# Improved Key Recovery Attacks on Reduced-Round AES with Practical Data and Memory Complexities

Achiya Bar-On[1], Orr Dunkelman[2], Nathan Keller[1],
Eyal Ronen[3], and Adi Shamir[3]

[1] Department of Mathematics, Bar-Ilan University, Israel
[2] Computer Science Department, University of Haifa, Israel
[3] Computer Science Department, The Weizmann Institute, Rehovot, Israel

**Abstract.** Determining the security of AES is a central problem in cryptanalysis, but progress in this area had been slow and only a handful of cryptanalytic techniques led to significant advancements. At Eurocrypt 2017 Grassi et al. presented a novel type of distinguisher for AES-like structures, but so far all the published attacks which were based on this distinguisher were inferior to previously known attacks in their complexity. In this paper we combine the technique of Grassi et al. with several other techniques to obtain the best known key recovery attack on 5-round AES in the single-key model, reducing its overall complexity from about $2^{32}$ to about $2^{22.5}$. Extending our techniques to 7-round AES, we obtain the best known attacks on AES-192 which use practical amounts of data and memory, breaking the record for such attacks which was obtained 18 years ago by the classical Square attack.

## 1 Introduction

The Advanced Encryption Standard (AES) is the best known and most widely used secret key cryptosystem, and determining its security is one of the most important problems in cryptanalysis. Since there is no known attack which can break the full AES significantly faster than via exhaustive search, researchers had concentrated on attacks which can break reduced round versions of AES. Such attacks are important for several reasons. First of all, they enable us to assess the remaining security margin of AES, defined by the ratio between the number of rounds which can be successfully attacked and the number of rounds in the full AES. In addition, they enable us to develop new attack techniques which may become increasingly potent with additional improvements. Finally, there are many proposals for using reduced round AES (and especially its 4 or 5 rounds versions) as components in larger schemes, and thus successful cryptanalysis of these variants can be used to attack those schemes. Examples of such proposals include ZORRO [17], LED [21] and AEZ [22] which use 4-round AES, and WEM [7], Hound [16], and ELmD [3] which use 5-round AES.

Over the last twenty years, dozens of papers on the cryptanalysis of reduced-round AES were published, but only a few techniques led to significant reductions

in the complexity of key recovery attacks. In the standard model (where the attack uses a single key rather than related keys), these techniques include the Square attack [8, 15], impossible differential cryptanalysis [1, 23], the Demirci-Selçuk attack [10, 12], and the Biclique attack [2]. In most of these cases, it took several years -- and a series of subsequent improvements — from the invention of the technique until it was developed into its current form. For example, impossible differential cryptanalysis was applied to AES already in 2000 [1] as an attack on 5-round AES, but it was only very recently that Boura et al. [6] improved it into its best currently known variant which breaks 7-round AES with an overall complexity of about $2^{107}$. The Demirci-Selçuk attack was presented in 2005 [10] with a huge memory complexity of over $2^{200}$, and it took 8 years before Derbez et al. [12] enhanced it in 2013 into an attack on 7-round AES with an overall complexity which is just below $2^{100}$. Therefore, the development of any new attack technique is a major breakthrough with potentially far reaching consequences.

The latest such development happened in 2017, when Grassi et al. [20] published a new property of AES, called *multiple-of-8*, which had not been observed before by other researchers. At first, it was not clear whether the new observation can at all lead to attacks on AES which are competitive with respect to previously known results. This question was partially resolved by Grassi [19], who used this observation to develop a new type of attack which can break 5-round AES in data, memory and time complexities of $2^{32}$. However, a variant of the Square attack [15] can break the same variant with comparable data and time complexities but with a much lower memory complexity of $2^9$. Consequently, the new technique did not improve the best previously known attack on 5 rounds, and its extensions to more than 5 rounds (see [19]) were significantly inferior to other attacks.

In this paper we greatly improve Grassi's attack, and show how to attack 5-round AES in data, memory and time complexities of less than $2^{22.5}$, which is about 500 times faster than any previous attack on the same variant. Due to the exceptionally low complexity of our attack, we could verify it experimentally by running it on real data generated from hundreds of randomly chosen keys. As we expected, the success rate of our full key recovery attack rose sharply from 0.24 to 1 as we increased the amount of available data from $2^{22}$ to $2^{23}$ in tiny increments of $2^{0.25}$.

By extending our technique to larger versions of AES, we obtain new attacks on AES-192 and AES-256 which have the best time complexity among all the attacks on 7-round AES which have practical data and memory complexities.

Low data and memory attacks were studied explicitly in a number of papers (e.g., [4, 5, 12]), but progress in applying such attacks to AES had been even slower than the progress in the "maximum complexity" metric. While some results were obtained on variants with up to 5 rounds, the best such attack on 6 and more rounds is still the improved Square attack presented by Ferguson et al. [15] in 2000. We use the observation of Grassi et al., along with the *dissection* technique [14] and several other techniques, to beat this 18-year old record and

develop the best attacks on 7-round AES in this model. In particular, our attack on 7-round AES with 192-bit keys requires $2^{30}$ data, $2^{32}$ memory and $2^{153}$ time, which outperforms the Square attack in all three complexity measures simultaneously.

A summary of the known and new key recovery attacks in the single key model on 5 and 7 rounds of AES appears in Tables 1 and 2, respectively. The specified complexities describe how difficult it is to find some of the key bytes. Since our new attacks can find with the same complexity any three key bytes which share the same generalized diagonal, we can rerun them several times for different diagonals to find the full key with only slightly elevated complexities.

| Attack | Data (Chosen plaintexts) | Memory (128-bit blocks) | Time (encryptions) |
|---|---|---|---|
| MitM [11] | 8 | $2^{56}$ | $2^{64}$ |
| Imp. Polytopic [25] | 15 | $2^{41}$ | $2^{70}$ |
| Partial Sum [26] | $2^{8}$ | small | $2^{38}$ |
| Square [9] | $2^{11}$ | small | $2^{44}$ |
| Square [9] | $2^{33}$ | $2^{32}$ | $2^{34}$ |
| Improved Square [15] | $2^{33}$ | small | $2^{33}$ |
| Yoyo [24] | $2^{11.3}$ ACC | small | $2^{31}$ |
| Imp. Diff. [1] | $2^{31.5}$ | $2^{38}$ | $2^{33}$ |
| Mixture Diff. [19] | $2^{32}$ | $2^{32}$ | $2^{32}$ |
| Our Attack (Sect. 4) | $2^{22.25}$ | $2^{20}$ | $2^{22.5}$ |

ACC Adaptive Chosen Plaintexts and Ciphertexts

**Table 1.** Attacks on 5-Round AES (partial key recovery)

The paper is organized as follows. In Section 2 we briefly describe AES and introduce our notations, and in Section 3 we describe the new 4-round distinguisher which was discovered and used by Grassi. In Section 4 we show how to exploit this distinguisher in a better way to obtain improved attacks on 5-round AES. We extend the attack to 6-round AES in Section 5, and then extend it again to 7 rounds in Section 6. In Section 7 we explore other points on the time-memory-data tradeoff curve. Section 8 summarizes our paper.

## 2 Brief Introduction to the AES

### 2.1 A Short Description of AES

The Advanced Encryption Standard (AES) [9] is a substitution-permutation network which has 128 bit plaintexts and 128, 192, or 256 bit keys. Its 128 bit internal state is treated as a byte matrix of size 4x4, where each byte represents a value in $GF(2^8)$. An AES round (described in Figure 1) applies four operations to this state matrix:

| AES Variant | Attack | Data (Chosen Plaintexts) | Memory (128-bit blocks) | Time (encryptions) |
|---|---|---|---|---|
| AES-128 | Imp. Diff. [6] | $2^{105}$ | $2^{74}$ | $2^{106.88}$ |
| | MitM [13] | $2^{97}$ | $2^{98}$ | $2^{99}$ |
| AES-192 | MitM [13] | $2^{97}$ | $2^{98}$ | $2^{99}$ |
| | MitM [12] | $2^{32}$ | $2^{129.7}$ | $2^{129.7}$ |
| | Collision [18] | $2^{32}$ | $2^{80}$ | $2^{140}$ |
| | Square [15] | $2^{36.2}$ | $2^{36.2}$ | $2^{155}$ |
| | Our Attack (Sect. 6) | $2^{30}$ | $2^{32}$ | $2^{153}$ |
| | Our Attack (Sect. 7) | $2^{32}$ | $2^{40}$ | $2^{145}$ |
| AES-256 | MitM [13] | $2^{97}$ | $2^{98}$ | $2^{99}$ |
| | MitM [12] | $2^{32}$ | $2^{133.7}$ | $2^{133.7}$ |
| | Collision [18] | $2^{32}$ | $2^{80}$ | $2^{140}$ |
| | Square [15] | $2^{36.4}$ | $2^{36.4}$ | $2^{172}$ |
| | Our Attack (Sect. 6) | $2^{30}$ | $2^{48}$ | $2^{161.6}$ |

**Table 2.** Attacks on 7-Round AES (full key recovery)

- SubBytes (SB) — applying the same 8-bit to 8-bit invertible S-box 16 times in parallel on each byte of the state,
- ShiftRows (SR) — cyclically shifting the $i$'th row by $i$ bytes to the left,
- MixColumns (MC) — multiplication of each column by a constant 4x4 matrix over the field $GF(2^8)$, and
- AddRoundKey (ARK) — XORing the state with a 128-bit subkey.

An additional AddRoundKey operation is applied before the first round, and in the last round the MixColumns operation is omitted.

For the sake of simplicity we shall denote AES with $n$-bit keys by AES-$n$. The number of rounds depends on the key length: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. The rounds are numbered $0, \ldots, Nr-1$, where $Nr$ is the number of rounds. We use 'AES' to denote all three variants of AES.

The key schedule of AES transforms the key into $Nr+1$ 128-bit subkeys. We denote the subkey array by $W[0, \ldots, 4 \cdot Nr+3]$, where each word of $W[\cdot]$ consists of 32 bits. When the length of the key is $Nk$ 32-bit words, the user supplied key is loaded into the first $Nk$ words of $W[\cdot]$, and the remaining words of $W[\cdot]$ are updated according to the following rule:

- For $i = Nk, \ldots, 4 \cdot Nr + 3$, do
  - If $i \equiv 0 \bmod Nk$ then $W[i] = W[i-Nk] \oplus SB(W[i-1] \lll 8) \oplus RCON[i/Nk]$,
  - else if $Nk = 8$ and $i \equiv 4 \bmod 8$ then $W[i] = W[i-8] \oplus SB(W[i-1])$,
  - Otherwise $W[i] = W[i-1] \oplus W[i-Nk]$,

where $\lll$ denotes rotation of the word by 8 bits to the left, and $RCON[\cdot]$ is an array of predetermined constants.

### 2.2 Notations

In the sequel we use the following definitions and notations.

The state matrix at the beginning of round $i$ is denoted by $x_i$, and its bytes are denoted by $0, 1, 2, \ldots, 15$, as described in Figure 1. Similarly, the state matrix after the SubBytes and the ShiftRows operations of round $i$ are denoted by $x'_i$ and $x''_i$, respectively. The difference between two values in state $x_i$ is denoted by $\Delta(x_i)$. We use this notation only when it is clear from the context which are the values whose difference we refer to.

We denote the subkey of round $i$ by $k_i$, and the first (whitening) key by $k_{-1}$, i.e., $k_i = W[4 \cdot (i+1)]||W[4 \cdot (i+1)+1]||W[4 \cdot (i+1)+2]||W[4 \cdot (i+1)+3]$. In some cases, we are interested in interchanging the order of the MixColumns operation and the subkey addition. As these operations are linear they can be interchanged, by first XORing the data with an equivalent subkey and only then applying the MixColumns operation. We denote the equivalent subkey for the altered version by $u_i$, i.e., $u_i = MC^{-1}(k_i)$. The bytes of the subkeys are numbered by $0, 1, \ldots, 15$, in accordance with the corresponding state bytes.

In cases when we interchange the order of the MixColumns operation of round $i$ and the subkey addition, we denote the state right after the subkey addition (and just before the MixColumns operation) by $\bar{x}_i$.

The plaintext is sometimes denoted by $x_{-1}$, and so $x_0 = x_{-1} \oplus k_{-1}$.

The $j$'th byte of the state $x_i$ is denoted $x_{i,j}$. When several bytes $j_1, \ldots, j_\ell$ are considered simultaneously, they are denoted $x_{i,\{j_1,\ldots,j_\ell\}}$. When a full column is considered, it is denoted $x_{i,\mathrm{Col}(j)}$, and if several columns are considered simultaneously, we denote them by $x_{i,\mathrm{Col}(j_1,\ldots,j_\ell)}$.

Sometimes we are interested in 'shifted' columns, i.e., the result of the application of ShiftRows to a set of columns. This is denoted by $x_{i,SR(\mathrm{Col}(j_1,\ldots,j_\ell))}$. Similarly, a set of 'inverse shifted' columns (i.e., the result of the application of $SR^{-1}$ to a set of columns) is denoted by $x_{i,SR^{-1}(\mathrm{Col}(j_1,\ldots,j_\ell))}$.

In the attacks on 5-round AES (both Grassi's attack and our attack), we consider encryptions of a quartet of values. To simplify notations, while the plaintext/ciphertext pairs are denoted by $(P_j, C_j)$, $j = 1, \ldots, 4$, we denote the intermediate values by $(x_i, y_i, z_i, w_i)$, where $x_i$ corresponds to the encryption process of $P_1$ and so $x_{-1} = P_1$, $y_i$ corresponds to the encryption process of $P_2$ and so $y_{-1} = P_2$, etc.

In the attacks on 6-round and 7-round AES, we consider encryptions of several (e.g., 4 or 8) pairs of values. To simplify notations, in this case we denote the plaintext pairs by $(P_j, \hat{P}_j)$, $j = 1, \ldots, 8$, the corresponding ciphertext pairs by $(C_j, \hat{C}_j)$, $j = 1, \ldots, 8$, and the corresponding pairs of intermediate values by $(x^j_{i,\ell}, \hat{x}^j_{i,\ell})$, for $j = 1, \ldots, 8$.

In all attacks, we exploit plaintext pairs $(P, \hat{P})$ for which the corresponding intermediate values satisfy $\Delta(x''_{4,SR(\mathrm{Col}(0))}) = 0$ (i.e., have a zero difference in the first shifted column just before the MixColumns operation of round 4). Hence, throughout the paper we call such pairs *good pairs*.

Finally, we measure the time complexity of all the attacks in units which are equivalent to a single encryption operation of the relevant reduced round variant

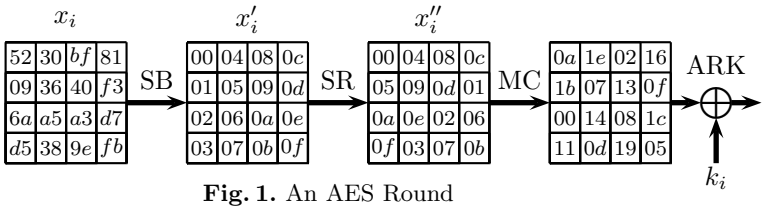


**Fig. 1.** An AES Round

of AES. We measure the space complexity in units which are equivalent to the storage of a single plaintext (namely, 128 bits). To be completely fair, we count all operations carried out during our attacks, and in particular we do not ignore the time and space required to prepare the various tables we use.

## 3 The 4-round Distinguisher of Grassi

In this section we present the distinguisher for 4-round AES, which serves as the basis to all our attacks. The distinguisher was presented by Grassi [19], as a variant of the 5-round distinguisher introduced at Eurocrypt'17 by Grassi et al. [20]. Note that the distinguisher holds in a more general setting than the one presented here. For sake of simplicity, we concentrate on the special case used in our attacks.

**Definition 1.** *Let $x_i, y_i$ be two intermediate values at the input to round $i$ of AES, such that $x_{i,\mathrm{Col}(1,2,3)} = y_{i,\mathrm{Col}(1,2,3)}$ (i.e., $x_i$ and $y_i$ may differ only in the first column). We say that $(z_i, w_i)$ is a mixture of $(x_i, y_i)$ if for each $j = 0, 1, 2, 3$, the unordered pairs $(x_{i,j}, y_{i,j})$ and $(z_{i,j}, w_{i,j})$ are equal. That is, either the $j$'th bytes of $z_i$ and $w_i$ are equal to those of $x_i$ and $y_i$, respectively, or they are swapped. In such a case, $(x_i, y_i, z_i, w_i)$ is called a* mixture quadruple.

*Remark 1.* Note that for each $(x_i, y_i)$ such that $x_{i,j} \neq y_{i,j}$ for all $j = 0, 1, 2, 3$, there are 7 possible (unordered) mixtures that can be represented by vectors in $\{0, 1\}^4$ which record whether $z_{i,j}$ is equal to $x_{i,j}$ or to $y_{i,j}$, for $j = 0, 1, 2, 3$. For example, (1000) corresponds to the mixture $(z_i, w_i)$ such that $z_{i,\mathrm{Col}(0)} = (x_{i,0}, y_{i,1}, y_{i,2}, y_{i,3})$ and $w_{i,\mathrm{Col}(0)} = (y_{i,0}, x_{i,1}, x_{i,2}, x_{i,3})$.

**Observation 1.** *Let $(x_i, y_i, z_i, w_i)$ be a mixture quadruple of intermediate values at the input to round $i$ of AES. Then the corresponding intermediate values $(x_{i+2}, y_{i+2}, z_{i+2}, w_{i+2})$ sum up to zero, i.e.,*

$$x_{i+2} \oplus y_{i+2} \oplus z_{i+2} \oplus w_{i+2} = 0. \tag{1}$$

*Consequently, if for $j \in \{0, 1, 2, 3\}$ we have $x_{i+2,SR^{-1}(\mathrm{Col}(j))} \oplus y_{i+2,SR^{-1}(\mathrm{Col}(j))} = 0$, then the corresponding intermediate values $(x''_{i+3}, y''_{i+3}, z''_{i+3}, w''_{i+3})$ (i.e., just before the MixColumns operation of round $i + 3$) satisfy*

$$x''_{i+3,SR(\mathrm{Col}(j))} \oplus y''_{i+3,SR(\mathrm{Col}(j))} = z''_{i+3,SR(\mathrm{Col}(j))} \oplus w''_{i+3,SR(\mathrm{Col}(j))} = 0.$$

*Proof.* Let $(x_i, y_i, z_i, w_i)$ be as in the assumption. The mixture structure is preserved through the SubBytes operation of round $i$, and then ShiftRows spreads

the active bytes between the columns, such that each column contains exactly one of them. As a result, for each $j \in \{0, 1, 2, 3\}$, the unordered pairs $(x''_{i,\mathrm{Col}(j)}, y''_{i,\mathrm{Col}(j)})$ and $(z''_{i,\mathrm{Col}(j)}, w''_{i,\mathrm{Col}(j)})$ are equal. This property is clearly preserved by MixColumns and by the subsequent AddRoundKey and SubBytes operations. It follows that the intermediate values $(x'_{i+1}, y'_{i+1}, z'_{i+1}, w'_{i+1})$ sum up to zero. As ShiftRows, MixColumns, and AddRoundKey are linear operations, this implies $x_{i+2} \oplus y_{i+2} \oplus z_{i+2} \oplus w_{i+2} = 0$.

Now, if for some $j$ we have $x_{i+2,SR^{-1}(\mathrm{Col}(j))} \oplus y_{i+2,SR^{-1}(\mathrm{Col}(j))} = 0$, then by the round structure of AES we have $x_{i+3,\mathrm{Col}(j)} \oplus y_{i+3,\mathrm{Col}(j)} = 0$, and thus, $x''_{i+3,SR(\mathrm{Col}(j))} \oplus y''_{i+3,SR(\mathrm{Col}(j))} = 0$. Furthermore, by (1) we have $z_{i+2,SR^{-1}(\mathrm{Col}(j))} \oplus w_{i+2,SR^{-1}(\mathrm{Col}(j))} = 0$, and thus by the same reasoning as for $(x, y)$, we get $z''_{i+3,SR(\mathrm{Col}(j))} \oplus w''_{i+3,SR(\mathrm{Col}(j))} = 0$, as asserted.

Grassi [19] used his distinguisher to mount an attack on 5-round AES with data, memory, and time complexities of roughly $2^{32}$. The attack algorithm is given in Algorithm 1.

**Algorithm 1** Grassi's 5-Round Attack

1: Ask for the encryption of $2^{32}$ chosen plaintexts in which $SR^{-1}(\mathrm{Col}(0))$ assumes all $2^{32}$ possible values and the rest of the bytes are constant.
2: Find a pair of ciphertexts $(C_1, C_2) = (x_5, y_5)$ with zero difference in $SR(\mathrm{Col}(0))$.
3: **for** each guess of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$ **do**
4: Partially encrypt the corresponding plaintexts $(P_1, P_2) = (x_{-1}, y_{-1})$ through AddRoundKey and round 0 to obtain $(x_1, y_1)$.
5: Let $(z_1, w_1)$ be a mixture of $(x_1, y_1)$, partially decrypt it to find the corresponding plaintext pair $(P_3, P_4) = (z_{-1}, w_{-1})$, and denote the corresponding ciphertexts by $(C_3, C_4) = (z_5, w_5)$.
6: **if** $(z_5, w_5)$ does not satisfy $z_{5,SR(\mathrm{Col}(0))} \oplus w_{5,SR(\mathrm{Col}(0))} = 0$ **then**
7: discard the key guess $k_{-1,SR^{-1}(\mathrm{Col}(0))}$.
8: **end if**
9: **end for**
10: Repeat Steps (1)–(8) for the other three columns, and check the remaining key guesses by trial encryption.

The structure of chosen plaintexts is expected to contain about $2^{63} \cdot 2^{-32} = 2^{31}$ pairs for which the ciphertexts have a zero difference in $SR(\mathrm{Col}(0))$. The adversary can find one of them easily in time $2^{32}$, using a hash table. Step 3 of the attack requires only a few operations for each key guess. Since $(x_1, y_1, z_1, w_1)$ form a mixture quadruple, by Observation 1 we know that if $(x_5, y_5)$ have zero difference in $SR(\mathrm{Col}(0))$, then we must have $z_{5,SR(\mathrm{Col}(0))} \oplus w_{5,SR(\mathrm{Col}(0))} = 0$. (Note that the MixColumns operation in the last round is omitted, and thus, the difference in the state $z_5$ is equal to the difference in the state $z''_4$ discussed in Observation 1.) Therefore, if the condition fails, we can safely discard the key guess. The probability of a random key to pass this filtering is $2^{-32}$, and thus, we expect only a few key guesses to remain. Thus, the data, memory, and time

complexities for recovering 32 key bits are $2^{32}$, and for recovering the full key are $2^{34}$.

## 4 Improved Attack on 5-Round AES

In this section we present our improved attack on 5-round AES, which requires less than $2^{22.5}$ data, memory, and time to recover 24 key bits and less than $2^{25.5}$ data, memory, and time to recover the full key. This is the first attack on 5-round AES whose all complexities are below $2^{32}$. The attack was fully verified experimentally.

Our attack is based on Grassi's attack and enhances it using several observations. First we present and analyze the observations, then we present the attack algorithm and analyze its complexity, and finally we describe the experiments we performed to verify the attack.

### 4.1 The Observations behind the Attack

**1. Reducing the data complexity to $2^{24}$.** Our first observation is that we can reduce the amount of data significantly, and still find the mixture quadruple we need for Grassi's attack. Indeed, as mentioned above, when we start with $2^{32}$ plaintexts, it is expected that the data contains about $2^{31}$ mixture quadruples, while we need only one mixture quadruple for the attack.

Instead, we may start with $2^{24}$ plaintexts taken arbitrarily from the structure of size $2^{32}$ used in Grassi's attack. These plaintexts form $2^{47}$ pairs, and we expect that in $2^{15}$ of them, the ciphertexts have zero difference in $SR(\mathrm{Col}(0))$. Fix one such pair, $(C_1, C_2) = (x_5, y_5)$. For each guess of the 32 bits of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$, and for each of the 7 possible types of mixture, the probability that the mixture of $(x_1, y_1)$ is contained in our data set is $(2^{24}/2^{32})^2 = 2^{-16}$. As there are $2^{15}$ possible pairs $(x_5, y_5)$ and 7 possible types of mixture, we expect that with probability $1 - (1 - 2^{-16})^{7 \cdot 2^{15}} \approx 0.97$, that the data contains a mixture quadruple with respect to the correct value of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$, which is sufficient for mounting the attack. Hence, the data complexity can be reduced to $2^{24}$ chosen plaintexts. As the memory is used only to store and filter the data, the memory complexity is reduced to $2^{24}$, as well.

However, if we simply apply Grassi's attack with the reduced number of plaintexts, its time complexity is increased significantly due to the need to go over the $2^{15}$ pairs of $(x_5, y_5)$ for each key guess. This will be resolved in the next observations.

**2. Reducing the time complexity by changing the order of operations.** Our second observation is that if $(x_1, y_1, z_1, w_1)$ is a mixture quadruple then $x_1 \oplus y_1 \oplus z_1 \oplus w_1 = 0$, and consequently, $x_0'' \oplus y_0'' \oplus z_0'' \oplus w_0'' = 0$ as well. This allows to perform a preliminary check of whether $(x_1, y_1, z_1, w_1)$ can be a mixture quadruple, by checking a local condition for each of the bytes in $k_{-1,SR^{-1}(\mathrm{Col}(0))}$ separately (i.e., bytes 0,5,10,15). That is, given a quartet of

**Algorithm 2** Efficient Guessing of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$

1: **for** each each guess of $k_{-1,0}$ **do**
2: Compute the corresponding differences $x''_{0,0} \oplus y''_{0,0}$ and $z''_{0,0} \oplus w''_{0,0}$.
3: **if** $x''_{0,0} \oplus y''_{0,0} \neq z''_{0,0} \oplus w''_{0,0}$ **then**
4: Discard the guess of $k_{-1,0}$.
5: **end if**
6: **end for**
7: Repeat the above steps for bytes 5,10,15 of $k_{-1}$ and bytes 1,2,3 of $x''$, $y''$, $z''$, and $w''$, respectively.
8: **for** each remaining guess of $k_{-1,\{SR^{-1}(\mathrm{Col}(0))\}}$ **do**
9: Encrypt the quartet through round 0.
10: Check whether the values $(x_1, y_1, z_1, w_1)$ constitute a mixture quadruple.
11: **end for**

plaintexts $(P_1, P_2, P_3, P_4)$, we can perform the check whether it is a mixture quadruple using Algorithm 2.

We can use this procedure to replace the guess of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$ performed in Grassi's attack. Specifically, as described above, given $2^{24}$ plaintexts, we expect $2^{15}$ pairs in which the ciphertexts have zero difference in $SR(\mathrm{Col}(0))$. We take all $2^{29}$ pairs of such pairs, and the procedure is applied for each of them.

As Steps 1–7 offer a 32-bit filtering condition, it is expected that only a few suggestions of the key $k_{-1,SR^{-1}(\mathrm{Col}(0)}$ pass to Steps 8–11. Then, each suggestion is checked using a 1-round partial encryption. It is clear that if the data set contains a mixture quadruple (which occurs with a decent probability as described above), then the procedure will succeed for the right guess of $k_{-1,SR^{-1}(\mathrm{Col}(0)}$. For a wrong guess, the probability to pass Steps 1 and 2 is $2^{-64}$, and so all wrong guesses are expected to be discarded.

Let us analyze the complexity of the attack. In Steps 1–6 we go over the $2^8$ possible values of $k_{-1,0}$, and check the condition for each of them separately. The same goes for each repetition of Step 7. The complexity of Steps 8-11 is even lower. Hence, the overall complexity of the attack is $2^{29} \cdot 2^8 \cdot 4 = 2^{39}$ applications of a single S-box, which are roughly equivalent to $2^{33}$ encryptions.

**3. Reducing the time complexity even further by using a precomputed table.** We can further reduce the time complexity of this step using a precomputed table of size $2^{21.4}$ bytes. To construct the table, we consider each quartet of inputs to SubBytes of the form $(0, a, b, c)$, where $(a, b, c)$ are arranged in increasing order (e.g., as numbers in base 2).[1] For each quartet, we go over the $2^8$ values of the key byte $\hat{k}$ and store in the entry $(a, b, c)$ of the table the values of $\hat{k}$ for which

$$SB(\hat{k}) \oplus SB(a \oplus \hat{k}) \oplus SB(b \oplus \hat{k}) \oplus SB(c \oplus \hat{k}) = 0. \tag{2}$$

[1] As the quartets in which in some byte, not all four values are distinct, are less than 7% of the quartets, we can remove them from the analysis for sake of simplicity, with a negligible effect on the attack's complexity.

It is expected that a single value of $\hat{k}$ satisfies Condition (2). Now, if we are given a quartet $(x, y, z, w)$ of plaintext bytes and want to find the value of $k_{-1,0}$ such that the four intermediate values after SubBytes sum up to zero, we do the following:

1. Consider the quartet $(0, y \oplus x, z \oplus x, w \oplus x)$.
2. Reorder it using the binary ordering to obtain $(0, a, b, c)$ with $a < b < c$. Then access the table at the entry $(a, b, c)$ and retrieve the value $\hat{k}$.
3. Set $k_{-1,0} = \hat{k} \oplus x$.

The key $k_{-1,0}$ we found is indeed the right one, since the values after the addition of $k_{0,-1}$ are $(\hat{k}, \hat{k} \oplus y, \hat{k} \oplus z, \hat{k} \oplus w)$, and thus Condition (2) means exactly that the four values after SubBytes sum up to zero.

The table requires $2^{24}/3! \approx 2^{21.4}$ bytes of memory. In a naive implementation, its generation requires $2^{21.4} \cdot 2^8 = 2^{29.4}$ applications of a single S-box and a few more XOR operations, which is less than $2^{23.4}$ 5-round encryptions. However, it can be generated much faster, as follows.

Instead of going over all triplets $(a, b, c)$ and for each of them going over all values of $\hat{k}$, we go over triplets $a, b, \hat{k}$. For each of them, we compute $t = SB(\hat{k}) \oplus SB(a \oplus \hat{k}) \oplus SB(b \oplus \hat{k})$. We know that Condition 2 holds for $(a, b, c, \hat{k})$ if and only if $SB(c \oplus \hat{k}) = t$, or equivalently, $c = SB^{-1}(t) \oplus \hat{k}$. (Note that this value may not be unique). Therefore, we write $\hat{k}$ in the table entry/entries of $(a, b, SB^{-1}(t) \oplus \hat{k})$ and move to the next value of $\hat{k}$. In this way, the table generation requires less than $2^{24}$ S-box applications, which is negligible with respect to other steps of the attack.

Once the table is constructed, Step 1 of the procedure described in Improvement 2 can be performed by 4 table lookups. Hence, the total time complexity of the attack is reduced to $2^{29}$ times (4 table lookups + one round of encryption), which is less than $2^{29}$ encryptions.

**4. Reducing the overall complexity to $2^{22.25}$ by a wise choice of plaintexts.** So far, we reduced the data and memory complexity to $2^{24}$ and the time complexity to $2^{29}$. We show now that all three parameters can be reduced to about $2^{22.25}$ by a specific choice of the plaintexts.

Recall that in Improvement 1 we assumed that the $2^{24}$ plaintexts are arbitrarily taken from the structure of size $2^{32}$ used in Grassi's attack (in which $SR^{-1}(\text{Col}(0))$ assume all possible values and the rest of the bytes are constant). Instead of doing this, we choose all plaintexts such that byte 0 is constant in all of them. We claim that this significantly increases the probability of a plaintext quartet to form a mixture quadruple.

Indeed, let us fix $(P_1, P_2)$ and a type of mixture, and estimate the probability that for another pair $(P_3, P_4)$, the intermediate values $(x_1, y_1, z_1, w_1)$ form a mixture quadruple of the fixed type. The check can be performed in two steps, like in the procedure described in Improvement 2. First we check whether the corresponding intermediate values $(x_1, y_1, z_1, w_1)$ sum up to zero (a 32-bit condition), and only then we check that the quadruple is indeed a mixture (which is a 31-bit condition, since as we already know that the four values sum up to

zero, once the condition for $z_1$ holds, the condition for $w_1$ holds for free, and there are two possibilities for the ordering between $z_1$ and $w_1$). As described in Improvement 2, the first condition is translated to four independent conditions of the form $x''_{0,j} \oplus y''_{0,j} \oplus z''_{0,j} \oplus w''_{0,j} = 0$, for $j \in \{0, 1, 2, 3\}$. In our case, due to the choice of plaintexts, the condition in byte 0 holds for free! Therefore, the overall probability is boosted from $2^{-63}$ to $2^{-55}$.

On the other hand, we note that only three of the 7 types of mixture are possible in this case. Indeed, in the mixtures of types $(1000), (0100), (0010)$ and $(0001)$, there are two of the values $(x_1, y_1, z_1, w_1)$ which differ in a single byte. As all four values $x''_0, y''_0, z''_0, w''_0$ agree on byte 0, this is impossible since the branching number of the MixColumns operation is 5 (which means that for any pair of inputs/outputs to MC, the number of bytes in which the inputs differ plus the number of bytes in which the outputs differ is at least 5).

Therefore, the probability of $(P_3, P_4)$ taken from our structure to lead into a mixture quadruple with $(P_1, P_2)$ is expected to be $3 \cdot 2^{-55}$. This allows us to reduce the data complexity, and consequently, also the memory and time complexities.

Assume that we start with $2^{22.25}$ plaintexts, taken arbitrarily from the $2^{24}$ plaintexts that assume all values in bytes $5, 10, 15$ and have all other bytes constant. These plaintexts form $(2^{22.25})^2/2 = 2^{43.5}$ unordered pairs, and thus, $2^{86}$ unordered pairs of pairs. The probability that a pair-of-pairs gives rise to a mixture quadruple is $3 \cdot 2^{-55}$. Hence, we expect $3 \cdot 2^{-55} \cdot 2^{86} = 3 \cdot 2^{31}$ mix quadruples. With a 'decent' probability, in at least one of them, the ciphertexts $(C_1, C_2)$ have zero difference in $SR(\text{Col}(0))$, and thus, it can be used for the attack.

In the attack, we first insert the ciphertexts into a hash table to find the pairs for which the ciphertexts have zero difference in $SR(\text{Col}(0))$. It is expected that $2^{43.5} \cdot 2^{-32} = 2^{11.5}$ pairs are found. Then, for each of the $(2^{11.5})^2/2 = 2^{22}$ pairs-of-pairs, we check whether the corresponding intermediate values $(x_1, y_1, z_1, w_1)$ constitute a mixture quadruple, as described in Improvement 3. Thus, the time complexity is reduced by a factor of $2^7$ (as we have to check $2^{22}$ quartets instead of $2^{29}$), and so the time complexity is less than $2^{22}$ encryptions, which is less than the time required for encrypting the plaintexts.

**5. Reducing the data complexity a bit further by checking several columns instead of one.** Finally, the data complexity can be reduced a bit further by considering not only plaintext pairs for which the ciphertexts have zero difference in $SR(\text{Col}(0))$, but also pairs for which the ciphertexts satisfy the same condition for one of the columns 1,2,3. This increases the probability of a quartet to be useful for the attack by a factor of 4, and thus, allows us to reduce the data complexity by another factor of $4^{1/4} = \sqrt{2}$. On the other hand, this requires to use four hash tables to filter the ciphertexts (each corresponding to a different shifted column of the ciphertext), and thus, increases the memory complexity by a factor of 4. As this is not a clear improvement but rather a data/memory tradeoff, we do not include it in the attack algorithm below.

### 4.2 The Attack Algorithm and its Analysis

The algorithm of our 5-round attack is given in Algorithm 3.

**Algorithm 3** Efficient 5-Round Attack

**Preprocessing**

Initialize an empty table $T$.
**for** all $a < b < c$ **do**
    Store in $T[a, b, c]$ all bytes values $\hat{k}$ which satisfy $SB(\hat{k}) \oplus SB(a \oplus \hat{k}) \oplus SB(b \oplus \hat{k}) \oplus SB(c \oplus \hat{k}) = 0$.
**end for**

**Online phase**

Ask for the encryption of $2^{22.25}$ chosen plaintexts in which bytes $5, 10, 15$ assume different values and the rest of the bytes are constant.
Store in a list $L$ all ciphertext pairs $(C_1, C_2)$ such that $C_{1,SR(\mathrm{Col}(0))} \oplus C_{2,SR(\mathrm{Col}(0))} = 0$.
**for** all pairs of pairs $(C_1, C_2), (C_3, C_4) \in L$ **do**
    Let the corresponding plaintexts be $(P_1, P_2, P_3, P_4) = (x_{-1}, y_{-1}, z_{-1}, w_{-1})$, respectively.
    Compute the values $(y_{-1,5} \oplus x_{-1,5}, z_{-1,5} \oplus x_{-1,5}, w_{-1,5} \oplus x_{-1,5})$, and sort the three bytes in an increasing order to obtain $(a, b, c)$.
    **for** each value $\hat{k}$ in $T[a, b, c]$ **do**
        Store in $L_5$ the value $k_{-1,5} = \hat{k} \oplus x_{-1,5}$.
    **end for**
    Repeat the above steps for bytes 10 and 15 (with lists $L_{10}$ and $L_{15}$, respectively).
    **for** all subkey candidates $(k_{-1,5} \in L_5, k_{-1,10} \in L_{10}, k_{-1,15} \in L_{15})$ **do**
        Partially encrypt $(P_1, P_2, P_3, P_4)$ through round 0 and compute $x_1, y_1, z_1, w_1$.
        **if** $(x_1, y_1, z_1, w_1)$ does not constitute a mixture quadruple **then**
            Discard subkey candidate.
        **end if**
    **end for**
**end for**
Output all guesses of $k_{-1,5}, k_{-1,10}, k_{-1,15}$ which remained.

As described in Improvement 4, $T$ can be prepared in time of $2^{21.4}$ S-box evaluations, and contains $2^{21.4}$ byte values. Steps 5,6 can be easily performed in time $2^{22.25}$ using a hash table of size $2^{22.25}$ 24-bit values, which are less than $2^{20}$ 128-bit blocks. The expected size of the list $L$ is $2^{43.5} \cdot 2^{-32} = 2^{11.5}$. Hence, Steps 7–20 are performed for $2^{22}$ pairs of pairs. As described in Improvement 4, these steps take less than a single encryption for each quartet, and thus, their total complexity is less than $2^{22}$ encryptions. Therefore, the data complexity of the attack is $2^{22.25}$ chosen plaintexts, the memory complexity is $2^{20}$ 128-bit blocks, and the time complexity is dominated by encrypting the plaintexts.

It is clear from the algorithm that if the data contains a mixture quadruple for which the ciphertexts $(P_1, P_2)$ have zero difference in $SR(\mathrm{Col}(0))$, then for

the right value of $k_{-1,\{5,10,15\}}$, this quadruple will be found and so the right key will remain. The probability of a wrong key suggestion to remain is extremely low, as in the attack we examine $((2^{22.25})^2/2)^2/2 = 2^{86}$ quartets and the filtering condition is on about 117.4 bits (which consist of 64 bits on the ciphertext side – requiring zero difference in $SR(\text{Col}(0))$ for two ciphertext pairs, and 53.4 bits on the plaintext side – requiring that the values $(x_1, y_1, z_1, w_1)$ constitute a mixture quadruple). So, the probability for a wrong key to pass the filtering is $2^{-31.4}$. As there are $2^{24}$ possible key suggestions, with a high probability no wrong keys remain.

Note that the attack does not recover the value of $k_{-1,0}$ since the question whether a quartet of plaintexts in the structure evolves into a mixture quadruple does not depend on $k_{-1,0}$.

In order to recover the full key, we repeat the attack for each of the four columns on the plaintext side, and apply it once again for the first column, with byte 15 as the 'constant byte' instead of byte 0. This recovers $4 \cdot 24 + 8 = 104$ bits of the key, and the rest of the key can be recovered by exhaustive key search. Therefore, for full key recovery we need data complexity of $5 \cdot 2^{22.25} \approx 2^{24.6}$ chosen plaintexts, memory complexity of $2^{20}$ 128-bit blocks (as the memory can be reused between the attacks), and time complexity of less than $2^{25.5}$ encryptions.

It is clear from the above analysis that the attack succeeds with a high probability, that can be made very close to 100% by increasing the data complexity by a factor of 2. To achieve the exact value, we fully implemented the attack experimentally.

### 4.3 Experimental Verification

We have successfully implemented the 5-round attack. To verify our attack success probability and its dependence on the data complexity, we performed the following experiment. We took four possible amounts of data, $2^{22}, 2^{22.25}, 2^{22.5}$, and $2^{23}$ chosen plaintexts, and for each of them we ran the attack which recovers 3 key bytes for 200 different keys. The results we obtained were the following: For $2^{22}$ plaintexts, the attack succeeded 100 times. For $2^{22.25}$ plaintexts, the attack succeeded 143 times. For $2^{22.5}$ plaintexts, the attack succeeded 187 times, and for $2^{23}$ plaintexts, the attack succeeded in all 200 experiments.

Based on these experiments, we calculated the success probability of full key recovery as $p^5$, where $p$ is the probability of recovering three key bytes (as in order to recover the full key we have to perform 5 essentially independent variants of the attack). Similarly, we calculated the probability when two diagonals in the ciphertext are examined as $1-(1-p)^2$, since the attack fails only if two essentially independent applications of the basic attack fail.

The full details are given in Table 3. As can be seen in the table, with $2^{22.5}$ chosen plaintexts, checking a single diagonal on the ciphertext side is already sufficient for a success rate of over 93% for recovering the first 3 key bytes, and over 70% for recovering the entire key. With $2^{22.25}$ plaintexts, checking a single diagonal in the ciphertext is sufficient for recovering 3 key bytes with success rate of over 70%, but if we want success rate of over 65% for recovering the entire

| Structure size | One diagonal | | Two diagonals | |
|---|---|---|---|---|
| Key material | 3 Bytes | Full key | 3 Bytes | Full key |
| $2^{22}$ | 0.5 | 0.031 | 0.75 | 0.24 |
| $2^{22.25}$ | 0.715 | 0.187 | 0.919 | 0.655 |
| $2^{22.5}$ | 0.935 | 0.715 | 0.996 | 0.979 |
| $2^{23}$ | 1 | 1 | 1 | 1 |

**Table 3.** Success probability of the attack for different data complexities

key, we have to check another diagonal on the ciphertext side, which slightly increases the memory complexity to $2^{21}$ 128-bit blocks.

The experimental results clearly support our analysis presented above. We note that the significant increase of the success rate when the data complexity is increased very moderately follows from the fact that the attack examines quartets, and so multiplying the data by a modest factor of $2^{0.25}$ doubles the number of quartets that can be used in the attack.

## 5 Attacks on 6-Round AES

In this section we present attacks on 6-round AES. We start with a simple extension of Grassi's attack to 6 rounds, and then we present several improvements that allow reducing the attack complexity significantly. Our best attack has data and memory complexities of $2^{27.5}$ and time complexity of $2^{81}$. These results are not very interesting on their own sake, as they are clearly inferior to the improved Square attack on the same variant of AES [15]. However, a further extension of the same attack techniques will allow us obtaining an attack on 7-round AES-192, which clearly outperforms all known attacks on reduced-round AES-192 with practical data and memory complexities (including the improved Square attack).

### 5.1 An Extension of Grassi's Attack to 6 Rounds

Recall that in Grassi's attack on 5-round AES, we take a structure of $2^{32}$ plaintexts that differ only in $SR^{-1}(\text{Col}(0))$, and search for ciphertext pairs that have zero difference in $SR(\text{Col}(0))$. Actually, the 4-round distinguisher underlying the attack guarantees a zero difference only in the state $x''_{4,SR(\text{Col}(0))}$, but as the 5th round is the last one, the MixColumns operation is omitted, and so, the zero difference can be seen in the ciphertext.

When we consider 6-round AES, in order to recover the state $x''_{4,SR(\text{Col}(0))}$ by partial decryption we must guess all 128 key bits. However, we can recover one of these 4 bytes by guessing only four equivalent key bytes. Indeed, if we guess $k_{5,SR(\text{Col}(0))}$ and interchange the order between the MixColumns and AddRoundKey operations of round 4, we can partially decrypt the ciphertexts through round 5 and MixColumns to obtain the value of byte 0 before MixColumns. As

AddRoundKey does not affect differences, this allows us evaluating differences at the state $x''_{4,0}$.

By the distinguisher, the difference in this byte for both pairs in the mixture quadruple is zero. However, this is only an 16-bit filtering. In order to obtain an additional filtering, we recall that by Remark 1, each pair has 7 mixtures. Checking the condition for all of them, we get a 64-bit filtering, that is sufficient for discarding almost all wrong key guesses. The attack is given in Algorithm 4.

**Algorithm 4** Attacking 6-Round AES

Ask for the encryption of $2^{32}$ chosen plaintexts in which $SR^{-1}(\text{Col}(0))$ assumes all $2^{32}$ possible values and the rest of the bytes are constant.
**for** each guess of $k_{-1,SR^{-1}(\text{Col}(0))}$ **do**
    Select arbitrarily $2^{32}$ plaintexts pairs $(P_1^i, \hat{P_1^i})$ from the structure.
    **for** each pair $(x^1_{-1}, y^1_{-1}) = (P_1, \hat{P_1})$ **do**
        Partially encrypt $(x^1_{-1}, \hat{x}^1_{-1})$ through round 0, obtain $(x^1_1, \hat{x}^1_1)$.
        Let the 7 mixtures of $(x^1_1, \hat{x}^1_1)$ be $(x^2_1, \hat{x}^2_1), \ldots, (x^8_1, \hat{x}^8_1)$.
        Partially decrypt the 7 mixtures to obtain the plaintext pairs $(P_2, \hat{P_2}) = (x^2_{-1}, \hat{x}^2_{-1}), \ldots, (P_8, \hat{P_8}) = (x^8_{-1}, \hat{x}^8_{-1})$.
        **for** each value of $k_{5,SR(\text{Col}(0))}$ **do**
            Take all ciphertext pairs $(C_1, \hat{C_1}), \ldots, (C_8, \hat{C_8})$.
            Partially decrypt them through rounds 5,4.
            **if** for all $1 \leq j \leq 8$: $x^{j''}_{4,0} \oplus \hat{x}^{j''}_{4,0} = 0$ (i.e., all 8 pairs have a zero difference in byte 0 before the MixColumns of round 4) **then**
                Store $k_{5,SR(\text{Col}(0))}$ in a list $L_1$.
            **end if**
        **end for**
        **if** $L_1$ is empty **then**
            Discard the pair $(P_1, \hat{P_1})$.
        **else**
            Repeat the same procedure for $k_{5,SR(\text{Col}(1))}$, with respect to the byte $x''_{4,7}$ and $L_2$.
            **if** $L_2$ is not empty **then**
                Repeat the same procedure for $k_{5,SR(\text{Col}(2))}$ and $k_{5,SR(\text{Col}(3))}$, with respect to bytes $x''_{4,10}$ and $x''_{4,15}$, respectively.
                Output the remaining key suggestion.
            **else**
                Discard the pair $(P_1, \hat{P_1})$.
            **end if**
        **end if**
    **end for**  ▷ If no pairs remain, move to the next guess of $k_{-1,SR^{-1}(\text{Col}(0))}$.
**end for**

For the right key guess, it is expected that after $2^{32}$ pairs $(P_1, \hat{P_1})$, we will encounter a good pair (i.e., a pair for which the difference in the state $x''_{4,SR(\text{Col}(0))}$ is zero), and then by the distinguisher, the difference in the same state for all other 7 mixtures is zero as well. Hence, the right key is expected to be

suggested. (Concretely, the probability that the right key is not suggested is $(1-2^{-32})^{2^{32}} \approx e^{-1}$). For wrong key guesses, for each pair $(P_1, \hat{P}_1)$, the probability to pass the filtering of Step 10 is $2^{-64}$, and thus, the probability that there exists a guess of $k_{5,SR(\mathrm{Col}(0))}$ that passes it is $2^{-32}$. Hence, for all values of $(P_1, \hat{P}_1)$ except a few values, the list $L_1$ remains empty and the pair is discarded after Step 15. For the few remaining pairs, the probability that there exists a guess of $k_{5,SR(\mathrm{Col}(1))}$ that passes the filtering of Step 17 is again $2^{-32}$, and so, for all but very few guesses of $k_{-1,\{SR^{-1}(\mathrm{Col}(0))\}}$, all pairs are discarded. The few remaining guesses are easily checked by trial encryption.

The most time consuming part of the attack is Steps 9,10, which are performed for $2^{64}$ key guesses and $2^{32}$ plaintext pairs. (Note that Step 17 is performed for a much smaller number of pairs, and thus is negligible.) Steps 9,10 essentially consist of partial decryption of one column through round 5 for 16 values – which is clearly less than a single 6-round encryption. Therefore, the time complexity of the attack is $2^{64} \cdot 2^{32} = 2^{96}$ encryptions. The memory complexity is $2^{32}$ (dominated by storing the plaintexts), and the success probability is $1 - e^{-1} = 0.63$. The attack recovers the full subkey $k_5$, which yields immediately the full secret key via the key scheduling algorithm.[2]

### 5.2 Improvements of the 6-Round Attack

In this section we present two improvements of the 6-round attack described above, which allow us to reduce its complexity significantly. While the resulting attack is still inferior to some previously known attacks on 6-round AES, we describe the improvements here since they will be used in our attack on 7-round AES, and will be easier to understand in the 'simpler' case of the 6-round variant.

**1. Using the meet-in-the-middle (MITM) approach.** We observe that instead of guessing the subkey $k_{5,\{0,7,10,13\}}$, we can use a MITM procedure. Indeed, the difference in the byte $x''_{4,0}$ (which we want to evaluate in the attack) is a linear combination of the differences in the four bytes $x_{5,0}, x_{5,1}, x_{5,2}, x_{5,3}$. Specifically, by the definition of MixColumns$^{-1}$, we have

$$\Delta(x''_{4,0}) = 0E_x \cdot \Delta(x_{5,0}) \oplus 0B_x \cdot \Delta(x_{5,1}) \oplus 0D_x \cdot \Delta(x_{5,2}) \oplus 09_x \cdot \Delta(x_{5,3}),$$

and thus the equation $\Delta(x''_{4,0}) = 0$ can be written in the form

$$0E_x \cdot \Delta(x_{5,0}) \oplus 0B_x \cdot \Delta(x_{5,1}) = 0D_x \cdot \Delta(x_{5,2}) \oplus 09_x \cdot \Delta(x_{5,3}). \tag{3}$$

Hence, instead of guessing the four key bytes $k_{5,\{0,7,10,13\}}$ which allow us to compute the values of $x_{5,0}, x_{5,1}, x_{5,2}, x_{5,3}$ and to check the 64-bit condition on the differences in $x''_{4,0}$, we can do the following:

1. Guess bytes $k_{5,\{0,7\}}$ and compute $x_{5,0}, x_{5,1}$. Store in a table the contribution of these bytes to Equation (3), i.e., the concatenation of the values $0E_x \cdot \Delta(x^j_{5,0}) \oplus 0B_x \cdot \Delta(x^j_{5,1})$ for $j = 1, \ldots, 8$.

[2] We assume, for sake of simplicity, that the attack is mounted on AES-128. When the attack is applied to AES-192 or AES-256, the rest of the key can be recovered easily by auxiliary techniques.

2. Guess bytes $k_{5,\{10,13\}}$ and compute $x_{5,2}, x_{5,3}$. Compute the contribution of these bytes to Equation (3), i.e., the concatenation of the values $0D_x \cdot \Delta(x^j_{5,2}) \oplus 09_x \cdot \Delta(x^j_{5,3})$ for $j = 1, \ldots, 8$, and search it in the table.
3. For each match in the table, store in $L_1$ the combination $k_{5,\{0,7,10,13\}}$. If there are no matches in the table, discard the pair $(P_1, \hat{P}_1)$.

This meet-in-the-middle procedure is clearly equivalent to guessing $k_{5,\{0,7,10,13\}}$ and checking the condition on $x''_{4,0}$ directly. The time complexity of the procedure, for each pair $(P_1, \hat{P}_1)$, is $2 \cdot 2^{16}$ evaluations of two S-boxes for 16 ciphertexts, and $2 \cdot 2^{16}$ lookups into a table of size $2^{16}$, which are less than $2^{16}$ encryptions.

This procedure can replace Steps 7–10 in the attack presented above, while all other parts of the attack remain unchanged. (Of course, Step 17 can be replaced similarly, but its complexity is anyway negligible.) This reduces the time complexity of the attack to $2^{32} \cdot 2^{32} \cdot 2^{16} = 2^{80}$ encryptions, without affecting the data and memory complexities.

**2. Reducing the data complexity by using less mixtures.** We would like to reduce the data complexity by considering only part of the structure of size $2^{32}$, as we did in Improvement 1 of the 5-round attack. In order to get a significant reduction in the data complexity, we first need to reduce the number of mixtures used in the attack.

We observe that we may use 3 mixtures of each pair instead of all possible 7 mixtures. As a result, the filtering in Step 10 above is reduced to 32 bits, and thus, for each pair $(P_1, \hat{P}_1)$, about one value of $k_{5,\{0,7,10,13\}}$ is inserted into the list $L_1$. Similarly, in Steps 17,19 it is expected that a few suggestions of the entire key $k_5$ remain *for each pair* $(P_1, \hat{P}_1)$. These suggestions are easily checked by trial encryption.

How does this modification affect the time complexity? On the one hand, Steps 9,10 are now repeated four times for each pair $(P_1, \hat{P}_1)$. On the other hand, each application of this step becomes twice faster since the partial decryptions are performed for 8 ciphertexts instead of 16. Hence, the overall time complexity becomes $4 \cdot \frac{1}{2} \cdot 2^{80} = 2^{81}$ encryptions. The memory complexity remains unchanged, and so is the success probability.

The reduction of the number of mixtures allows us reducing the data complexity effectively, similarly to the reduction we did in Improvement 1 of the 5-round attack. Note that the entire structure of $2^{32}$ plaintext contains $2^{63}$ pairs, and about $2^{31}$ of them are good and so can be used in the attack. Hence, if we take a random subset $S$ of the structure of size $\alpha 2^{32}$, the probability that one of these $2^{31}$ good pairs, along with at least three of its 7 mixtures, is included in $S$, is approximately $2^{31}\alpha^8 \cdot \binom{7}{3} \approx 2^{36}\alpha^8$. Hence, if we take $\alpha = 2^{-4.5}$, with a good probability the plaintext set $S$ contains a pair that can be used in the attack.

Formally, the changes required in the attack algorithm are only in Steps 1–6, and are the following:

1. Take a structure of $2^{27.5}$ chosen plaintexts with the same value in all bytes but those of $SR^{-1}(\mathrm{Col}(0))$.

2. For each guess of subkey $k_{-1,SR^{-1}(\mathrm{Col}(0))}$, go over all pairs of plaintexts $(P_1, \hat{P}_1)$ in $S$, and for each of them do the following:
   (a) Partially encrypt $(P_1, \hat{P}_1) = (x^1_{-1}, \hat{x}^1_{-1})$ through AddRoundKey and round 0 to obtain $(x^1_1, \hat{x}^1_1)$. Consider all 7 mixtures of $(x^1_1, \hat{x}^1_1)$ (denoted by $(x^2_1, \hat{x}^2_1), \ldots, (x^8_1, \hat{x}^8_1)$), and partially decrypt them to find the corresponding plaintext pairs $(P_2, \hat{P}_2) = (x^2_{-1}, \hat{x}^2_{-1}), \ldots, (P_8, \hat{P}_8) = (x^8_{-1}, \hat{x}^8_{-1})$. Check whether for at least three of them, both plaintexts are included in $S$. If yes, continue as in the original attack. If no, discard the pair $(P_1, \hat{P}_1)$.

The complexity of checking all pairs $(P_1, \hat{P}_1)$ is less than $2^{54}$ encryptions, which is negligible with respect to other steps of the attack. Since the expected number of pairs that are not discarded instantly is $2^{32}$, the attack complexity is the same as the original attack – $2^{81}$ encryptions. The success probability is $1-(1-2^{-32})^{2^{32}} \approx 1 - e^{-1} = 0.63$, as in the original attack.

To summarize, the data and memory complexity of the improved attack is $2^{27.5}$, and its time complexity is $2^{81}$ encryptions. Both improvements will be used in the 7-round attacks presented in the next section.

## 6 Attacks on 7-Round AES-192 and AES-256

In this section we present our new attacks on 7-round AES. First we present the attack on AES-256, which extends the 6-round attack by another round using a MITM technique, and then uses *dissection* [14] to reduce the memory complexity of the attack. Then we show how in the case of AES-192, the key schedule can be used (in conjunction with a more complex dissection attack) to further reduce the data and time complexities of the attack. Our best attack on AES-192 recovers the full key with data complexity of $2^{30}$, memory complexity of $2^{32}$, and time complexity of $2^{153}$, which is better than all previously known attacks on reduced-round AES-192 with practical data and memory complexities.

### 6.1 Basic Attack on AES-192 and AES-256

The basic attack is a further extension by one round of the 6-round attack. Recall that in the 6-round attack we guess the subkey bytes $k_{5,\{0,7,10,13\}}$ and check whether the state bytes $x_{5,\{0,1,2,3\}}$ satisfy a linear condition (Equation 3). When we consider 7-round AES, in order to check this condition we have to guess *the entire subkey* $k_6$ and bytes $0, 7, 10, 13$ of the equivalent subkey $u_5$. Of course, this leads to an extremely high time complexity. In addition, the filtering condition – which is on only 64 bits (i.e., 8 pairs with zero difference in a single byte) – is far from being sufficient for discarding such a huge amount of key material.

In order to improve the filtering condition, we attack *two columns simultaneously.* That is, we guess the entire subkey $k_6$ and bytes $0, 7, 10, 13$ and $1, 4, 8, 14$

of $u_5$ and check linear conditions on both state bytes $x_{5,\{0,1,2,3\}}$ (which is Equation 3) and state bytes $x_{5,\{4,5,6,7\}}$ (which is the following equation:

$$0D_x \cdot \Delta(x_{5,5}) \oplus 09_x \cdot \Delta(x_{5,6}) = 0B_x \cdot \Delta(x_{5,4}) \oplus 0E_x \cdot \Delta(x_{5,7}), \tag{4}$$

that corresponds to the condition $\Delta(x''_{4,7}) = 0$). Thus, we have 4 more key bytes to guess, but the filtering is increased to 128 bits.

In order to reduce the time complexity, we extend the MITM procedure described in Section 5.2 to cover round 6 as well. Specifically, we modify the MITM procedure described in Improvement 1 of Section 5.2 as follows:

1. Guess bytes $k_{6,SR(\mathrm{Col}(0,3))}$ and $u_{5,\{0,1,13,14\}}$, and compute $x_{5,0}, x_{5,1}$ and $x_{5,5}, x_{5,6}$. Store in a table the contribution of bytes $x_{5,0}, x_{5,1}$ to Equation (3) (i.e., the concatenation of the values $0E_x \cdot \Delta(x^j_{5,0}) \oplus 0B_x \cdot \Delta(x^j_{5,1})$ for $j = 1, \ldots, 8$), and the contribution of bytes $x_{5,5}, x_{5,6}$ to Equation (4) (i.e., the concatenation of the values $0D_x \cdot \Delta(x^j_{5,5}) \oplus 09_x \cdot \Delta(x^j_{5,6})$ for $j = 1, \ldots, 8$) – 128 bits in total.
2. Guess bytes $k_{6,SR(\mathrm{Col}(1,2))}$ and $u_{5,\{4,7,8,10\}}$, and compute $x_{5,2}, x_{5,3}$ and $x_{5,4}, x_{5,7}$. Compute the contribution of bytes $x_{5,2}, x_{5,3}$ to Equation (3) and the contribution of bytes $x_{5,4}, x_{5,7}$ to Equation (4), and search it in the table.
3. For each match in the table, store in $L_1$ the combination $k_6, u_{5,\{0,1,4,7,8,10,13,14\}}$.

After the MITM procedure, for each guess of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$ and for each pair $(P_1, \hat{P}_1)$, we remain with $2^{192} \cdot 2^{-128} = 2^{64}$ key suggestions. To discard the wrong ones, we repeat the attack for Col(2) of $x_5$ (where now the only key bytes we need to guess are $u_{5,\{2,5,8,15\}}$ and we can again use MITM), and for Col(3) of $x_5$ (where the only key bytes we need to guess are $u_{5,\{3,6,9,12\}}$ and we can again use MITM). In total, we have a 256-bit filtering, and so we obtain on average $2^{256} \cdot 2^{-256} = 1$ suggestions for the entire subkeys $u_5, k_6$, which of course yield the secret key. The remaining suggestions can be checked easily by trial encryption.

What is the time complexity of the attack? The most time consuming operation is the first MITM procedure which is performed for $2^{32}$ guesses of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$ and for $2^{32}$ pairs $(P_1, \hat{P}_1)$, and consists of $2^{96}$ times decrypting a full AES round and one column of another round, for 16 ciphertexts, plus $2 \cdot 2^{96}$ table lookups. Estimating a table lookup as one full AES round (following common practice), the total time complexity is $2^{32} \cdot 2^{32} \cdot 2^{96} \cdot 3 = 2^{161.6}$ encryptions.

The memory complexity is $2^{96}$, required for the MITM procedure. The data complexity of the attack is $2^{32}$ chosen plaintexts. However, it can be reduced using Improvement 2 described in Section 5.2. Instead of taking the full structure of $2^{32}$ plaintexts, we can take an arbitrary subset of $2^{30}$ plaintexts. As discussed above, there are $2^{31}$ good pairs that can be used in the attack, and given a pair, the probability that it belongs to $S$ along with all its 7 mixtures is approximately $2^{31} \cdot (2^{-2})^{16} = 1/2$. In addition, if the attack fails for all pairs, we can repeat the attack with other shifted columns of $x''_4$ instead of $x''_{4,SR(0)}$. As described in Improvement 5 in Section 4.1, this increases the number of good pairs by a factor of 4 – which means that on average, 2 pairs will be included in $S$ along with all

their 7 mixtures. Therefore, starting with $2^{30}$ plaintexts, the success probability of the attack is still above $1 - e^{-1} = 0.63$.

We do not present the attack algorithm here, as it will be subsumed by the improved attack algorithm we present in the next subsection.

### 6.2 Improved Attack on AES-192 and AES-256 Using Dissection

In this section we show that the memory complexity of the attack described above can be reduced from $2^{96}$ to $2^{48}$ without affecting the data and time complexities, using the *dissection* technique [14].

For ease of exposition, we first briefly recall the generic dissection attack on 4-encryption (denoted in [14] $Dissect_2(4,1)$) and then present its application in our case.

The algorithm $Dissect_2(4,1)$ is given four plaintext/ciphertext pairs $(P_1, C_1)$, $\ldots, (P_4, C_4)$ to a 4-round cipher. It is assumed that the block length is $n$ bits, and that in each round $i$ (for $i = 0, 1, 2, 3$) there is an independent $n$-bit key $k_i$. The algorithm finds all values of $(k_0, k_1, k_2, k_3)$ that comply with the 4 plaintext/ciphertext pairs (the expected number of keys is, of course, one), in time $O(2^{2n})$ and memory $O(2^n)$. Instead of using the notations of [14], we will be consistent with our notations, and denote the plaintexts by $(x_0, y_0, z_0, w_0)$ and the intermediate values before round $i$ by $(x_i, y_i, z_i, w_i)$.

The dissection algorithm is the following:

1. Given plaintexts $(x_0, y_0, z_0, w_0) = (P_1, P_2, P_3, P_4)$ and their corresponding ciphertexts $(x_4, y_4, z_4, w_4) = (C_1, C_2, C_3, C_4)$, for each candidate value of $x_2$:
2. (a) Run a standard MITM attack on 2-round encryption with $(x_0, x_2)$ as a single plaintext/ciphertext pair, to find all keys $(k_0, k_1)$ which 'encrypt' $x_0$ to $x_2$. For each of these $2^n$ values, partially encrypt $y_0 = P_2$ using $(k_0, k_1)$, and store in a table the corresponding values of $y_2$, along with the values of $(k_0, k_1)$.
   (b) Run a standard MITM attack on 2-round encryption with $(x_2, x_4)$ as a single plaintext/ciphertext pair, to find all keys $(k_2, k_3)$ which 'encrypt' $x_2$ to $x_4$. For each of these $2^n$ values, partially decrypt $C_2$ using $(k_2, k_3)$ and check whether the suggested value for $y_2$ appears in the table. If so, check whether the key $(k_0, k_1, k_2, k_3)$ suggested by the table and the current $(k_2, k_3)$ candidate encrypts $P_3$ and $P_4$ into $C_3$ and $C_4$, respectively.

We call the two 2-round MITM procedures *internal* ones, and the final MITM step *external*.

The time complexity of each of the two internal 2-round MITM attacks is about $2^n$, and so is the time complexity of the external MITM procedure. As these procedures are performed for each value of $x_2$, the time complexity of the attack is $O(2^{2n})$ operations. The memory complexity is $O(2^n)$, required for each of the MITM procedures. Note that the time complexity of the attack is not better than the complexity of a simple MITM attack on a 4-round cipher with independent round keys. The advantage of dissection is the significant reduction in memory complexity – from $2^{2n}$ to $O(2^n)$.

While this may not be clear at a first glance, a standard MITM attack can be transformed into a $Dissect_2(4,1)$ attack whenever each of the two parts of the MITM procedure can be further subdivided into two parts whose contributions are independent, given that a 'partial guess in the middle' (like the guess of $x_2$ above) can be performed. This is the case in our attack.

Note that the contribution of the first part of the MITM procedure described above to each of Equations (3),(4) can be represented as the XOR of two independent contributions: the contribution of state bytes $x_{5,0}$ and $x_{5,5}$, which can be computed by guessing $k_{6,SR(\mathrm{Col}(0))}$ and $u_{5,\{0,1\}}$, and the contribution of state bytes $x_{5,1}$ and $x_{5,6}$ which can be computed by guessing $k_{6,SR(\mathrm{Col}(3))}$ and $u_{5,\{13,14\}}$. The second half can be divided similarly. The contribution of each side to Equations (3),(4) plays the role of the guessed intermediate value. Hence, we introduce the following auxiliary notations. For $1 \leq j \leq 8$, let

$$a_j = 0E_x \cdot \Delta(x^j_{5,0}) \oplus 0B_x \cdot \Delta(x^j_{5,1}) = 0D_x \cdot \Delta(x^j_{5,2}) \oplus 09_x \cdot \Delta(x^j_{5,3}) \qquad (5)$$

denote the contributions of the two sides to Equation (3) for ciphertext pair $(C_j, \hat{C}_j)$, and let

$$b_j = 0D_x \cdot \Delta(x^j_{5,5}) \oplus 09_x \cdot \Delta(x^j_{5,6}) = 0B_x \cdot \Delta(x_{5,4}) \oplus 0E_x \cdot \Delta(x_{5,7}) \qquad (6)$$

denote the contributions of the two sides to Equation (4) for ciphertext pair $(C_j, \hat{C}_j)$.

This allows us to mount the following attack:

1. **Constructing the plaintext pool.** Take a structure $S$ of $2^{30}$ chosen plaintexts with the same value in all bytes but those of $SR^{-1}(\mathrm{Col}(0))$.
2. For each guess of subkey $k_{-1,SR^{-1}(\mathrm{Col}(0))}$, go over all chosen pairs of plaintexts $(P_1, \hat{P}_1)$ in $S$, and for each of them do the following:
   (a) **Checking that the pair can be used in the attack, i.e., the pair and all its 7 mixtures are in the plaintext pool.** Partially encrypt $(P_1, \hat{P}_1) = (x^1_{-1}, \hat{x}^1_{-1})$ through AddRoundKey and round 0 to obtain $(x^1_1, \hat{x}^1_1)$. Consider all 7 mixtures of $(x^1_1, \hat{x}^1_1)$, which we denote $(x^2_1, \hat{x}^2_1), \ldots, (x^8_1, \hat{x}^8_1)$, and partially decrypt them to find the corresponding plaintext pairs $(P_2, \hat{P}_2) = (x^2_{-1}, \hat{x}^2_{-1}), \ldots, (P_8, \hat{P}_8) = (x^8_{-1}, \hat{x}^8_{-1})$. Check whether for all of them, both plaintexts are included in $S$. If no, discard the pair $(P_1, \hat{P}_1)$.
   (b) For each candidate value of $(a_1, a_2, a_3, a_4, a_5, a_6)$ do the following:
   (c) **First internal MITM procedure:**
      i. Guess bytes $k_{6,SR(\mathrm{Col}(0))}$ and $u_{5,0}$, and compute $x_{5,0}$. Store in a table the contribution of the byte $x_{5,0}$ to Equation (3) for the pairs $(C_1, \hat{C}_1), \ldots, (C_6, \hat{C}_6)$, i.e., the concatenation of the values $0E_x \cdot \Delta(x^j_{5,0}) \oplus a_j$ for $j = 1, \ldots, 6$ – 48 bits in total.
      ii. Guess bytes $k_{6,SR(\mathrm{Col}(3))}$ and $u_{5,13}$, and compute $x_{5,1}$. Compute the contribution of the byte $x_{5,1}$ to Equation (3) for the pairs $(C_1, \hat{C}_1), \ldots,$ $(C_6, \hat{C}_6)$, i.e., the concatenation of the values $0B_x \cdot \Delta(x^j_{5,1})$ for $j = 1, \ldots, 6$, and check it in the table.

iii. For each value found in the table, use the suggested value of $k_{6,SR(\mathrm{Col}(0,3))}$ and $u_{5,\{0,13\}}$, guess bytes $u_{5,\{1,14\}}$, and partially decrypt the ciphertexts to obtain the values $(a_7, a_8, b_1, b_2, \ldots, b_8)$. Store them in a table, together with the suggestion for $k_{6,SR(\mathrm{Col}(0,3))}$ and $u_{5,\{0,1,13,14\}}$.

(d) **Second internal MITM procedure:**

i. Guess bytes $k_{6,SR(\mathrm{Col}(1))}$ and $u_{5,7}$, and compute $x_{5,3}$. Store in a table the contribution of the byte $x_{5,3}$ to Equation (3) for the pairs $(C_1, \hat{C}_1), \ldots, (C_6, \hat{C}_6)$, i.e., the concatenation of the values $09_x \cdot \Delta(x^j_{5,3}) \oplus a_j$ for $j = 1, \ldots, 6$ – 48 bits in total.

ii. Guess bytes $k_{6,SR(\mathrm{Col}(2))}$ and $u_{5,10}$, and compute $x_{5,2}$. Compute the contribution of the byte $x_{5,2}$ to Equation (3) for the pairs $(C_1, \hat{C}_1), \ldots, (C_6, \hat{C}_6)$, i.e., the concatenation of the values $0D_x \cdot \Delta(x^j_{5,1})$ for $j = 1, \ldots, 6$, and check it in the table.

iii. For each value found in the table, use the suggested value of $k_{6,SR(\mathrm{Col}(1,2))}$ and $u_{5,\{7,10\}}$, guess bytes $u_{5,\{4,8\}}$, and partially decrypt the ciphertexts to obtain the values $(a_7, a_8, b_1, b_2, \ldots, b_8)$. Check whether the vector exists in the table. If yes, store in a table $L$ the suggested value of $k_5$ and $u_{5,\{0,1,4,7,8,10,13,14\}}$.

(e) **Completing the attack:** For each remaining key suggestion, repeat the attack for the two last shifted columns of $u_5$, with respect to the state bytes $x''_{4,10}$ and $x''_{4,13}$, to filter wrong key guesses and obtain suggestions for the entire $k_6$ and $u_5$. For each remaining suggestion, use $k_6, u_5$ to retrieve the full key and check it by trial encryption.

The memory complexity of the attack is $2^{48}$ 80-bit values (required in Step 2(c)), which are less than $2^{48}$ 128-bit blocks. As for the time complexity, for each guess of $k_{-1,\{0,5,10,15\}}$, each pair $(P_1, \hat{P}_1)$, and each guessed 48-bit value $(a_1, \ldots, a_6)$, the internal MITM procedures take $2^{40}$ time and the external MITM procedure consists of $2^{48}$ times decrypting a full AES round and one column of another round, for 16 ciphertexts, plus $2 \cdot 2^{96}$ table lookups. Estimating a table lookup as one full AES round, the total time complexity of this step is $2^{32} \cdot 2^{32} \cdot 2^{48} \cdot 2^{48} \cdot 3 = 2^{161.6}$ encryptions. The complexity of all other steps is negligible.

Therefore, the data complexity of the attack is $2^{30}$ chosen plaintexts, the memory complexity is $2^{48}$ 128-bit blocks, and the time complexity is $2^{161.6}$ encryptions. The success probability is $1 - e^{-1} = 0.63$.

### 6.3 Improved Attacks on AES-192 Exploiting the Key Schedule

While in the attack on AES-256 the subkeys we guess in the last two rounds are independent, in the case of AES-192 there exists a strong relation between $u_5$ and $k_6$. Specifically, by the AES key schedule we have

$$k_{5,\mathrm{Col}(1)} = k_{6,\mathrm{Col}(2)} \oplus k_{6,\mathrm{Col}(3)}, \tag{7}$$

and

$$k_{5,\mathrm{Col}(0)} = k_{6,\mathrm{Col}(2)} \oplus SB(k_{6,\mathrm{Col}(1)} \lll 8) \oplus RCON[5]. \tag{8}$$

Since $u_{5,\mathrm{Col}(j)} = MC^{-1}(k_{5,\mathrm{Col}(j)})$ for each $j$, two columns of $u_5$ can be expressed as combinations of bytes of $k_6$. As these two columns contain half of the bytes of $u_5$ guessed in the attack, we will be able to use them to enhance the filtering condition. Specifically, this enables us to attack a single column in $x_5$ (and so guess only bytes $u_{5,SR(\mathrm{Col}(0))}$ and the entire $k_6$, a total of 160 key bits), and use Equations (7) and (8) as additional filtering conditions in the MITM procedure, thus increasing the filtering to 80 bits. This allows us to reduce the time complexity of the attack to $2^{152}$ and the memory complexity of the attack to $2^{40}$, without affecting the data complexity.

By using a much more complex variant of the dissection attack, we can further reduce the memory complexity to $2^{32}$ without affecting the data and time complexities, thus obtaining an attack which recovers the full secret key in $2^{30}$ data, $2^{32}$ memory, and $2^{153}$ time, which outperforms the classical Square attack in all three complexity parameters. The full deatils appear in Appendix A.

## 7 An Alternative Improvement for the 6-Round and 7-Round Attacks

As we mentioned in several places, an obvious point in which Grassi's attack can be enhanced is deploying the fact that while the structure of size $2^{32}$ contains $2^{31}$ good pairs, we need only one good pair (along with its mixtures) to apply the attack. So far, we exploited the abundance of good pairs to reduce the data complexity – we took a smaller structure of plaintexts, which was sufficiently large so that at least one good pair, along with the required mixtures, is included in our structure.

In this section we suggest an alternative way to exploit the abundance of good pairs – ask that the good pair we use in the attack will satisfy some additional property, which will allow reducing the *time complexity* of the attack. We first demonstrate the improvement on the 6-round attack, and then we apply it (or more precisely, a variant of it) to the 7-round attack on AES-192.

**An alternative improvement to the 6-round attack.** Recall that in the 6-round attack, we guess bytes $k_{-1,SR^{-1}(\mathrm{Col}(0))}$, go over $2^{32}$ plaintext pairs $(P_1, \hat{P}_1)$, and perform a MITM attack on 4 bytes of the subkey $k_5$. Now, instead of taking any ciphertext pair which corresponds to a plaintext pair $(P_1, \hat{P}_1)$, we add a restriction on the ciphertext pair, that can be checked easily. Specifically, we require that in the ciphertext pair $(C_1, \hat{C}_1)$, there is a zero difference in the entire shifted column $SR(\mathrm{Col}(0))$. Among the $2^{63}$ ciphertext pairs, about $2^{31}$ satisfy this extra condition. But importantly, out of the $2^{31}$ good pairs, about $2^7$ satisfy this condition, since in the good pairs, we already know that $\Delta(x''_{4,0}) = 0$, and so 8 bits out of the 32 bits of the extra condition are satisfied for sure. Thus, among the pairs that satisfy the extra condition, the probability of a pair to be good is enhanced from $2^{-32}$ to $2^{-24}$.

This implies that instead of checking $2^{32}$ pairs $(P_1, \hat{P}_1)$ as we do in the basic attack, it is sufficient to check $2^{24}$ pairs that satisfy the extra condition. We can thus modify Steps 1–3 of the attack as follows:

1. Consider a structure of $2^{32}$ chosen plaintexts in which $SR^{-1}(\text{Col}(0))$ assume all $2^{32}$ possible values and the rest of the bytes are constant. Insert the corresponding ciphertexts into a hash table indexed by bytes $SR(\text{Col}(0))$ of the ciphertext, and extract all plaintext pairs $(P_1, \hat{P}_1)$ for which the corresponding ciphertexts have difference zero in $SR(\text{Col}(0))$.

Then the attack is applied without change, with the advantage that it is sufficient to apply Step 2 for $2^{24}$ pairs instead of $2^{32}$. This reduces the time complexity by a factor of $2^8$, without affecting the other parameters of the attack.

While this improvement cannot be completely combined with the data complexity reduction described in Improvement 2 of Section 5 (as once we case an additional restriction, the number of good pairs we can use is reduced significantly), the data complexity can still be slightly reduced. Note that after the initial filtering of Step 1, the data still contains $2^7$ good pairs, while we need only a single pair. By the same argument as in Improvement 2, if we take a subset $S$ of size $\alpha 2^{32}$, the probability that one of these $2^7$ remaining good pairs, along with at least three of its 7 mixtures (that do not need to satisfy the basic filtering condition!), is included in $S$, is approximately $2^7\alpha^8 \cdot \binom{7}{3} \approx 2^{12}\alpha^8$. Hence, if we take $\alpha = 2^{-1.5}$, with a good probability the plaintext set $S$ contains a good pair that can be used in the attack.

Therefore, overall we obtain an attack with data and memory complexity of $2^{30.5}$, and time complexity of $2^{73}$ encryptions.

**An alternative improvement to the attack on 7-round AES-192.** Recall that in the first step of the attack, we guess bytes $k_{-1,SR^{-1}(\text{Col}(0))}$, go over $2^{32}$ plaintext pairs $(P_1, \hat{P}_1)$, and perform a MITM attack on 4 bytes of the subkey $u_5$ and the entire subkey $k_6$ (a total of 160 subkey bits), with an 80-bit filtering. After that step, we are left with $2^{144}$ key suggestions and have to find a source for additional filtering. We obtain this filtering by examining $x_{5,\text{Col}(1)}$ and using the condition $\Delta(x''_{4,7}) = 0$. Since we already know the subkey bytes $u_{5,\{1,4\}}$, we can guess bytes $u_{5,\{11,14\}}$, partially decrypt the ciphertexts to find the values $x_{5,\text{Col}(1)}$, and obtain a 64-bit filtering by checking the condition on the state $\Delta(x''_{4,7})$, for all 8 pairs. Naively, this increases the time complexity to $2^{160}$. In Section A we suggested either to perform a MITM procedure on these two key bytes, or to retrieve them instantly using a large precomputed table. The former suggestion increases the time complexity to $2^{153}$, while the latter increases the memory complexity to $2^{144}$. We show how to obtain the additional filtering without increasing neither the time nor the memory complexity.

As in the 6-round attack, we add a requirement on the good pairs. Specifically, we require that in the ciphertext pair $(C_1, \hat{C}_1)$, there is a zero difference in the entire shifted column $SR(\text{Col}(2))$. As a result, we know that in the intermediate values that correspond to $(C_1, \hat{C}_1)$, we have $\Delta(x_{5,7}) = 0$. In addition, as we know $u_{5,\{1,4\}}$, we can compute $\Delta(x_{5,\{4,5\}})$ for the same pair. Furthermore, assuming that $(C_1, \hat{C}_1)$ is a good pair, we also know that its intermediate values satisfy $\Delta(x''_{4,7}) = 0$. Now, consider the MixColumns operation in round 4, Column 1 in the encryption process of $(C_1, \hat{C}_1)$. We know the difference in three bytes after MixColumns and in one byte before MixColumns. By the structure

of MixColumns, this allows to retrieve the input and output differences in all other bytes, by simply solving a system of linear equations. In particular, we retrieve $\Delta(x_{5,6})$. On the other hand, we can obtain the difference $\Delta(x'_{5,6})$ by partial decryption. This gives us the input and output differences to the SubBytes operation in round 5, byte 6, which allows us to retrieve the actual values in the state $x'_{5,6}$ by a single lookup into a precomputed table of size $2^{16}$. Finally, from the value $x'_{5,6}$ we can recover $u_{5,14}$ by partial decryption, and then we can repeat the above procedure with one of the other pairs $(C_j, \hat{C}_j)$ to retrieve $u_{5,11}$, using the fact that we can compute the difference $\Delta(x_{5,\{4,5,6\}})$ with the subkey material we already know.

As a result, we obtain the subkey bytes $u_{5,\{11,14\}}$ and can apply the additional filtering, without increasing neither the time nor the memory complexity. To summarize, the data complexity of the attack is $2^{32}$ (note that we cannot reduce the data complexity in this attack, since only very few good pairs satisfy our additional restriction on $(C_1, \hat{C}_1)$ and so we must keep all of them available), the memory complexity is $2^{40}$ and the time complexity is $2^{145.6}$ (where both the memory and the time complexities are dominated by the first step of the MITM procedure).

## 8 Summary

In this paper we developed and experimentally verified the best known key recovery attack on 5-round AES, reducing its total complexity from $2^{32}$ to $2^{22.5}$. We then extended the attack to 7-round AES, obtaining the best key recovery attacks on the 192 and 256 bit versions of this cryptosystem which have practical data and memory complexities. The main problems left open by our results is whether it is possible to extend our new attacks to larger versions of AES, and whether it is possible to use our results to attack other primitives which use reduced-round AES (e.g., 5-round AES) as a component.

## Acknowledgements

The research of Achiya Bar-On and of Nathan Keller was supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. The research of Orr Dunkelman was supported by the Israel Ministry of Science and Technology.

## References

1. Biham, E., Keller, N.: Cryptanalysis of Reduced Variants of Rijndael (1999), unpublished manuscript.

2. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7073, pp. 344–371. Springer (2011)
3. Bossuet, L., Datta, N., Mancillas-López, C., Nandi, M.: ELmD: A Pipelineable Authenticated Encryption and Its Hardware Implementation. IEEE Trans. Computers 65(11), 3318–3331 (2016)
4. Bouillaguet, C., Derbez, P., Dunkelman, O., Fouque, P., Keller, N., Rijmen, V.: Low-Data Complexity Attacks on AES. IEEE Trans. Information Theory 58(11), 7002–7017 (2012)
5. Bouillaguet, C., Derbez, P., Fouque, P.: Automatic Search of Attacks on Round-Reduced AES and Applications. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6841, pp. 169–187. Springer (2011)
6. Boura, C., Lallemand, V., Naya-Plasencia, M., Suder, V.: Making the Impossible Possible. J. Cryptology 31(1), 101–133 (2018), https://doi.org/10.1007/s00145-016-9251-7
7. Cho, J., Choi, K.Y., Dinur, I., Dunkelman, O., Keller, N., Moon, D., Veidberg, A.: WEM: A New Family of White-Box Block Ciphers Based on the Even-Mansour Construction. In: Handschuh, H. (ed.) Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10159, pp. 293–308. Springer (2017)
8. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: Biham, E. (ed.) Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer (1997)
9. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)
10. Demirci, H., Selçuk, A.A.: A Meet-in-the-Middle Attack on 8-Round AES. In: Nyberg, K. (ed.) Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5086, pp. 116–126. Springer (2008), https://doi.org/10.1007/978-3-540-71039-4
11. Derbez, P.: Meet-in-the-middle attacks on AES. Ph.D. thesis, Ecole Normale Suprieure de Paris — ENS Paris (2013)
12. Derbez, P., Fouque, P.: Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES. In: Moriai, S. (ed.) Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8424, pp. 541–560. Springer (2013)
13. Derbez, P., Fouque, P., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 371–387. Springer (2013), https://doi.org/10.1007/978-3-642-38348-9_23
14. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial

Search Problems. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 719–740. Springer (2012)
15. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D.A., Whiting, D.: Improved Cryptanalysis of Rijndael. In: Schneier, B. (ed.) Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1978, pp. 213–230. Springer (2000)
16. Fouque, P., Karpman, P., Kirchner, P., Minaud, B.: Efficient and Provable White-Box Primitives. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 159–188 (2016)
17. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.: Block Ciphers That Are Easier to Mask: How Far Can We Go? In: Bertoni, G., Coron, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8086, pp. 383–399. Springer (2013)
18. Gilbert, H., Minier, M.: A collision attack on 7 rounds of rijndael. In: AES Candidate Conference. pp. 230–241 (2000)
19. Grassi, L.: Mixture Differential Cryptanalysis: New Approaches for Distinguishers and Attacks on round-reduced AES. Cryptology ePrint Archive, Report 2017/832 (2017), `https://eprint.iacr.org/2017/832`
20. Grassi, L., Rechberger, C., Rønjom, S.: A New Structural-Differential Property of 5-Round AES. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10211, pp. 289–317 (2017)
21. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6917, pp. 326–341. Springer (2011)
22. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust Authenticated-Encryption AEZ and the Problem That It Solves. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 15–44. Springer (2015)
23. Mala, H., Dakhilalian, M., Rijmen, V., Modarres-Hashemi, M.: Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In: Gong, G., Gupta, K.C. (eds.) Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6498, pp. 282–291. Springer (2010)
24. Rønjom, S., Bardeh, N.G., Helleseth, T.: Yoyo Tricks with AES. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Secu-

rity, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 217–243. Springer (2017)
25. Tiessen, T.: Polytopic Cryptanalysis. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9665, pp. 214–239. Springer (2016)
26. Tunstall, M.: Improved "Partial Sums"-based Square Attack on AES. In: Samarati, P., Lou, W., Zhou, J. (eds.) SECRYPT 2012 - Proceedings of the International Conference on Security and Cryptography, Rome, Italy, 24-27 July, 2012, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications. pp. 25–34. SciTePress (2012)

# A Improved Attack on AES-192 Exploiting the Key Schedule

As in the attack on 7-round AES-256 presented in Section 6 we guess a significant amount of subkey material, it is reasonable to ask whether the attack can be enhanced by exploiting relations between the guessed subkey bits that hold due to the AES key schedule. In the case of AES-256, the subkeys $u_5$ and $k_6$ are completely independent, and so the only relation that can theoretically be exploited is between the guessed bytes of $u_5, k_6$ and the guessed bytes of $k_{-1}$. Due to the large distance between the subkeys, we weren't able to find such exploitable relations.

The situation for AES-192 is different, as there exists a strong relation between $u_5$ and $k_6$. Specifically, by the AES key schedule we have

$$k_{5,\mathrm{Col}(1)} = k_{6,\mathrm{Col}(2)} \oplus k_{6,\mathrm{Col}(3)},$$

and

$$k_{5,\mathrm{Col}(0)} = k_{6,\mathrm{Col}(2)} \oplus SB(k_{6,\mathrm{Col}(1)} \lll 8) \oplus RCON[5].$$

Since $u_{5,\mathrm{Col}(j)} = MC^{-1}(k_{5,\mathrm{Col}(j)})$ for each $j$, two columns of $u_5$ can be expressed as a combination of bytes of $k_6$. As these two columns contain half of the bytes of $u_5$ guessed in the attack, we will be able to use them to enhance the filtering condition.

Recall that in the 6-round attack, we have a 64-bit filtering condition, while only 4 subkey bytes ($k_{5,SR(\mathrm{Col}(0))}$) are guessed. In the 7-round attack described in the previous subsections we had to guess 128 more key bytes (the entire $k_6$ in addition to 4 bytes of $u_5$), and thus, we attacked two columns simultaneously in order to enhance the filtering condition. As a result, we had to guess four more bytes in $u_5$. Now, we will be able to attack a single column (and so guess only bytes $u_{5,SR(\mathrm{Col}(0))}$ and the entire $k_6$, a total of 160 key bits), and will use the key schedule to enhance the filtering to 80 bits. Specifically, we use the following

equations:

$$\begin{aligned} u_{5,0} &= MC^{-1}(k_{5,\mathrm{Col}(0)})_0 = 0E_x \cdot k_{5,0} \oplus 0B_x \cdot k_{5,1} \oplus 0D_x \cdot k_{5,2} \oplus 09_x \cdot k_{5,3} \\ &= 0E_x \cdot (k_{6,8} \oplus SB(k_{6,7}) \oplus 20_x) \oplus 0B_x \cdot (k_{6,9} \oplus SB(k_{6,4})) \oplus \\ &\oplus 0D_x \cdot (k_{6,10} \oplus SB(k_{6,5})) \oplus 09_x \cdot (k_{6,11} \oplus SB(k_{6,6})), \end{aligned}$$

and

$$\begin{aligned} u_{5,7} &= MC^{-1}(k_{5,\mathrm{Col}(1)})_3 = 0B_x \cdot k_{5,4} \oplus 0D_x \cdot k_{5,5} \oplus 09_x \cdot k_{5,6} \oplus 0E_x \cdot k_{5,7} \\ &= 0B_x \cdot (k_{6,8} \oplus k_{6,12}) \oplus 0D_x \cdot (k_{6,9} \oplus k_{6,13}) \oplus \\ &\oplus 09_x \cdot (k_{6,10} \oplus k_{6,14}) \oplus 0E_x \cdot (k_{6,11} \oplus k_{6,15}). \end{aligned}$$

As a preparation for modifying the MITM procedure, we rewrite the equations such that each side will consist of bytes guessed in one side of the MITM, and denote the equal parts by $c_1$ and $c_2$. We get:

$$\begin{aligned} c_1 = 0E_x \cdot 20_x \oplus u_{5,0} \oplus 0E_x \cdot SB(k_{6,7}) \oplus 0B_x \cdot k_{6,9} \oplus 0D_x \cdot k_{6,10} \oplus \\ \oplus 09_x \cdot SB(k_{6,6}) = 0E_x \cdot k_{6,8} \oplus 0B_x \cdot SB(k_{6,4}) \oplus 0D_x \cdot SB(k_{6,5}) \oplus 09_x \cdot k_{6,11}, \end{aligned} \tag{9}$$

and

$$\begin{aligned} c_2 = 0B_x \cdot k_{6,12} \oplus 0D_x \oplus k_{6,13} \oplus 0D_x \cdot k_{6,9} \oplus \\ \oplus 09_x \cdot k_{6,10} = u_{5,7} \oplus 09_x \cdot k_{6,14} \oplus 0E_x \cdot k_{6,15} \oplus 0B_x \cdot k_{6,8} \oplus \oplus 0E_x \cdot k_{6,11}. \end{aligned} \tag{10}$$

Note that the values $c_1, c_2$ can be computed *independently* by the two sides of the MITM procedure presented above. Hence, we can modify the procedure by using $c_1, c_2$ as additional filtering conditions, as follows:

1. Guess bytes $k_{6,SR(\mathrm{Col}(0,3))}$ and $u_{5,\{0,13\}}$ and compute $x_{5,0}, x_{5,1}$. Store in a table the contribution of bytes $x_{5,0}, x_{5,1}$ to Equation (3) (i.e., the concatenation of the values $0E_x \cdot \Delta(x^j_{5,0}) \oplus 0B_x \cdot \Delta(x^j_{5,1})$ for $j = 1, \ldots, 8$), and the contribution of the guessed subkey bytes to Equations (9),(10) (i.e., the values $c_1, c_2$) – 80 bits in total.
2. Guess bytes $k_{6,SR(\mathrm{Col}(1,2))}$ and $u_{5,\{7,10\}}$, and compute $x_{5,2}, x_{5,3}$. Compute the contribution of bytes $x_{5,2}, x_{5,3}$ to Equation (3) and the contribution of the guessed subkey bytes to Equations (9),(10) (i.e., $c_1, c_2$), and search it in the table.
3. For each match in the table, store in $L_1$ the combination $k_6, u_{5,SR(\mathrm{Col}(0))}$.

The modified procedure (which is performed for each guess of the subkey $k_{-1,SR^{-1}(\mathrm{Col}(0))}$ and for each of $2^{32}$ pairs $(P_1, \hat{P}_1)$) is a standard MITM procedure with 80 guessed key bits on each side and an 80-bit filtering. Its time complexity is $2^{80}$ times decrypting a full AES round and one column of another round, for 16 ciphertexts, plus $2 \cdot 2^{80}$ table lookups, which are roughly equivalent to $3 \cdot 2^{80}$ full encryptions. After the filtering, we are left with $2^{80}$ candidates for 160 subkey bits,

for each guess of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$ and each pair $(P_1, \hat{P}_1)$ – that is, a total of $2^{80} \cdot 2^{32} \cdot 2^{32} = 2^{144}$ candidates.

Naively, the memory complexity so far is $2^{80}$ since we performed a MITM procedure with 80 guessed subkey bits on each side. However, the complexity can be reduced to $2^{40}$ by transforming the MITM procedure into a $Dissect_2(4,1)$ algorithm, as presented in the previous section. This modification does not affect the data and time complexities.

A possible way of continuing the attack is to guess another shifted column of $u_5$ (e.g., $u_{5,SR(\mathrm{Col}(1))}$) and check another condition on $x''_4$ (e.g., $\Delta(x''_{4,7}) = 0$) to obtain another 64-bit filtering. Note that by the AES-192 key schedule, the knowledge of the full $k_6$ gives us for free the bytes $u_{5,\{1,4\}}$. Thus, it is sufficient to guess $u_{5,\{11,14\}}$. If we just guess these key bytes and check the condition, the time complexity of the attack increases to about $3 \cdot 2^{160}$.

A better way is to use a small MITM procedure in which on the one side we guess $u_{5,11}$ and compute its contribution to Equation (4), and on the other side we guess $u_{5,14}$ and compute its contribution to Equation (4). (Note that two terms of the equation can be computed using the key bytes we already know, and so are considered constant). In this way, the time complexity of the attack is about $3 \cdot 2^{152}$.

A yet more efficient way is to use a precomputed table, which is similar to the table used in Improvement 3 in Section 4.1. Recall that our goal in this step is to find the values of the two subkey bytes $u_{5,\{11,14\}}$ for which

$$0D_x \cdot \Delta(x^j_{5,5}) \oplus 09_x \cdot \Delta(x^j_{5,6}) = 0B_x \cdot \Delta(x^j_{5,4}) \oplus 0E_x \cdot \Delta(x^j_{5,7})$$

holds for $j = 1, \ldots, 8$. Using the key material we know so far, we can compute the terms $0B_x \cdot \Delta(x^j_{5,4})$ and $0D_x \cdot \Delta(x^j_{5,5})$, as well as the equivalent state bytes $\bar{x}^j_{5,14}$ and $\bar{x}^j_{5,11}$ after the subkey addition of $u_5$ and before the interchanged MixColumns operation, for all $j$. Hence, we can rewrite the equation as:

$$09_x \cdot \Delta(SB^{-1}(\bar{x}^j_{5,14} \oplus u_{5,14})) \oplus 0E_x \cdot \Delta(SB^{-1}(\bar{x}^j_{5,11} \oplus u_{5,11})) = d_j, \qquad (11)$$

for constants $d_j$ whose values depend on the already known subkeys. Of course, the equations for $j = 1, 2$ are already sufficient for determining $u_{5,\{11,14\}}$ almost uniquely.

Now, we can precompute a table of size $2^{64}$ which takes as inputs pairs-of-pairs of two-byte values $(\bar{x}^j_{5,14}, \bar{x}^j_{5,11}), (\hat{\bar{x}}^j_{5,14}, \hat{\bar{x}}^j_{5,11})$, for $j = 1, 2$, and outputs all pairs of subkey bytes $(u_{5,14}, u_{5,11})$ such that Equation (11) holds for $j = 1, 2$. Then in the online phase of the attack, we can recover subkey bytes $u_{5,\{11,14\}}$ by a few S-box evaluations and a single table lookup, and then apply the remaining 48-bit filtering, which reduces the number of key guesses to $2^{144} \cdot 2^{16} \cdot 2^{-64} = 2^{96}$.

At this stage, we can easily conclude the attack by attacking the two remaining columns and checking the remaining key suggestions by trial encryption. This reduces the time complexity of this step to $2^{144}$ table lookups, which are less than $2^{142}$ encryptions and thus are negligible with respect to the previous steps of the attack.

The size of the table can be further reduced to $2^{47}$ 2-byte values, as like in Section 4.1, it is sufficient to store in the table pairs-of-pairs of the form $((0,0),(e'_1,f'_1)),((e'_2,f'_2),(e'_3,f'_3))$, where $(e'_2,f'_2) < (e'_3,f'_3)$ as binary numbers, and then if we are given a pair-of-pairs $((e_0,f_0),(e_1,f_1)),((e_2,f_2),(e_3,f_3))$, we transform it into $((0,0),(e_0 \oplus e_1, f_0 \oplus f_1)),((e_0 \oplus e_2),(f_0 \oplus f_2),(e_0 \oplus e_3, f_0 \oplus f_3))$, rearrange the two last pairs in increasing order, look at the table to find the suggested subkey/s $\hat{k}$, and for each of them, take $\hat{k} \oplus (e_0, f_0)$ as a suggestion for $(u_{5,14}, u_{5,11})$. We do not elaborate on the details as they are exactly the same as in Section 4.1. On the bottom line, the memory complexity is reduced to $2^{47}$ 2-byte values, which are equivalent to $2^{44}$ 128-bit blocks, without affecting the data and time complexities.

To summarize, the data complexity of the attack is $2^{30}$ chosen plaintexts, the memory complexity is $2^{44}$, and the time complexity is $2^{145.6}$ encryptions.

### A.1 A Further Reduction of the Memory Complexity to $2^{32}$

We can further reduce the memory complexity by another refinement of the MITM procedure. This time, instead of using the values $c_1, c_2$ from Equations (9),(10) as an additional filtering, we *externally guess these values* before the MITM procedure. As a result, the external guess increases to 32+32+16 = 80 bits (i.e., 4 bytes of $k_{-1}$, $2^{32}$ pairs, and the values $c_1, c_2$), while the MITM procedure still guesses 80 key bits in each side. On the other hand, as the values $c_1, c_2$ are known, each side of the MITM can guess 64 bits (i.e., 8 byte values) and recover the two remaining bytes via Equations (9),(10), using the knowledge of $c_1$ and $c_2$. Thus, the MITM procedure effectively has only 64 subkey bits on each side, and hence, its time and memory complexities are about $2^{64}$. This does not change the overall time complexity (as 16 bits were guessed externally), but decreases the memory complexity from $2^{80}$ to $2^{64}$.

However, recall that in our actual attack, the memory complexity of the MITM procedure is much lower – $2^{40}$ – due to the use of dissection. Can we reduce the memory complexity of the dissection procedure using the external subkey guess? It turns out we can, but this is more complex. The problem is that in the internal MITM procedures, where only 5 subkey bytes are guessed from each side, the values of $c_1, c_2$ are not sufficient for guessing only 4 subkey bytes from each side and retrieving the fifth one, since each side of Equations (9), (10) contains values from the two sides of each internal MITM procedure.

Instead, we have to further subdivide Equations (9),(10) and introduce additional guesses of intermediate values into the internal MITM procedures. To this end, we define $c_3, c_4, c_5, c_6$, which break each of the parts of Equations (9),(10) into two parts. We start with Equation (9):

$$\begin{aligned} c_3 &= c_1 \oplus 0E_x \cdot 20_x \oplus u_{5,0} \oplus 0E_x \cdot SB(k_{6,7}) \oplus 0D_x \cdot k_{6,10} = \\ &= 0B_x \cdot k_{6,9} \oplus 09_x \cdot SB(k_{6,6}), \end{aligned} \tag{12}$$

and

$$c_4 = c_1 \oplus 0B_x \cdot SB(k_{6,4}) \oplus 09_x \cdot k_{6,11} = 0E_x \cdot k_{6,8} \oplus \oplus 0D_x \cdot SB(k_{6,5}). \tag{13}$$

Note that in Equation 12, the left hand side can be computed given only the subkey bytes $k_{6,SR(\mathrm{Col}(0))}$ and $u_{5,0}$, while the right hand side can be computed given only the subkey bytes $k_{6,SR(\mathrm{Col}(3))}$ and $u_{5,13}$. Hence, each side of the first internal MITM procedure is able to compute $c_3$. Similarly, in Equation 13, the left hand side can be computed given only the subkey bytes $k_{6,SR(\mathrm{Col}(1))}$ and $u_{5,7}$ while the right hand side can be computed given only the subkey bytes $k_{6,SR(\mathrm{Col}(2))}$ and $u_{5,10}$. Hence, each side of the second internal MITM procedure is able to compute $c_4$.

Now we do the same for Equation 10:

$$c_5 = c_2 \oplus 0D_x \cdot k_{6,13} \oplus 09_x \cdot k_{6,10} = 0B_x \cdot k_{6,12} \oplus \oplus 0D_x \cdot k_{6,9}, \tag{14}$$

and

$$c_6 = c_2 \oplus u_{5,7} \oplus 09_x \cdot k_{6,14} \oplus 0E_x \cdot k_{6,11} = 0E_x \cdot k_{6,15} \oplus 0B_x \cdot k_{6,8}. \tag{15}$$

Now we are ready to present the improved attack algorithm and analyze it.

1. **Constructing the plaintext pool.** Take a structure $S$ of $2^{30}$ chosen plaintexts with the same value in all bytes but those of $SR^{-1}(\mathrm{Col}(0))$.
2. For each guess of subkey $k_{-1,SR^{-1}(\mathrm{Col}(0))}$, go over all chosen pairs of plaintexts $(P_1, \hat{P}_1)$ in $S$, and for each of them do the following:
   (a) **Checking that the pair can be used in the attack, i.e., the pair and all its 7 mixture are in the plaintext pool.** Partially encrypt $(P_1, \hat{P}_1) = (x^1_{-1}, \hat{x}^1_{-1})$ through AddRoundKey and round 0 to obtain $(x^1_1, \hat{x}^1_1)$. Consider all 7 mixture of $(x^1_1, \hat{x}^1_1)$, which we denote $(x^2_1, \hat{x}^2_1), \ldots, (x^8_1, \hat{x}^8_1)$, and partially decrypt them to find the corresponding plaintext pairs $(P_2, \hat{P}_2) = (x^2_{-1}, \hat{x}^2_{-1}), \ldots, (P_8, \hat{P}_8) = (x^8_{-1}, \hat{x}^8_{-1})$. Check whether for all of them, both plaintexts are included in $S$. If no, discard the pair $(P_1, \hat{P}_1)$.
   (b) **External guess of two auxiliary bytes.** Guess the byte values $c_1, c_2$ defined in Equations (9) and (10).
   (c) For each candidate value of $(a_1, a_2, a_3, a_4)$ defined in Equation 5:
   (d) **First internal MITM procedure:**
      i. Guess bytes $c_3$ and $c_5$ defined in Equations (12) and (14), respectively.
      ii. Guess bytes $k_{6,\{0,7,10\}}$ and use the knowledge of $c_3, c_5$ to compute $k_{6,13}$ and $u_{5,0}$. Partially decrypt the ciphertext pairs to obtain the values in the state $x_{5,0}$. Store in a table the contribution of the byte $x_{5,0}$ to Equation (3) for the pairs $(C_1, \hat{C}_1), \ldots, (C_4, \hat{C}_4)$, i.e., the concatenation of the values $0E_x \cdot \Delta(x^i_{5,0}) \oplus a_i$ for $i = 1, \ldots, 4$ – 32 bits in total.
      iii. Guess bytes $k_{6,\{3,9\}}$ and $u_{5,13}$ and use the knowledge of $c_3, c_5$ to compute $k_{6,6}$ and $k_{6,12}$. Partially decrypt the ciphertext pairs to obtain the values in the state $x_{5,1}$. Compute the contribution of the byte $x_{5,1}$ to Equation (3) for the pairs $(C_1, \hat{C}_1), \ldots, (C_4, \hat{C}_4)$, i.e., the concatenation of the values $0B_x \cdot \Delta(x^i_{5,1})$ for $i = 1, \ldots, 4$, and check it in the table.

iv. For each value found in the table, use the suggested value of $k_{6,SR(\mathrm{Col}(0,3))}$ and $u_{5,\{0,13\}}$ and partially decrypt the ciphertexts to obtain the values $(a_5, a_6, a_7, a_8)$. Store them in a table, together with the suggestion for $k_{6,SR(\mathrm{Col}(0,3))}$ and $u_{5,\{0,13\}}$.

(e) **Second internal MITM procedure:**

i. Guess bytes $c_4$ and $c_6$ defined in Equations (13) and (15), respectively.

ii. Guess bytes $k_{6,\{1,11\}}$ and $u_{5,7}$ and use the knowledge of $c_4, c_6$ to compute $k_{6,4}$ and $k_{6,14}$. Partially decrypt the ciphertext pairs to obtain the values in the state $x_{5,3}$. Store in a table the contribution of the byte $x_{5,3}$ to Equation (3) for the pairs $(C_1, \hat{C}_1), \ldots, (C_4, \hat{C}_4)$, i.e., the concatenation of the values $09_x \cdot \Delta(x^i_{5,3}) \oplus a_i$ for $i = 1, \ldots, 4$ – 32 bits in total.

iii. Guess bytes $k_{6,\{2,8\}}$ and $u_{5,10}$ and use the knowledge of $c_4, c_6$ to compute $k_{6,5}$ and $k_{6,15}$. Partially decrypt the ciphertext pairs to obtain the values in the state $x_{5,2}$. Compute the contribution of the byte $x_{5,2}$ to Equation (3) for the pairs $(C_1, \hat{C}_1), \ldots, (C_4, \hat{C}_4)$, i.e., the concatenation of the values $0D_x \cdot \Delta(x^i_{5,1})$ for $i = 1, \ldots, 4$, and check it in the table.

iv. For each value found in the table, use the suggested value of $k_{6,SR(\mathrm{Col}(1,2))}$ and $u_{5,\{7,10\}}$ and partially decrypt the ciphertexts to obtain the values $(a_5, a_6, a_7, a_8)$. Check whether the vector exists in the table. If yes, output the suggested value of $k_5$ and $u_{5,SR(\mathrm{Col}(0))}$ for subsequent analysis (on-the-fly).

(f) **Attacking the second column to obtain additional filtering:** For each value of $k_5$ and $u_{5,SR(\mathrm{Col}(0))}$ suggested in the previous step:

i. Guess $u_{5,11}$, compute its contribution to Equation (4), i.e., the concatenation of the values $0E_x \cdot \Delta(x^i_{5,7})$ for $i = 1, \ldots, 8$, and store it in a table. (The table contains $2^8$ 64-bit values.)

ii. Guess $u_{5,14}$ and compute the sum of the rest of the terms in Equation (4), i.e., the concatenation of the values $0D_x \cdot \Delta(x^i_{5,5}) \oplus 09_x \cdot \Delta(x^i_{5,6}) \oplus 0B_x \cdot \Delta(x^i_{5,4})$ for $i = 1, \ldots, 8$, and search for it in the table. If no match is found, discard the key guess. Otherwise, continue to the next step with the suggestion for $k_6$ and $u_{0,1,4,7,10,11,13,14}$.

(g) **Checking the remaining key suggestions by trial encryption.** For each remaining key suggestion, guess the rest of $u_5$ (only bytes $u_{5,\{8,9,12,15\}}$ are unknown at this stage), use the knowledge of $u_6, k_5$ to recover the key, and check it by trial encryption. If the check fails, move to the next pair $(P_1, \hat{P}_1)$.

Let us analyze the time complexity of the attack. The first internal MITM procedure (Step 2(d)) is performed for each guess of $k_{-1,SR^{-1}(\mathrm{Col}(0))}$, each of $2^{32}$ pairs $(P_1, \hat{P}_1)$, and each guess of $c_1, c_2, a_1, a_2, a_3, a_4$ – that is, $2^{112}$ times in total. In this procedure, we guess two more bytes $(c_3, c_5)$ and for each guess, we perform $2^{24}$ decryptions of two columns through one AES round for 8 ciphertexts, plus $2 \cdot 2^{24}$ table lookups, which are roughly equivalent to $2^{24}$ full encryptions.

Hence, the time complexity of this step is $2^{112} \cdot 2^{16} \cdot 2^{24} = 2^{152}$ encryptions. The complexity of the second internal MITM procedure (Step 2(e)) is the same. Each of the internal MITM procedures yields $2^{32}$ key suggestions (as a total of 10 key bytes are guessed, and there is a 48-bit filtering), and thus the external MITM procedure takes $2^{112} \cdot 2^{32} = 2^{144}$ time, which is negligible with respect to the previous steps. At the end of Step 2(e), we are left with $2^{144}$ suggestions. For each of them, we apply the MITM procedure of Step 2(f), which consists of $2^8$ computations of a single round for 16 plaintexts and $2 \cdot 2^8$ table lookups. To save time, we can implement this step gradually – first checking the condition for two pairs and then for the rest of the pairs (as a 16-bit filtering is already sufficient to make the other steps negligible). Hence, the complexity of this step is less than $2^{144} \cdot 2^8 = 2^{152}$ encryptions. The complexity of Step 2(g) is negligible. Therefore, the total time complexity of the attack is $2^{153}$ encryptions.

As for the memory complexity: Storing the plaintexts requires $2^{30}$ 128-bit blocks. The internal MITM procedures requires $2^{24}$ (due to the external guesses), the external MITM procedure requires $2^{32}$ 80-bit values, and the other steps are negligible. (Note that the $2^{32}$ key suggestions that are output from Step 2(e) are analyzed on-the-fly, and so we do not have to store them). Therefore, the total memory complexity of the attack is less than $2^{32}$ 128-bit blocks.

To summarize, the attack requires $2^{30}$ chosen plaintexts, and has memory and time complexities of $2^{32}$ and $2^{153}$, respectively.