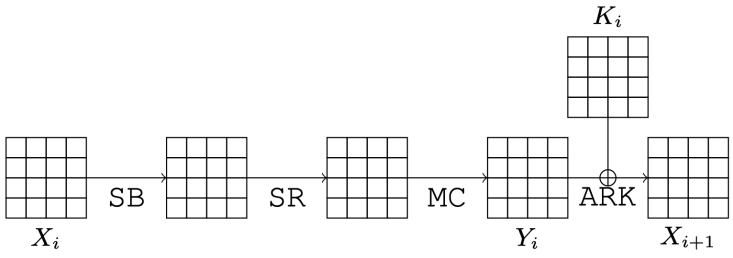

Fig. 1. The `AES` round function. X_1 is obtained by xoring the input block and the subkey K_0. The output block is X_{N_r+1}. MC is omitted for the last round.

2.2 AES Differential Characteristics

Differential cryptanalysis is a classical technique for symmetric primitives introduced in 1990 by Biham and Shamir [1]. The idea of this technique is to study the propagation of an input difference through several rounds of the cipher. In order for a cipher to be immune against this family of attacks, there should not exist an input difference a that propagates to an output difference b with a probability higher than expected for a random permutation. Such couples of input/output differences (a, b) are called *differentials*. Computing the exact probability of a differential is a very hard computational problem. In practice, we search to approximate this probability by studying sequences of differences $(a = \delta_0, \delta_1, \ldots, \delta_R = b)$ that start from the difference a and end with the difference b, and that we call *differential characteristics*. As the number of differential characteristics is too high to be exhausted, a common search method is to use a *truncated* representation of the characteristics [16]. This approach that works particularly well for word-oriented ciphers consists in abstracting each word by a Boolean value that indicates whether this word is *active*, i.e. has a non-zero difference on it, or *inactive*. Furthermore, we say that an S-box is active if there exists a non-zero difference at its input. The number of active S-boxes of a differential characteristic is an important quantity as, combined with the maximum differential probability of a non-trivial transition through the S-box, permits to provide an upper bound on the probability of any differential characteristic following the truncated pattern. The higher the number of active S-boxes, the lower the probability of a characteristic can be.

The authors of the AES employed an approach known as the *wide-trail strategy* [7] to ensure that all characteristics have, after a certain number of rounds, a relatively high number of active S-boxes. Thanks to this, the AES can be proven immune to classical differential attacks in the single-key setting. On the other hand, the attacks of Biryukov et al. [3, 4] and the works that followed showed that if differences are permitted in the key, then there exist related-key characteristics with much less active S-boxes than classical characteristics of the same length.

Modeling the Propagation of Truncated Differences through the AES Modeling the propagation of truncated differences on the AES can be done quite easily by exploiting the byte-oriented structure of the cipher. A state in a truncated differential of the AES is seen as the concatenation of 16 Boolean variables, each indicating whether the corresponding byte is active or inactive. Then the activity pattern of a byte does not change after the application of the S-box, as this operation is bijective. ShiftRows is a simple reorganization of the bytes inside the state and for MixColumns we use the fact that the matrix is MDS and that its branch number is 5. This means that the sum of the active bytes in a column before and after the application of the matrix is 0 if the column is inactive and at least 5 if the column is active. Finally, we model the AddRoundKey operation, by supposing that the XOR of two active bytes can give an active

byte or an inactive byte. We call a *pure* truncated differential characteristic a sequence of truncated differences that respect these propagation rules. A truncated related-key characteristic for 3 rounds of `AES-128` is depicted in Figure 2.

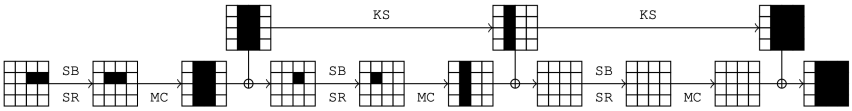

Fig. 2. A 3-round related key characteristic for `AES-128` (drawn with the library [14]).

Invalid Truncated Differential Characteristics It can happen that a truncated differential characteristic that follows the propagation rules described above cannot be instantiated with real differences. Such characteristics are called *invalid*. Some invalid characteristics can however directly be avoided by exploiting linear relations between the round function and the key schedule. An example of an invalid truncated differential characteristic is shown in Figure 3.

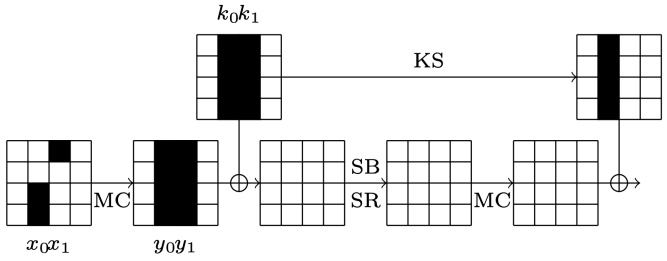

Fig. 3. Example of a linear incompatibility. The key schedule's transition is only possible if the subkeys' active columns k_0 and k_1 are equal. This equality also implies an equality between the columns y_0 and y_1. This contradicts the fact that the columns x_0 and x_1 are different.

A common method to remove such invalid characteristics consists in writing down a system of equations including all or a subset of linear relations resulting from the round function and the key schedule and apply linear algebra to it.

2.3 Mixed Integer Linear Programming

A well-known method to get a lower bound on the number of active S-boxes of a differential characteristic consists in reducing the problem to a Constraint Optimization Problem (COP) than can be solved by a dedicated solver. Among

all existing methods, the MILP (Mixed Integer Linear Program) approach became during the last decade particularly popular among cryptographers. This approach was used for the first time by Mouha et al. in [17] and by Wu and Wang in [19] to prove, among others, lower bounds on the minimal number of active S-boxes for the AES in the single key setting. In a MILP model the variables are either integers or real numbers, the constraints are linear inequalities and the objective function, if any, is a linear function of the variables. The goal is to find values for the variables such that all the constraints are satisfied and such that the objective function is optimized (i.e. maximized or minimized). This modeling technique is particularly well suited for byte-oriented ciphers like the AES: each byte is abstracted by a Boolean that indicates whether this byte is active or not; the objective function simply corresponds to the sum of all the variables that go through an S-box and each of the byte-oriented operations can easily be encoded. For instance, the XOR of 3 bytes a, b and c can be modeled with 3 linear inequalities (see Algorithm 1).

Algorithm 1: `XOR(model, a, b, c)`

```
model.addConstr(1 - a + b + c ≥ 1)
model.addConstr(a + 1 - b + c ≥ 1)
model.addConstr(a + b + 1 - c ≥ 1)
```

The constraints for the basic model for the AES are described in Algorithm 2 and 3 with the same notations as in Figure 1 (i.e. for $1 \leq \mathtt{r} \leq \mathtt{R}$, $\mathtt{X_r}$ refers to the state after the AddRoundKey operation of round `r-1`, $\mathtt{Y_r}$ corresponds to the state after the MixColumns operation and $\mathtt{K_r}$ is the subkey used in round `r`). The objective function to be minimized is returned by the function `getSboxes`. This basic model allows one to easily get a bound on the number of active S-boxes. However, all the truncated characteristics that are solutions of the model cannot be instantiated into actual characteristics. In particular, inconsistencies may come from the fact that encoding a set of linear equations between variables does not encode the vector space spanned by these equations. A first approach to reduce the number of invalid trails is to add extra variables and constraints to the model. Another approach, used by Derbez et al. in [8] is to perform linear algebra to check if the solution found by the solver respects the whole system of linear equations induced by the cipher. When a linear inconsistency is detected, the authors of [8] used the callback functionality of the solver Gurobi [13] to add new constraints during the solving process in order to prevent this inconsistency for the upcoming solutions.

Algorithm 2: `addConstrForAddRoundKey(model, R)`

```
for r = 1...R-1 do
    for i = 0...15 do
        XOR(model, X_{r+1}[i], K_r[i], Y_r[i])
```

Algorithm 3: `addConstrForShiftRowsMixColumns(model, R)`

```
for r = 1...R−1 do
    for c = 0...3 do
        e ← 0
        for i = 0...3 do
            e ← e + Y_r[c + 4i]
            e ← e + X_r[(c + i) mod 4 + 4i]
        Let f be a dummy binary variable
        model.addConstr(e ≤ 8f)
        model.addConstr(e ≥ 5f)
```

Algorithm 4: `getSboxes(model, R)`

```
output: The number of active S-boxes
Sboxes ← 0
for r = 1... R do
    for i = 0...15 do
        Sboxes ← Sboxes + X_r[i]
return Sboxes
```

3 Permutation-based Key Schedules for the AES

While the AES is secure in the single key model, its two bigger variants were shown to be vulnerable to full-round related-key attacks that exploit the existence of high probability differential characteristics for some number of rounds [3, 2]. It is widely admitted today that the success of these attacks is mainly due to weaknesses in the key schedules of AES-192 and AES-256. Furthermore, while the design of the AES round function was based on solid mathematical properties, the design of the key schedule was done with much less formal criteria in mind. It is therefore natural to ask the question whether an alternative key schedule design could strengthen the resistance of the AES against related-key differential-like attacks.

A natural idea for designing an alternative key schedule is to use a simple byte-permutation of the master key. This design choice is clearly relevant, as key schedules of this type offer excellent implementation properties both in software and hardware. This idea was investigated notably by Khoo et al. in [15], where the authors searched (among others) to replace the key schedule of AES-128 by a key schedule of this type with the goal of increasing the minimal number of active S-boxes of any differential characteristic after some rounds. The main result of this part of their paper is the discovery of a byte-permutation that could play the role of the AES-128 key schedule and that permits to reach more active S-boxes in the pure related-key setting starting from the third round. This permutation is:

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} \longrightarrow \begin{pmatrix} 14 & 15 & 12 & 13 \\ 3 & 0 & 1 & 2 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \end{pmatrix},$$

and its table representation is

$$P_1 = (5,\ 6,\ 7,\ 4,\ 8,\ 9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15,\ 2,\ 3,\ 0,\ 1).$$

This key schedule permits to achieve at least 5, 10, 14, 18 and 21 active S-boxes after respectively 3, 4, 5, 6 and 7 rounds of computation in the pure truncated differential setting, i.e. when linear inconsistencies are not taken into account. To compare, the original `AES-128` key schedule leads to only 3, 9, 11 and 13 active S-boxes for the same number of rounds in the same setting. To design this permutation, the authors of [15] started from their human-readable proof of the bound on 3 rounds of `AES-128` and searched for modifications in the design that could increase the minimum number of active S-boxes.

The problem of finding a byte-permutation to replace the key schedule of `AES`, was then further analyzed by Derbez et al. in [9]. First, they showed that, without considering linear inconsistencies, the permutation from [15] is optimal by exhibiting differential characteristics that hold for any permutation and reaching the corresponding number of active S-boxes. Then, they introduced the idea to use a more automated approach to search for good permutations and to consider the underlying equations as well, hence removing all linearly inconsistent truncated characteristics. For this search, to test the minimum number of active S-boxes of `AES-128` with a permutation playing the role of the key schedule could achieve, the authors wrote up a Constraint Programming (CP) model. As a result, they provided permutations achieving a better security than the one from [15] and gave upper bounds on the minimum number of active S-boxes a permutation could reach. More precisely, the authors proposed the permutation

$$P_2 = (4,\ 1,\ 10,\ 6,\ 7,\ 9,\ 3,\ 11,\ 8,\ 2,\ 14,\ 15,\ 12,\ 13,\ 5,\ 0),$$

that could reach at least 16 active S-boxes for 5 rounds and the permutation

$$P_3 = (14,\ 5,\ 0,\ 7,\ 4,\ 3,\ 6,\ 15,\ 9,\ 2,\ 1,\ 11,\ 13,\ 8,\ 10,\ 12)$$

that could reach at least 20 active S-boxes for 6 rounds.

3.1 Analyzing the Results of [9]

We wrote a simple MILP model to compute the minimum number of active S-boxes that could be achieved by any truncated differential characteristic for `AES-128` with a given permutation-based key schedule. The constraints to add to the model are described in Algorithms 2, 3 and 5 and the objective function to minimize (i.e. the number of active Sboxes) is returned by the function `getSboxes`. We also handled linear dependencies of the truncated differential characteristics with the same method as Derbez et al. in [8].

Algorithm 5: `addConstrForKeySchedule128(model, R, P)`

```
for r = 2...R-1 do
    for i = 0...15 do
        model.addConstr(K_r[P(i)] = K_{r-1}[i])
```

With this MILP model, we confirmed the bounds for the permutation P_1 built in [15] in the pure truncated differential model announced by the authors and showed, that by taking linear dependencies into account, this permutation actually leads to at least 19 active S-boxes for 6 rounds (see Table 5 of [15]).

However, we were not able to confirm any of the results of [9] obtained with their CP model. We checked the two proposed permutations P_2 and P_3 with our MILP model and we got that the minimum number of active S-boxes reached was much smaller than what the authors announced. More precisely, we discovered that the permutation P_2 led to a minimum number of 10 active S-boxes after 5 rounds instead of the 16 S-boxes announced, and that P_3 resulted in at least 17 active S-boxes after 6 rounds instead of the claimed 20 active S-boxes. As a proof, we provide an example of a truncated differential characteristic with 10 active S-boxes for 5 rounds with the permutation P_2 in Figure 4 and a characteristic with 17 active S-boxes for 6 rounds with the permutation P_3 is given in Figure 5. The above prove that the CP model used in [9] was flawed and thus none of the results of that paper can be considered as correct. For example, the "proof" that there exists no permutation for the key schedule permitting to reach a minimum of 18 active S-boxes after 5 rounds of `AES-128`, cannot be trusted anymore as this proof was computational and the computations were based on the badly flawed CP model.

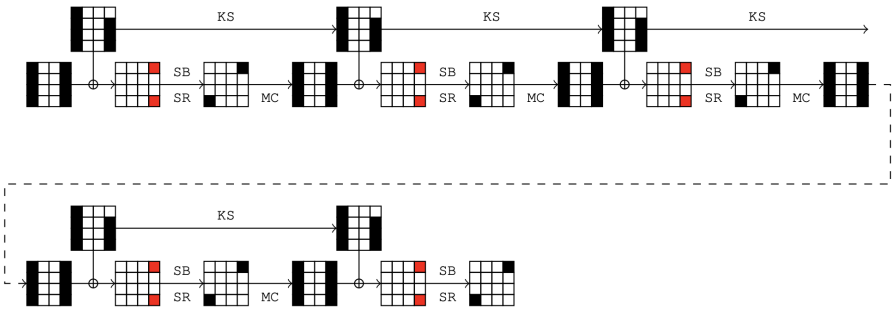

Fig. 4. Example of a truncated differential characteristic with 10 active S-boxes (in red) for 5 rounds with the permutation P_2.

We contacted the authors of [9] to let them know about our findings and after verification they confirmed there is indeed a problem with their CP model and that the results of this paper should be considered as flawed.

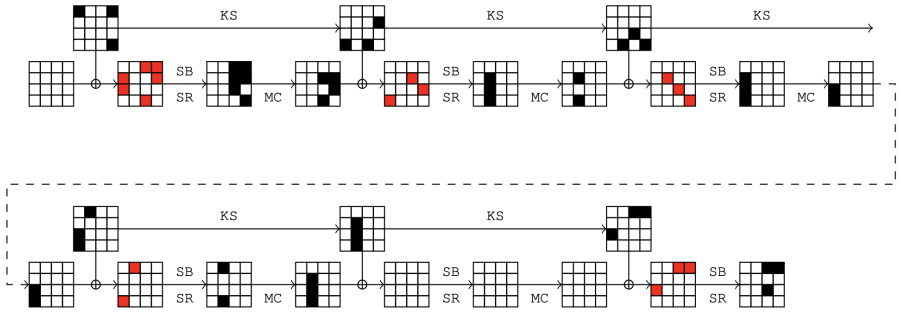

Fig. 5. Example of a truncated characteristic with 17 active S-boxes (in red) for 6 rounds with the permutation P_3.

4 A Cycle-Decomposition Approach to Search for Good Key Schedules

Once we saw that the results of [9] were wrong, we decided to search ourselves for alternative permutation-based key schedules for `AES-128`. We describe in this section the method we used to do so. This method is based on the decomposition of a permutation in its disjoint cycles and is inspired from what was done in [9]. As we will see, this method is efficient for `AES-128` but is too expensive for `AES-192` and `AES-256`. Thus, we develop a different approach in Section 5 that we successfully adapt to all three `AES` variants.

4.1 Description of the Method

The idea on which the method is based is the following. It is a well known fact that any permutation can be decomposed in cycles in a unique way. For example, the permutation P_1 above can be written as (0, 5, 9, 13, 3, 4, 8, 12, 2, 7, 11, 15, 1, 6, 10, 14) and consists of a single cycle of length 16, while the permutation P_2 can be written as (0, 4, 7, 11, 15)(2, 10, 14, 5, 9)(3, 6)(1)(8)(12)(13) and is thus decomposed in 2 cycles of length 5, a cycle of length 2 and four cycles of length 1. Viewing a permutation as a composition of cycles has an important advantage: it is possible to evaluate the quality of a permutation to play the role of the key schedule by only partially defining its decomposition in cycles. Suppose for example, that we want to verify if a permutation that has in its decomposition the cycle (0, 2, 7, 13, 15) can lead to a minimum of 15 active S-boxes after 5 rounds. Then we can write a MILP model for which the key schedule is only partially defined and the only permitted active bytes in the first subkey are among the bytes $0, 2, 7, 13$ and 15. The cycle structure permits us to know where these active bytes will be moved to by the key schedule in any of the following subkeys. If the MILP program manages to find a valid differential

characteristic with less than 15 active S-boxes then we know that we can throw away all permutations that have this cycle as part of their decomposition.

Moreover, we do not need to restrict ourselves to complete cycles and can check with this method permutations for which we have only partially specified a cycle they contain. To give an example, suppose we want to evaluate a permutation that has in its decomposition the partially defined cycle (0, 2, 7, 13, 15, ...). If the partially defined cycle is at least as long as the number of rounds we want to find a bound for, we can still partially evaluate its strength, by activating only bytes in the key for which the partial knowledge of the incomplete cycle permits us to propagate through all subkeys that are needed for the computation. If our target is again at least 15 active S-boxes for 5 rounds and if this partial evaluation permits to exhibit a characteristic with less than 15 active S-boxes, we know that we can remove all permutations that have inside cycles containing the trail $0 \rightarrow 2 \rightarrow 7 \rightarrow 13 \rightarrow 15$.

We describe now our global approach for this method. This approach is based on the recursive Algorithm 6. This algorithm takes as input the number of rounds `r` to analyze, a target bound `b` for the minimum number of active S-boxes, a table P_{KS} corresponding to the partially specified permutation the algorithm is working on, an element `x` for which we want to fix the image by the permutation and a variable `length` corresponding to the actual length of the cycle the algorithm is working on.

The first call to the algorithm is done for $\texttt{x} = 0$ and `length = 1`. The algorithm first checks (line 1) if the image of `x` has been fixed. If this is the case, meaning that the cycle is complete, the MILP-based routine `EvaluatePerm`(P_{KS}) checks whether the partial knowledge of P_{KS} permits to exhibit a characteristic activating less than `b` S-boxes. At this step, with the method of Derbez et al. in [8], we also detect linear inconsistencies using linear algebra and handle them using the callback functionality of the solver Gurobi. If a valid characteristic is found, then the algorithm returns to the instance that called it as this means that this partially defined cycle decomposition can never lead to permutations reaching more than `b` active S-boxes. On the other hand, if the routine `EvaluatePerm`(P_{KS}) returns a value higher or equal to the bound `b`, then if P_{KS} is entirely specified (line 4) this means that a permutation with the desired property has been found. If there are still values that remain to be fixed, the algorithm will start working on a new cycle, by choosing as the beginning of this new cycle the first available element `y` (line 7).

Finally, if the image of `x` has not yet been fixed meaning that the cycle is not yet complete (line 9), then if the current length of the cycle is long enough to permit an evaluation of the permutation, the routine `EvaluatePerm`(PK_S) is called (line 10). If the return value is smaller than `b` then this partially defined cycle is abandoned. Otherwise, the next available value `y` is chosen to continue the cycle (line 12), the image of `x` is set to `y` and the search continues (line 13).

An Improvement The basic algorithm described above can be improved by taking into account the column symmetry. Indeed let P_{KS} be a permutation for

Algorithm 6: `CycleSearch(r, b,` P_{KS}`, x, length)`

```
   output: All permutations reaching at least b active S-boxes for r rounds of
           AES-128
 1 if P_KS[x] has been fixed then
 2     if EvaluatePerm(P_KS) < b then
 3         return
 4     if all the images of P_KS have been fixed then
 5         return P_KS
 6     else
 7         Choose the next available value y to start a new cycle.
 8         CycleSearch(r,b,P_KS,y,1)
 9 else
10     if length >= r-1 and EvaluatePerm(P_KS) < b then
11         return
12     Find the next available value y to continue the cycle.
13     P_KS[x] = y
14     CycleSearch(r,b,P_KS,y,length + 1)
```

the key schedule and let $P_{\ggg}$ a permutation that shifts the columns of P_{KS}. Then, both the permutations P_{KS} and $P_{\ggg} \circ P_{KS} \circ P_{\ggg}^{-1}$ are equivalent and lead to exactly the same bounds. We have incorporated this observation to our algorithm to decrease the search space.

5 Double-MILP Model For Permutations

The strategy we presented in Section 4 can be hardly adapted to the bigger variants of the `AES`. The reason is that the search space becomes too big, as it necessitates going through permutations of 24 bytes for `AES-192` and 32 bytes for `AES-256`. For this reason, we present here an entirely different strategy to find good alternative permutation-based key schedules that we applied to all `AES` versions. This method combines a first MILP model to generate permutations with a second MILP model to evaluate the generated permutations. The aim of the second model is twofold. First, it detects when a permutation leads to the desired minimum number of active S-boxes. Second, it identifies bad subkeys patterns that a good permutation should prevent. This information is used to refine the constraints of the first model.

In the following, to simplify the notations, we only describe our algorithms for the case of `AES-128`. Note that they can be extended to `AES-192` and `AES-256` in a rather straightforward way.

Algorithm 7 summarizes the overall search process. For the initialization of the model `m1` that generates permutations, we add constraints to restrict the solutions of `m1` to permutation matrices of size 16×16. Some extra empirical constraints can possibly be added at this step (see discussion below). Then, we

generate a permutation with the model `m1` and evaluate it with the function `evaluateP128`. If the key schedule defined by this permutation allows a differential characteristic with too few active S-boxes, the function `evaluateP128` outputs a bad subkey pattern to be removed by calling the function `addConstrToRemoveKeyPattern128`. The form of this bad subkey pattern and the constraints to remove it will be detailed below. We repeat this until there is no more permutation matrix satisfying the constraints in `m1` or until a permutation guarantying `nbWantedSboxes` after `R` rounds of `AES-128` is found.

Algorithm 7: `searchP128(R, nbWantedSboxes)`

```
Initialize a model m1
▷ Ensure that P is a permutation matrix.
e1 ← 0
e2 ← 0
for i = 0...15 do
    for j = 0...15 do
        e1 ← e1 + P[i][j]
        e2 ← e2 + P[j][i]
m1.addConstr(e1 = 1)
m1.addConstr(e2 = 1)
▷ Generate a permutation P with the model m1 and test it
while True do
    m1.optimize()
    if No solution found then
        return
    P ← m1.getASolution()
    badKeyPattern ← evaluateP128(P, nbWantedSboxes, R)
    if badKeyPattern = ∅ then
        // P guarantees nbWantedSboxes after R rounds
        return P
    addConstrToRemoveKeyPattern128(m1, badKeyPattern)
```

The function `evaluateP128` describes a basic MILP model similar to the one we used for Algorithm 6. However, note that we do not optimize the number of active S-boxes. Instead, we only add a constraint to know whether there exists a truncated differential characteristic activating less than `nbWantedSboxes` S-boxes. Then, if such a truncated differential characteristic exists, the model will minimize the number of active bytes in the master key. This is directly related to the number of permutations for which the characteristic does hold. Indeed, we observed that in practice, valid truncated characteristics have few active key bytes (hardly more than 6) and thus, lower this number is, higher the number of permutations satisfying the pattern and as a consequence, higher the number

Algorithm 8: `evaluateP128(P, nbWantedSboxes, R)`

output:
- $\emptyset$ if there is no `R`-round characteristic with less than `NbWantedSboxes` when the alternative `AES-128` key schedule is based on the permutation `P`,
- A tuple of subkeys which leads to a characteristic with less than `NbWantedSboxes` and which minimizes the number of active bytes in the master key otherwise.

```
Initialize a model m2.
▷ Key schedule and round constraints
addConstrForKeySchedule128(m2, R, P)
addConstrForShiftRowsMixColumns(m2, R)
addConstrForAddRoundKey(m2, R)
▷ Number of active Sboxes
Sboxes ← getSboxes(m2, R)
m2.addConstr(sboxes ≥ 1)
m2.addConstr(sboxes ≤ nbWantedSboxes)
▷ Number of active bytes in the master key for AES-128
obj ← 0
for i = 0...15 do
    obj ← obj + K_1[i]
m2.addConstr(obj ≥ 1)
▷ Minimize the objective function
▷ Handle linear inconsistencies with the callback functionality of Gurobi
m2.minimize(obj)
if No solution found then
    return ∅
else
    badKeyPattern ← (K_1, K_2, K_3, ..., K_{R-1})
    return badKeyPattern
```

of permutations removed by the constraint will be. Of course, this is not always true but remains a quite reasonable assumption.

Algorithm 9: `addConstrToRemoveKeyPattern128(m1,(K_1,K_2,...,K_{R-1}))`

```
e ← 0
for r = 1, ..., R-2 do
    for a such that K_r[a] is an active byte do
        bound ← bound + 1
        for b such that K_{r+1}[b] is an active byte do
            e ← e + P[a][b]
m1.addConstr(e ≤ bound -1)
```

Removing Patterns. In order to better explain how we exploit a particular "bad" truncated differential characteristic to reduce the search space of possible permutations, let us focus on a simple example. For this, we denote by P the permutation that plays the role of the key schedule, and we write $P^2 = P \circ P$, $P^3 = P \circ P \circ P$, etc. Further, we suppose that M_P represents the permutation matrix associated to P, that is $M_P[i][j] = 1$ if $P(i) = j$ and is 0 otherwise. Assume now for instance that the active bytes of the characteristic are $\{0, 1\}$ on the first subkey and $\{2, 6\}$ on the second one and that $P(0) = 2$ and $P(1) = 6$. A naive way to discard the permutations leading to the exact same truncated characteristic is to forbid either $P(0) = 2$ or $P(1) = 6$. This can be easily done by adding the constraint $M_P[0][2] + M_P[1][6] \leq 1$. However, in many cases we can safely remove the transition $P(\{0, 1\}) = \{2, 6\}$, which includes, among others, the configuration $P(0) = 6$ and $P(1) = 2$. Being allowed to remove the transition from the first set to the second one, depends on the characteristic and more precisely on the relation between the active key bytes. If the characteristic is valid if and only if $\Delta k_0[0] = \alpha \Delta k_0[1]$ with $\alpha \neq \alpha^{-1}$, then we cannot ensure that swapping the images of $P(0)$ and $P(1)$ will not affect the validity of the truncated characteristic and thus we cannot discard the transition $P(\{0, 1\}) = \{2, 6\}$. On another hand, whenever $\alpha = \alpha^{-1}$ or if both $\Delta k_0[0]$ and $\Delta k_0[1]$ can be chosen independently, we can immediately forbid the transition between both sets by adding the constraint $M_P[0][2] + M_P[0][6] + M_P[1][2] + M_P[1][6] \leq 1$. In practice, two key bytes have to be related only to satisfy linear constraints on specific rounds but rarely on all the rounds. As a consequence, given a truncated differential characteristic, the corresponding constraint added to discard it consists in forbidding at least one transition between the sets of active bytes on two consecutive subkeys or at least one transition between actual values for powers of P.

Note that the accurate constraint makes the model more complicated since we might need to add constraints on P^2, P^3, etc. Removing the constraints on them, and by then potentially discarding good permutations, leads to a simpler and faster model which can be useful as a heuristic search algorithm. This is actually the version we used for the two bigger versions of `AES` since exhausting the whole search space would have been out of reach anyway.

An Additional Constraint for `AES-128` We tried to add several constraints to the MILP model that generates permutations for `AES-128` in order to restrict the search space while ensuring good properties. One of them permitted us to find permutations that outperform the permutation in [15]. This constraint is as follows: a byte of the state cannot be sent by the permutation to a column where the ShiftRows (SR) permutation would send it. For example byte `0` cannot be sent by the permutation to the first column while byte `4` cannot be sent to the last column. This constraint, while being simple, is quite natural as it tries to minimize the overlapping between ShiftRows and the key schedule in order to avoid cancellations between the state and the key addition. As a result we noticed that the permutation P_1 found in [15] was actually quite common as we

were able to generate a very high number of permutations achieving a minimum of 14 active S-boxes after 5 rounds and a minimum of 19 active S-boxes after 6 rounds. More importantly, we were also able to generate permutations achieving a minimum of 15 active S-boxes after 5 rounds and at least 20 active S-boxes after 6 rounds.

6 Results

In this section we present the results we obtained with the algorithms and models described in the previous sections.

6.1 Results for AES-128

We obtained several permutations reaching at least 20 active S-boxes for 6 rounds. We present two such permutations, respectively denoted by P_4 and P_5. The permutation P_4 was discovered with the method of Section 4, while P_5 was found with the method of Section 5.

$$P_4 = (6, 0, 4, 9, 13, 10, 8, 3, 7, 12, 15, 14, 11, 5, 1, 2)$$
$$P_5 = (3, 15, 11, 8, 2, 1, 10, 5, 4, 0, 9, 7, 6, 12, 13, 14)$$

The bounds for 2 to 7 rounds for these two permutations are given in Table 1. These permutations can be compared to P_1 given in [15] and to P_2 and P_3 given in [9]. These permutations P_4 and P_5 achieve better differential bounds than the one proposed by Khoo et al. but none of them is strictly better. Still, permutation P_4 reaches similar or higher bounds up to 6 rounds and ensures that no differential characteristic with a probability higher than 2^{-128} does exist on 7 rounds (assuming the best probability of a non-trivial transition through the S-box is 2^{-6} as for AES).

Rounds	2	3	4	5	6	7	Ref.
P_1	1	5	10	14	19	23	[15]
P_2	1	3	7	10	12	14	[9]
P_3	1	3	7	11	17	22	[9]
P_4	1	5	9	15	20	23	Sec. 4
P_5	1	5	10	14	20	22	Sec. 5

Table 1. Bounds on the minimal number of active S-boxes for 2 to 7 rounds for our permutations P_4 and P_5 and for the three permutations given previously by Khoo et al. [15] and Derbez et al [9].

Other Results and Open Problems We used Algorithm 6 on a cluster equipped with 128 cores to scan the space of all permutations and show that there does not exist a permutation that could lead to 18 active S-boxes for 5 rounds. We also searched with both methods of Section 4 and Section 5 to find permutations that could give at least 16 S-boxes for 5 rounds or at least 21 S-boxes for 6 rounds, but we were not able to find any such permutation. It is thus an open problem if such permutations exist.

6.2 Results for AES-192 and AES-256

For the first time, we investigate how a permutation as a key schedule could affect the resistance against differential cryptanalysis for the two bigger versions of AES. Because the search space is very big for those two variants, we only used the approach described in Section 5. Actually, it was surprisingly easy to obtain permutations leading to much stronger variants than with the original key schedules. In particular, while 9 and 13 rounds respectively are required to ensure the non-existence of differential distinguishers on both the 192 and 256-bit versions of AES, we found permutations for which only 8 and 9 rounds are enough. For AES-256 this is 4 rounds less, something we believe is quite impressive and supports the belief that the key schedule for this version was far from being optimal with respect to differential cryptanalysis.

The two permutations we propose for these two versions are:

$$P_{192} = \{2, 17, 19, 9, 13, 12, 23, 0, 4, 21, 18, 16, 10, 20, 22, 1, 11, 3, 7, 5, 15, 6, 14, 8\}$$

$$P_{256} = \{27, 16, 9, 25, 11, 13, 14, 18, 22, 21, 19, 23, 28, 31, 29, 3, 2, 15, 8, 24, 17, 1, 26, 0, 7, 20, 10, 4, 6, 30, 12, 5\}$$

These two permutations are visualized below by showing how the bytes are re-arranged inside the key state:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix} \longrightarrow \begin{pmatrix} 7 & 15 & 0 & 17 & 8 & 19 \\ 21 & 18 & 23 & 3 & 12 & 16 \\ 5 & 4 & 22 & 20 & 11 & 1 \\ 10 & 2 & 13 & 9 & 14 & 6 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \end{pmatrix} \longrightarrow \begin{pmatrix} 23 & 21 & 16 & 15 & 27 & 31 & 28 & 24 \\ 18 & 2 & 26 & 4 & 30 & 5 & 6 & 17 \\ 1 & 20 & 7 & 10 & 25 & 9 & 8 & 11 \\ 19 & 3 & 22 & 0 & 12 & 14 & 29 & 13 \end{pmatrix}$$

As already stated in the introduction, it is well-known that there exist related-key boomerang attacks on the full AES-192 and on the full AES-256. A boomerang

Note that the bounds are computed assuming the master key is filled into the first round keys. Shifting the round keys does slightly modify some of the bounds.

distinguisher is composed of two differentials and its probability mostly depends on the probability of the underlying differentials. Let denote by n_r the minimum number of active S-boxes after r rounds. A first approximation of the probability of a boomerang characteristic on R-round `AES` would be $\min_r 2^{-6\times 2(n_r+n_{R-r})}$. While we know this formula is not accurate, especially since the work of Cid et al. regarding Boomerang Connectivity Table (BCT) [6], it still gives the intuition that $n_r + n_{R-r}$ should be as high as possible to ensure good resistance against boomerang attacks. Hence, when searching for replacement permutations for both `AES-192` and `AES-256` we tried to reach 22 active S-boxes with as few rounds as possible and to optimize $\min_r n_r + n_{R-r}$ for several values of R. Since most boomerang attacks only add few rounds around their inner distinguisher, we primarily focused on $R = 10$ for `AES-192` and $R = 12$ for `AES-256`. We also decided empirically to favor 5 and 6 rounds respectively to decide between two permutations.

Rounds	2	3	4	5	6	7	8	9	10
P_{192}	0	1	5	10	13	17	22	25	28
P_{256}	0	1	2	5	10	14	16	22	26

Table 2. Bounds on the minimal number of active S-boxes for 2 to 10 rounds for the permutations P_{192} and P_{256}.

To test our permutations against boomerang cryptanalysis we modified the MILP model proposed in [8] to handle a linear key schedule. We also removed the part of the model related to the key recovery process since the complete model was too slow to finish in a reasonable time. Hence we only searched for the number of rounds after which there is no boomerang characteristic with a probability higher than 2^{-128}. As a result we obtain that 10 rounds are enough for `AES-192` and 11 rounds for `AES-256`. This is much better than with the original versions of the key schedule and it is highly unlikely that our variants could be fully broken by this cryptanalysis technique. We believe this result is important since it supports that the number of rounds set by the designers for all the different versions would have been enough to ensure full security of the `AES` family.

Generic Bounds for Reduced-round `AES-192` and `AES-256` By looking at the bounds obtained for `AES-192` and `AES-256`, one may wonder if it is possible to establish in a generic way bounds on the minimum number of active S-boxes for a modified `AES` with a permutation-based key schedule. As we show in Proposition 1 such bounds can be easily obtained for a small number of rounds and are valid for any key schedule of this form.

Proposition 1. *For* `AES-256` *or* `AES-192` *used with a permutation-based key schedule there always exist a 4-round differential characteristic with 5 or less active S-boxes and a 2-round differential characteristic with 0 active S-box. Moreover for* `AES-256` *(resp.* `AES-192`*) there exists a 3-round differential characteristic with 1 (resp. less than 2) active S-boxes.*

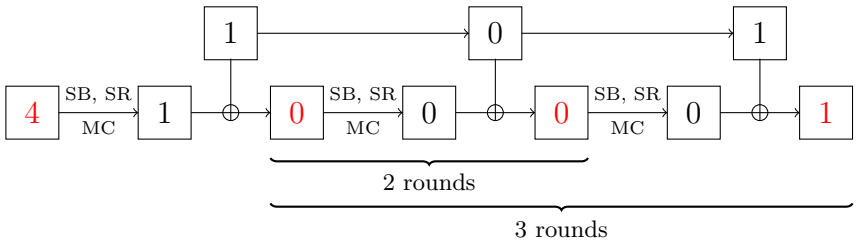

Fig. 6. A 4-round differential characteristic. The numerals represent the number of active bytes. They are depicted in red if these bytes go through an S-box.

Proof. Let us denote by $w(K_i)$ the number of active bytes of a subkey K_i. For any permutation-based key schedule for `AES-256` there exist
- a configuration such that $(w(K_1), w(K_2), w(K_3)) \in \{(1,0,0),(1,0,1)\}$;
- a configuration such that $((w(K_1), w(K_2)) = (0,1)$.

For any permutation-based key schedule for `AES-192` there exist
- a configuration such that $((w(K_1), w(K_2), w(K_3)) = (1,0,1)$;
- a configuration such that $((w(K_1), w(K_2)) \in \{(0,1),(0,2)\}$.

Together with Figure 6, the following table finishes the proof.

# rounds	$w(K_1)$	$w(K_2)$	$w(K_3)$	# active S-boxes that can be reached
4	1	0	1	5 (see the 4 rounds of Figure 6)
4	1	0	0	4 (change the last subkey of the 4 rounds)
3	0	1		1 (see the 3 rounds of Figure 6)
3	0	2		2 (change the last subkey of the 3 rounds)
2	0			0 (see the 2 rounds of Figure 6)

7 Conclusion and Open Problems

We investigate in this work two strategies to find, in an automated way, alternative permutation-based key schedules for `AES` that resist differential related-key attacks. The first one is based as in [9] on a cycle decomposition of permutations. The other one is based on two nested MILP models that generate and test permutations. We were able to confirm the results of [15] with our tool, which is an indication that our program is correct. Further, we analyzed the differential characteristics matching the lower bound and verified that for none of them

removing one of their active S-boxes was possible, which is another indication that the bounds obtained are exact. These arguments do not form of course a formal proof, but providing such a proof is extremely difficult.

Our work is a step forward to the understanding of how to design good key schedules and raises new questions. First, regarding our double-MILP model, it is natural to ask whether adding some extra constraints to the model that generates permutations can improve the search. For `AES-128` we tried to add several constraints related to the composition of permutations and their relations to the ShiftRows operation. It was in fact the simplest one that gave the best results. For the other versions of `AES`, as there is a discrepancy between the size of the permutation and the size of the subkeys, it is not clear what would good constraints for these cases be. More generally, more research efforts are needed to better understand how good key schedules for block ciphers and tweakable block ciphers should be designed and what the design criteria should be.

References

1. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO '90. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer (1990)
2. Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. Lecture Notes in Computer Science, vol. 5912, pp. 1–18. Springer (2009)
3. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and related-key attack on the full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 231–249. Springer (2009)
4. Biryukov, A., Nikolic, I.: Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to aes, camellia, khazad and others. In: Gilbert, H. (ed.) EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110, pp. 322–344. Springer (2010)
5. Boura, C., David, N., Derbez, P., Leander, G., Naya-Plasencia, M.: Differential meet-in-the-middle cryptanalysis. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part III. Lecture Notes in Computer Science, vol. 14083, pp. 240–272. Springer (2023)
6. Cid, C., Huang, T., Peyrin, T., Sasaki, Y., Song, L.: Boomerang connectivity table: A new cryptanalysis tool. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 683–714. Springer (2018)
7. Daemen, J., Rijmen, V.: The wide trail design strategy. In: Honary, B. (ed.) IMA 2001. Lecture Notes in Computer Science, vol. 2260, pp. 222–238. Springer (2001)
8. Derbez, P., Euler, M., Fouque, P., Nguyen, P.H.: Revisiting related-key boomerang attacks on AES using computer-aided tool. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part III. Lecture Notes in Computer Science, vol. 13793, pp. 68–88. Springer (2022)
9. Derbez, P., Fouque, P., Jean, J., Lambin, B.: Variants of the AES key schedule for better truncated differential bounds. In: Cid, C., Jr., M.J.J. (eds.) SAC 2018. Lecture Notes in Computer Science, vol. 11349, pp. 27–49. Springer (2018)

10. FIPS 197: Announcing the Advanced Encryption Standard (AES). National Institute for Standards and Technology, Gaithersburg, MD, USA (November 2001)
11. Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Computing AES related-key differential characteristics with constraint programming. Artif. Intell. **278** (2020)
12. Guo, J., Song, L., Wang, H.: Key structures: Improved related-key boomerang attack against the full AES-256. In: Nguyen, K., Yang, G., Guo, F., Susilo, W. (eds.) ACISP 2022. Lecture Notes in Computer Science, vol. 13494, pp. 3–23. Springer (2022)
13. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), `https://www.gurobi.com`
14. Jean, J.: TikZ for Cryptographers. `https://www.iacr.org/authors/tikz/` (2016)
15. Khoo, K., Lee, E., Peyrin, T., Sim, S.M.: Human-readable proof of the related-key security of AES-128. IACR Trans. Symmetric Cryptol. **2017**(2), 59–83 (2017)
16. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) Fast Software Encryption. pp. 196–211. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
17. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C., Yung, M., Lin, D. (eds.) Inscrypt 2011. Lecture Notes in Computer Science, vol. 7537, pp. 57–76. Springer (2011)
18. Nikolic, I.: Tweaking AES. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. Lecture Notes in Computer Science, vol. 6544, pp. 198–210. Springer (2010)
19. Wang, N., Jin, C.: Security evaluation against differential and linear cryptanalyses for feistel ciphers. Frontiers Comput. Sci. China **3**(4), 494–502 (2009)