New Representations of the AES Key Schedule

Gaëtan Leurent and Clara Pernot

Inria, Paris, France
{gaetan.leurent,clara.pernot}@inria.fr

Abstract. In this paper we present a new representation of the AES key schedule, with some implications to the security of AES-based schemes. In particular, we show that the AES-128 key schedule can be split into four independent parallel computations operating on 32 bits chunks, up to linear transformation. Surprisingly, this property has not been described in the literature after more than 20 years of analysis of AES. We show two consequences of our new representation, improving previous cryptanalysis results of AES-based schemes.
First, we observe that iterating an odd number of key schedule rounds results in a function with short cycles. This explains an observation of Khairallah on mixFeed, a second-round candidate in the NIST lightweight competition. Our analysis actually shows that his forgery attack on mixFeed succeeds with probability 0.44 (with data complexity 220GB), breaking the scheme in practice. The same observation also leads to a novel attack on ALE, another AES-based AEAD scheme.
Our new representation also gives efficient ways to combine information from the first subkeys and information from the last subkeys, in order to reconstruct the corresponding master keys. In particular we improve previous impossible differential attacks against AES-128.

Keywords: AES · Key schedule · mixFeed · ALE · Impossible Differential Attack

1 Introduction

The AES [1,17] is the most widely used block cipher today, designed by Daemen and Rijmen in 1999 and selected for standardization by NIST. Like all symmetric cryptography primitives, the security of the AES can only be evaluated with cryptanalysis, and there is a constant effort to study its resistance again old and new attacks, and to evaluate its security margin. There are three versions of AES, with different key sizes, and different number of rounds: AES-128 with 10 rounds, AES-192 with 12 rounds, and AES-256 with 14 rounds. After twenty years of cryptanalysis, many different attacks have been applied to AES, and we have a strong confidence in its security: the best attacks against AES-128 in the single-key setting reach only 7 rounds out of 10. The best attacks known so far are either impossible differential attacks (following a line of work starting with [2]) or meet-in-the-middle attacks (with a line of work starting from [18]), as listed in Table 2.

Table 1. Comparison of attacks against ALE.

Attack		Enc.	Verif.	Time	Ref.
Existential Forgery	Known Plaintext	$2^{110.4}$	2^{102}	$2^{110.4}$	[34]
Existential Forgery	Known Plaintext	2^{103}	2^{103}	2^{104}	[30]
Existential Forgery	Known Plaintext	1	2^{120}	2^{120}	[30]
State Recovery, Almost Univ. Forgery	Known Plaintext	1	2^{121}	2^{121}	[30]
State Recovery, Almost Univ. Forgery	Chosen Plaintext	$2^{57.3}$	0	$2^{104.4}$	**New**

Table 2. Best single-key attacks against 7-round AES-128.

Attack	Data	Time	Mem.	Ref.	Note
Meet-in-the-middle	2^{97}	2^{99}	2^{98}	[19]	
	2^{105}	2^{105}	2^{90}	[19]	
	2^{105}	2^{105}	2^{81}	[9]	
	2^{113}	2^{113}	2^{74}	[9]	
Impossible differential	2^{113}	2^{113}	2^{74}	[13]	Using 4 out. diff. and state-test
	$2^{105.1}$	2^{113}	$2^{74.1}$	[13][a]	Using 4 out. diff
	$2^{106.1}$	$2^{112.1}$	$2^{73.1}$		Variant of [13] using 1 out. diff.
	$2^{104.9}$	$2^{110.9}$	$2^{71.9}$	**New**	Using 1 out. diff.

[a] The time complexity is incorrectly given as $2^{106.88}$ in [13].

1.1 Our results

The key schedule is arguably the weakest part of the AES, and it is well known to cause issues in the related-key setting [5,6,7]. In this paper, we focus on the key schedule of AES, and we show a surprising alternative representation, where the key schedule is split into several independent chunks, and the actual subkeys are just linear combinations of the chunks.

Application to mixFeed and ALE. This representation is motivated by an observation made by Khairallah [29] on the AEAD scheme mixFeed: when the 11-round AES-128 key schedule is iterated there are apparently many short cycles of length roughly 2^{34}. Our representation explains this observation, and proves that the forgery attack of Khairallah against mixFeed actually succeeds with a very high probability. It only requires the encryption of one known message of length at least $2^{33.7}$ blocks, and generates a forgery with probability 0.44, making it a practical break of the scheme.

We also apply the same observation to ALE, another AES-based scheme that iterates the AES key schedule. We obtain a novel attack against ALE, with a much lower data complexity than previous attacks, but we need chosen plaintexts rather than known plaintexts (see Table 1).

Key recovery attack against AES-128. We also improve key recovery attacks against AES-128 based on impossible differential cryptanalysis. This type of at-

tacks targets bytes of the first subkey and of the last subkey, and excludes some values that are shown impossible. Then, the attacker must iterate over the remaining candidates, and reconstruct the corresponding master keys. Using our new representation of the key schedule, we make the reconstruction of the master key more efficient. Therefore we can start from a smaller data set: we identify fewer impossible keys, but we process the larger number of key candidates without making this step the bottleneck.

While the improvement is quite modest (see Table 2), it is notable that we improve this attack in a non-negligible way, because cryptanalysis of AES has achieved a high level of technicality, and attacks are already thoroughly optimized. In particular, we obtain the best attack so far when the amount of memory is limited (*eg.* below 2^{75}).

1.2 Organisation of the paper

We start with a description of the AES-128 key schedule and describe our alternative representation in Section 2, before presenting applications to mixFeed (Section 3), ALE (Section 4) and impossible differential attacks against AES-128 (Section 5). We then describe an alternative representation of the AES-192 and AES-256 key schedules in Section 6, and some properties of the AES key schedules that might be useful in future works in Section 7.

2 A New Representation of the AES-128 Key Schedule

In AES-128, the key schedule is an iterative process to derive 11 subkeys from one master key. To start with, the 128 bits of the master key are divided into 4 words of 32 bits each: w_i for $0 \le i \le 3$. The following notations are used within the algorithm:

RotWord performs a cyclic permutation of one byte to the left.
SubWord applies the AES Sbox to each of the 4 bytes of a word.
RCon(i) is a round constant defined as $[x^{i-1}, 0, 0, 0]$ in the field $\mathbb{F}_{2^8}$ described in [1]. For simplicity, we denote x^{i-1} as c_i.

In order to construct w_i for $i \ge 4$, one applies the following steps:

- if $i \equiv 0 \bmod 4$, $w_i = \mathsf{SubWord}(\mathsf{RotWord}(w_{i-1})) \oplus \mathsf{RCon}(i/4) \oplus w_{i-4}$.
- else, $w_i = w_{i-1} \oplus w_{i-4}$.

The subkey at round r is the concatenation of the words w_{4r} to w_{4r+3}. We can also express the key schedule at the byte level, using k_i^r with $0 \le i < 16$ to denote byte i of the round-r subkey (we use $k^r_{\langle i,j,\ldots\rangle}$ as a shorthand for $k_i^r, k_j^r, \ldots$). The subkey is typically represented as a 4×4 matrix with the AES byte ordering, with $w_i = k^{i/4}_{4(i \bmod 4)} \| k^{i/4}_{4(i \bmod 4)+1} \| k^{i/4}_{4(i \bmod 4)+2} \| k^{i/4}_{4(i \bmod 4)+3}$:

$$\begin{bmatrix} k_0^r & k_4^r & k_8^r & k_{12}^r \\ k_1^r & k_5^r & k_9^r & k_{13}^r \\ k_2^r & k_6^r & k_{10}^r & k_{14}^r \\ k_3^r & k_7^r & k_{11}^r & k_{15}^r \end{bmatrix} = \begin{bmatrix} \\ w_{4r} & w_{4r+1} & w_{4r+2} & w_{4r+3} \\ \\ \end{bmatrix}$$

The key schedule can be written as follows, with k the key schedule state, k'_i the state after one round of key schedule, and S the AES Sbox (see Figure 1 and 3):

$$
\begin{aligned}
k'_0 &= k_0 \oplus S(k_{13}) \oplus c_i & k'_8 &= k_8 \oplus k_4 \oplus k_0 \oplus S(k_{13}) \oplus c_i \\
k'_1 &= k_1 \oplus S(k_{14}) & k'_9 &= k_9 \oplus k_5 \oplus k_1 \oplus S(k_{14}) \\
k'_2 &= k_2 \oplus S(k_{15}) & k'_{10} &= k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{15}) \\
k'_3 &= k_3 \oplus S(k_{12}) & k'_{11} &= k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{12}) \\
k'_4 &= k_4 \oplus k_0 \oplus S(k_{13}) \oplus c_i & k'_{12} &= k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{13}) \oplus c_i \\
k'_5 &= k_5 \oplus k_1 \oplus S(k_{14}) & k'_{13} &= k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{14}) \\
k'_6 &= k_6 \oplus k_2 \oplus S(k_{15}) & k'_{14} &= k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{15}) \\
k'_7 &= k_7 \oplus k_3 \oplus S(k_{12}) & k'_{15} &= k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{12})
\end{aligned}
$$

Invariant subspaces. Recently, several lightweight block ciphers have been analyzed using *invariant subspace* attacks. This type of attack was first proposed on PRINTcipher by Leander *et al.* [31]; the basic idea is to identify a linear subspace V and an offset u such that the round function F of a cipher satisfies $F(u+V) = F(u) + V$. At Eurocrypt 2015, Leander, Minaud and Rønjom [32] introduced an algorithm in order to detect such invariant subspaces. By applying this algorithm to four rounds of the AES-128 key schedule, we find invariant subspaces of dimension four over $\mathbb{F}_{2^8}$, and this implies a decomposition of the key schedule.

First, let's recall the generic algorithm for a permutation $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$:

1. Guess an offset $u \in \mathbb{F}_2^n$ and a one-dimensional subspace V_0.
2. Compute $V_{i+1} = \operatorname{span}\{(F(u+V_i) - F(u)) \cup V_i\}$.
3. If the dimension of V_{i+1} equals the dimension of V_i, we found an invariant subspace: $F(u+V) = F(u) + V$.
4. Else, we go on step 2.

In the case of the AES-128 key schedule, we use subspaces of $\mathbb{F}_{2^8}^{16}$ over the field $\mathbb{F}_{2^8}$ rather than over $\mathbb{F}_2$. If we apply this algorithm with the permutation F corresponding to 4 rounds of key schedule, with any key state u, and with V_0 the vector space generated by one of the first four bytes, we obtain 4 invariant affine subspaces whose linear parts are:

$$
\begin{aligned}
E_0 &= \{(a,b,c,d, \quad 0,b,0,d, \quad a,0,0,d, \quad 0,0,0,d) \quad \text{for } a,b,c,d \in \mathbb{F}_{2^8}\} \\
E_1 &= \{(a,b,c,d, \quad a,0,c,0, \quad 0,0,c,d, \quad 0,0,c,0) \quad \text{for } a,b,c,d \in \mathbb{F}_{2^8}\} \\
E_2 &= \{(a,b,c,d, \quad 0,b,0,d, \quad 0,b,c,0, \quad 0,b,0,0) \quad \text{for } a,b,c,d \in \mathbb{F}_{2^8}\} \\
E_3 &= \{(a,b,c,d, \quad a,0,c,0, \quad a,b,0,0, \quad a,0,0,0) \quad \text{for } a,b,c,d \in \mathbb{F}_{2^8}\}
\end{aligned}
$$

When we consider a single round R of the key schedule, the subspaces are not invariant, but are images of each other. We have the following relations, with u_0 an element in $(\mathbb{F}_{2^8})^{16}$ and $u_i = R^i(u_0)$, for $(1 \le i < 5)$:

$$
\begin{aligned}
R(E_0 + u_0) &= E_1 + u_1, & R(E_1 + u_1) &= E_2 + u_2, \\
R(E_2 + u_2) &= E_3 + u_3, & R(E_3 + u_3) &= E_0 + u_4
\end{aligned}
$$

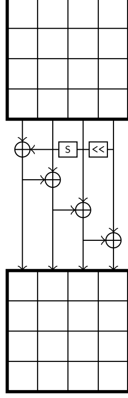

Fig. 1. AES key schedule. (figure adapted from [28])

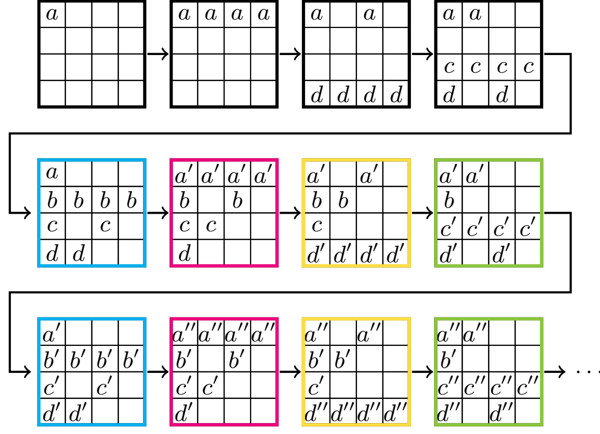

Fig. 2. Evolution of a difference located on the first byte after several rounds of key schedule.

In other words, if the difference pattern between two states is in E_i, then after r rounds of key schedule, the difference pattern will be in $E_{(i+r)\%4}$.

This can be verified by tracking the differences in the key schedule, starting from a difference $(a, 0, 0, 0,\ 0, 0, 0, 0,\ 0, 0, 0, 0,\ 0, 0, 0, 0)$, as shown in Figure 2. After four rounds we reach a difference $(a, b, c, d,\ 0, b, 0, d,\ 0, b, c, 0,\ 0, b, 0, 0)$, with differential transitions $a \to d$, $d \to c$, and $c \to b$ through the Sbox. Next, we obtain a difference $(a', b, c, d,\ a', 0, c, 0,\ a', b, 0, 0,\ a', 0, 0, 0)$, after an Sbox transition $b \to a \oplus a'$. Surprisingly, the dimension of the difference state does not increase, because there is a single active Sbox in each round, and it affects a difference that is already independent of the rest of the state. Therefore we have the four transitions given above, and the spaces are indeed invariant.

New representation from invariant subspaces. We actually have a much stronger property than just invariant spaces: the full space is the direct sum of those four vector spaces, with parallel invariant subspaces for any offset u:

$$(\mathbb{F}_{2^8})^{16} = E_0 \oplus E_1 \oplus E_2 \oplus E_3$$
$$\forall u,\ \forall i,\ F(u \oplus E_i) = F(u) \oplus E_i.$$

This implies that we can split the internal state according to those vector spaces. Indeed, there exists unique linear projections $\pi_i : (\mathbb{F}_{2^8})^{16} \to E_i$ for $0 \leq i < 4$ such that $\forall x \in E_i$, $\pi_i(x) = x$, and $\pi_i(E_j) = 0$ for $i \neq j$. In particular, we have $\forall x$, $x = \pi_0(x) \oplus \pi_1(x) \oplus \pi_2(x) \oplus \pi_3(x)$. This implies:

$$\begin{aligned} F(x) &= F\big(\pi_0(x) \oplus \pi_1(x) \oplus \pi_2(x) \oplus \pi_3(x)\big) \\ &\in F\big(\pi_0(x) \oplus \pi_1(x) \oplus \pi_2(x)\big) \oplus E_3 \\ &\in F\big(\pi_0(x) \oplus \pi_1(x)\big) \oplus E_3 \oplus E_2 \\ &\in F\big(\pi_0(x)\big) \oplus E_3 \oplus E_2 \oplus E_1 \end{aligned}$$

Therefore $\pi_0(F(x)) = \pi_0\big(F(\pi_0(x))\big)$. Similarly, $\pi_i(F(x)) = \pi_i\big(F(\pi_i(x))\big)$, and finally we can split the permutation in four independent 32-bit computations:

$$F(x) = \pi_0\big(F(\pi_0(x))\big) \oplus \pi_1\big(F(\pi_1(x))\big) \oplus \pi_2\big(F(\pi_2(x))\big) \oplus \pi_3\big(F(\pi_3(x))\big).$$

To obtain a representation that makes the 4 subspaces appear clearly, we perform a change of basis. Let $\{e_0, e_1, \ldots, e_{15}\}$ be our new basis of $(\mathbb{F}_{2^8})^{16}$ defined as follows:

$$\text{Base of } E_0 \begin{cases} e_0 &= (0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1) \\ e_1 &= (0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0) \\ e_2 &= (0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0) \\ e_3 &= (1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0) \end{cases}$$

$$\text{Base of } E_1 \begin{cases} e_4 &= (0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0) \\ e_5 &= (0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \\ e_6 &= (1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0) \\ e_7 &= (0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0) \end{cases}$$

$$\text{Base of } E_2 \begin{cases} e_8 &= (0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0) \\ e_9 &= (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \\ e_{10} &= (0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0) \\ e_{11} &= (0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0) \end{cases}$$

$$\text{Base of } E_3 \begin{cases} e_{12} &= (1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0) \\ e_{13} &= (0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0) \\ e_{14} &= (0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0) \\ e_{15} &= (0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0) \end{cases}$$

Let $s_0, s_1, \ldots, s_{15}$ be the coordinates in the new basis. They can be obtained by multiplying the original coordinates $(k_0, \ldots, k_{15})$ with the matrix $A = C_0^{-1}$, where the columns of the transition matrix C_0 are the coordinates of the vectors $e_0, e_1, \ldots, e_{15}$ expressed in the old basis (canonical basis):

$$C_0 = \begin{pmatrix} 0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0 \\ 0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&1 \\ 0&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0 \\ 1&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0 \\ 1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix} \qquad A = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0 \end{pmatrix}$$

Therefore, we use:

$$\begin{array}{cccc} s_0 = k_{15} & s_1 = k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 & s_2 = k_{13} \oplus k_5 & s_3 = k_{12} \oplus k_8 \\ s_4 = k_{14} & s_5 = k_{13} \oplus k_9 \oplus k_5 \oplus k_1 & s_6 = k_{12} \oplus k_4 & s_7 = k_{15} \oplus k_{11} \\ s_8 = k_{13} & s_9 = k_{12} \oplus k_8 \oplus k_4 \oplus k_0 & s_{10} = k_{15} \oplus k_7 & s_{11} = k_{14} \oplus k_{10} \\ s_{12} = k_{12} & s_{13} = k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 & s_{14} = k_{14} \oplus k_6 & s_{15} = k_{13} \oplus k_9 \end{array} \tag{1}$$

After defining s' with the same transformation from k', we can verify that:

$$\begin{aligned}
s'_0 &= k'_{15} = k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{12}) && = s_{13} \oplus S(s_{12}) \\
s'_1 &= k'_{14} \oplus k'_{10} \oplus k'_6 \oplus k'_2 = k_{14} \oplus k_6 && = s_{14} \\
s'_2 &= k'_{13} \oplus k'_5 = k_{13} \oplus k_9 && = s_{15} \\
s'_3 &= k'_{12} \oplus k'_8 = k_{12} && = s_{12} \\
s'_4 &= k'_{14} = k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{15}) && = s_1 \oplus S(s_0) \\
s'_5 &= k'_{13} \oplus k'_9 \oplus k'_5 \oplus k'_1 = k_{13} \oplus k_5 && = s_2 \\
s'_6 &= k'_{12} \oplus k'_4 = k_{12} \oplus k_8 && = s_3 \\
s'_7 &= k'_{15} \oplus k'_{11} = k_{15} && = s_0 \\
s'_8 &= k'_{13} = k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{14}) && = s_5 \oplus S(s_4) \\
s'_9 &= k'_{12} \oplus k'_8 \oplus k'_4 \oplus k'_0 = k_{12} \oplus k_4 && = s_6 \\
s'_{10} &= k'_{15} \oplus k'_7 = k_{15} \oplus k_{11} && = s_7 \\
s'_{11} &= k'_{14} \oplus k'_{10} = k_{14} && = s_4 \\
s'_{12} &= k'_{12} = k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{13}) \oplus c_i && = s_9 \oplus S(s_8) \oplus c_i \\
s'_{13} &= k'_{15} \oplus k'_{11} \oplus k'_7 \oplus k'_3 = k_{15} \oplus k_7 && = s_{10} \\
s'_{14} &= k'_{14} \oplus k'_6 = k_{14} \oplus k_{10} && = s_{11} \\
s'_{15} &= k'_{13} \oplus k'_9 = k_{13} && = s_8
\end{aligned} \tag{2}$$

This is represented by Figure 4. In the rest of this paper we use the notation k_i^r to denote byte i of the round-r subkey, and s_i^r to denote bytes of the alternative representation at round r, where the relations between k_i^r and s_i^r follow (1).

To further simplify the description, we write the output as

$$(s'_4, s'_5, s'_6, s'_7, \quad s'_8, s'_9, s'_{10}, s'_{11}, \quad s'_{12}, s'_{13}, s'_{14}, s'_{15}, \quad s'_0, s'_1, s'_2, s'_3).$$

This corresponds to "untwisting" the rotation of the 4-byte blocks, so that each block of 4 output bytes depends on the same 4 input bytes. This results in our alternate representation of the AES-128 key schedule:

1. We first apply the linear transformation A to the state, corresponding to the change of variable above.
2. Then the rounds of the key schedule are seen as the concatenation of 4 functions each acting on 32-bit words (4 bytes), as seen in Figure 5.
3. In order to extract the subkey of round r, another linear transformation $C_{r \bmod 4}$ is applied to the state, depending of the round number modulo 4. C_i is defined as $C_i = A^{-1} \times \mathsf{SR}^i$, with SR the matrix corresponding to

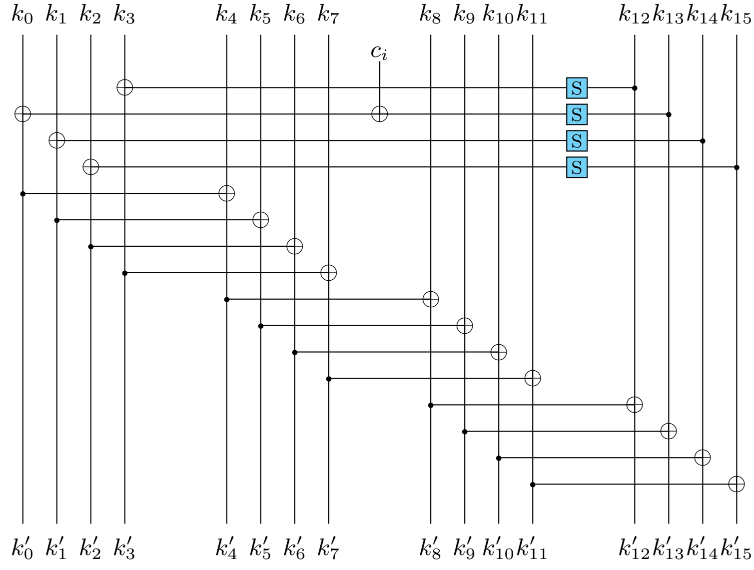

Fig. 3. One round of the AES-128 key schedule.

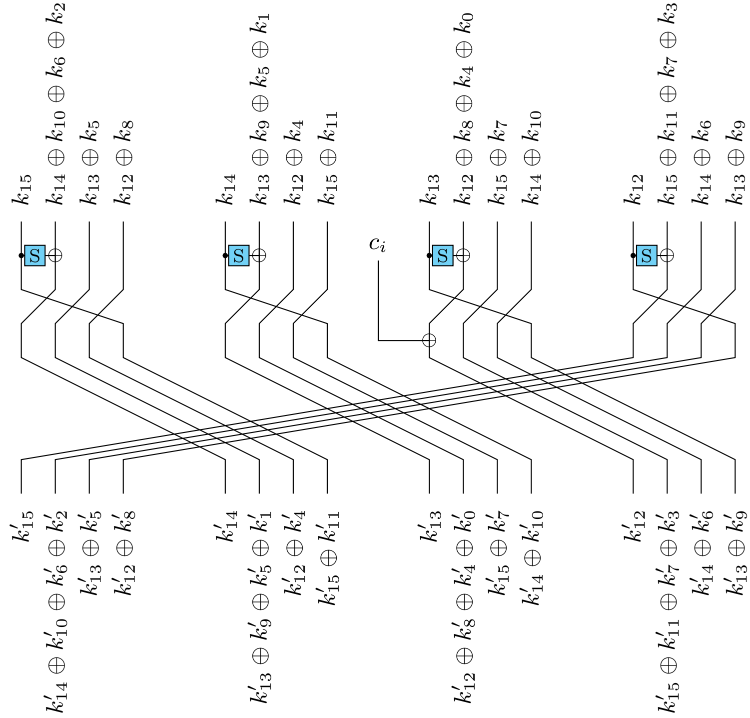

Fig. 4. One round of the AES-128 key schedule (alternative representation).

rotation of 4 bytes to the right (see below). In particular $C_0 = A^{-1}$.

$$A = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0
\end{pmatrix}
\qquad
\mathsf{SR} = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0
\end{pmatrix}$$

In this new representation, there are clearly 4 independent chunks each acting on 4 bytes, and the subkeys are reconstructed with linear combinations of the alternative key schedule state. This representation also preserves the symmetry of the key schedule: the original key schedule is invariant by rotation of the columns (up to constants), and this corresponds to a rotation of four bytes in the new representation.

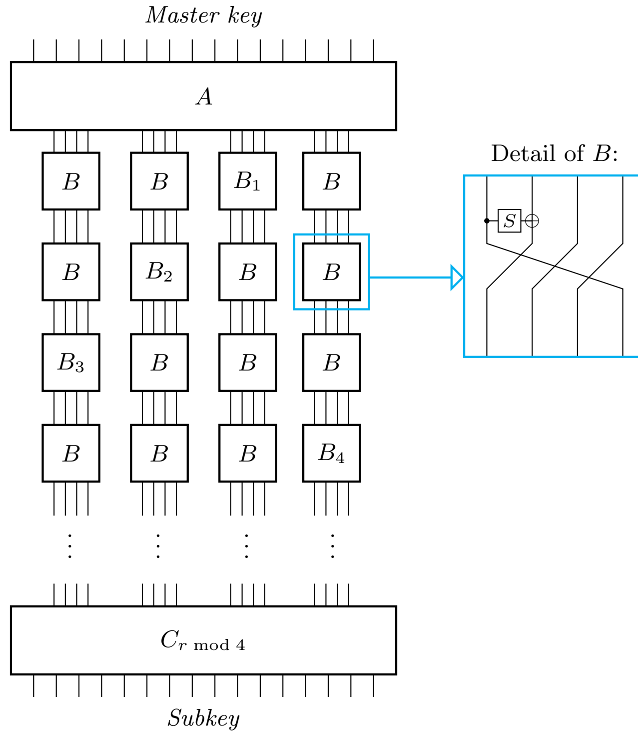

Fig. 5. r rounds of the key schedule in the new representation. B_i is similar to B but the round constant c_i is XORed to the output of the Sbox.

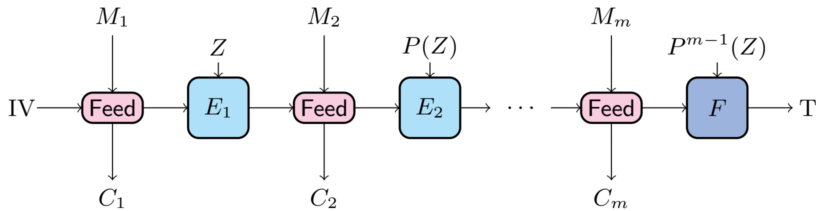

Fig. 6. Simplified scheme of mixFeed encryption.

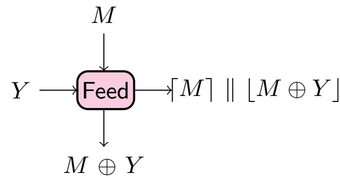

Fig. 7. Function Feed with a full message block.

3 Application to mixFeed

The AEAD scheme mixFeed [14] is a second-round candidate in the NIST Lightweight Standardization Process, submitted by Chakraborty and Nandi, and based on the AES block cipher. It is a rate-1 feedback-based mode inspired by COFB. For each message block, a Feed function is used to compute the ciphertext and the block cipher input from the previous internal state, and the internal state is replaced by the block cipher output. In COFB, there is a need for an extra state variable, to make each Feed function different. In order to reduce the state size, mixFeed instead makes each block cipher call different, applying a permutation P to the key between each block. For optimal efficiency, the permutation P just corresponds to eleven round of the AES key schedule, so that the subkeys for all the AES calls just correspond to running the AES key schedule indefinitely.

In [29], Khairallah observed that some keys generate short cycles when iterating the P permutation, and he built a forgery attack for keys in short cycles. In this work, we show that the new representation of the key schedule explains the existence of these short cycles, and we characterize the keys belonging to such cycles. This shows that the permutation P cannot be considered as a random permutation.

3.1 Description of mixFeed

For simplicity, we only describe a simplified mixFeed without associated data; the full description of mixFeed can be found in [14].

Notations: We use M and C to denote the plaintext and ciphertext. For the sake of simplicity, we assume that M is made of m 128-bit blocks.

The following functions are used in mixFeed:

- E: a modified version of AES-128 including MixColumns in the last round;
- P: the permutation corresponding to eleven rounds of AES-128 key schedule;
- Feed: the feedback function defined as (see Figure 7):

$$\begin{aligned}\mathsf{Feed}(Y, M) &= (X, C) \\ &= (\lceil M \rceil \| \lfloor M \oplus Y \rfloor, M \oplus Y),\end{aligned}$$

  where $\lceil D \rceil$ represent the 64 most significant bits of D, and $\lfloor D \rfloor$ the 64 least significant bits.

The computations are as follow (see Figure 6):

Initialization of the state. An initial value IV $= Y_0$ and a internal key Z are computed from the nonce N and the key K.

Encryption and authentication. For i from 1 to m, the Feed function is applied to the current state Y_{i-1} and message block M_i. Feed returns the ciphertext block C_i, and a new state X_i which is then encrypted under the key $P^{i-1}(Z)$ using E to obtain Y_i. At the end of this step, a finalization function computes the tag from the final state and the internal key $P^{m-1}(Z)$, we denote as F the composition of the cipher call of last round and the finalization function.

3.2 Short Cycles of P

In [29], Khairallah found 20 keys belonging to small cycles of P, and observed that all of them have the same cycle length[1]: 14018661024. He deduced a forgery attack, assuming that the subkey falls in one of those cycles, but did not further analyse the probability of having such a subkey. Later the designers of mixFeed published a security proof for the scheme [15], under the assumption that the number of keys in a short cycle is sufficiently small. More precisely, they wrote:

Assumption 1 ([15]) *For any $K \in \{0,1\}^n$ chosen uniformly at random, probability that K has a period at most ℓ is at most $\ell/2^{n/2}$.*

The 20 keys identified by Khairallah do not contradict this assumption, but if there are many such keys the assumption does not hold, and mixFeed can be broken by a forgery attack. We now provide a theoretical explanation of the observation of Khairallah, and a full characterization of the cycles of P. We find that a random key is in a cycle of length smaller than 2^{34} with probability 0.44; this contradicts the assumption made in [15], and allows a practical forgery attack.

[1] Khairallah actually reported the length as 1133759136, probably because of a 32-bit overflow.

Analysis of the structure of P. Using our new representation, the 11-round key schedule P consists of:

- The linear transformation A
- 4 parallel 32-bit functions that we denote $f_1 \| f_2 \| f_3 \| f_4$, with

$$f_1 = B_{11} \circ B \circ B \circ B \circ B_7 \circ B \circ B \circ B \circ B_3 \circ B \circ B$$
$$f_2 = B \circ B_{10} \circ B \circ B \circ B \circ B_6 \circ B \circ B \circ B \circ B_2 \circ B$$
$$f_3 = B \circ B \circ B_9 \circ B \circ B \circ B \circ B_5 \circ B \circ B \circ B \circ B_1$$
$$f_4 = B \circ B \circ B \circ B_8 \circ B \circ B \circ B \circ B_4 \circ B \circ B \circ B$$

  (the functions differ only by the round constants)
- The linear transformation $C_3 = A^{-1} \times \mathsf{SR}^{-1}$

To simplify the analysis, we consider the cycle structure of $\widetilde{P} = A \circ P \circ A^{-1}$, which is the same as the cycle structure of P:

$$\widetilde{P} : (a, b, c, d) \mapsto (f_2(b), f_3(c), f_4(d), f_1(a))$$

To further simplify the analysis, we consider the cycle structure of $\widetilde{P}^4$, which is closely related to the cycle structure of $\widetilde{P}$. A cycle of $\widetilde{P}^4$ of length ℓ corresponds to a cycle of $\widetilde{P}$, of length ℓ, 2ℓ or 4ℓ. Conversely a cycle of $\widetilde{P}$ of length ℓ corresponds to one or several cycles of $\widetilde{P}^4$, of length ℓ, $\ell/2$ or $\ell/4$ (depending on the divisibility of ℓ). Analyzing $\widetilde{P}^4$ is easier because it can be decomposed into 4 parallel functions, cancelling the left rotation induced by SR^{-1}:

$$\widetilde{P}^4 : (a, b, c, d) \mapsto (\phi_1(a), \phi_2(b), \phi_3(c), \phi_4(d))$$
$$\phi_1(a) = f_2 \circ f_3 \circ f_4 \circ f_1(a)$$
$$\phi_2(b) = f_3 \circ f_4 \circ f_1 \circ f_2(b)$$
$$\phi_3(c) = f_4 \circ f_1 \circ f_2 \circ f_3(c)$$
$$\phi_4(d) = f_1 \circ f_2 \circ f_3 \circ f_4(d)$$

If (a, b, c, d) is in a cycle of length ℓ of $\widetilde{P}^4$, we have $\widetilde{P}^{4\ell}(a, b, c, d) = (a, b, c, d)$, that is to say:

$$\phi_1^\ell(a) = a \qquad \phi_2^\ell(b) = b \qquad \phi_3^\ell(c) = c \qquad \phi_4^\ell(d) = d$$

In particular, a, b, c and d must be in cycles of ϕ_1, ϕ_2, ϕ_3, ϕ_4 (respectively) of length dividing ℓ. Conversely, if a, b, c, d are in small cycles of the corresponding ϕ_i, then (a, b, c, d) is in a cycle of $\widetilde{P}^4$ of length the lowest common multiple of the small cycle lengths.

Moreover, due to the structure of the ϕ_i functions, all of them have the same cycle structure. This implies that $\widetilde{P}$ has a large number of small cycles. Indeed, if we consider a cycle of ϕ_i of length ℓ, and elements a, b, c, d in the corresponding cycles, (a, b, c, d) is in a cycle of P^4 of length ℓ. There are ℓ^4 choices of a, b, c, d, which correspond to ℓ^3 different cycles of P. If we assume that ϕ_i behaves like a random 32-bit permutation, we expect that the largest cycle has length about 2^{31}, which gives around 2^{93} cycles of $\widetilde{P}^4$ of length $\approx 2^{31}$, and around 2^{93} cycles of $\widetilde{P}$ of length $\approx 2^{33}$.

Cycle analysis of 11-round AES-128 key schedule. In order to identify the small cycles of the permutation P, we start by analyzing the cycle structure of the 32-bit function $\phi_1 = f_2 \circ f_3 \circ f_4 \circ f_1$: it can be decomposed into cycles of lengths 3504665256, 255703222, 219107352, 174977807, 99678312, 13792740, 8820469, 7619847, 5442633, 4214934, 459548, 444656, 14977, 14559, 5165, 4347, 1091, 317, 27, 6, 5 (3 cycles), 4 (2 cycles), 2 (3 cycles), and 1 (2 fixed points). In particular, the largest cycle has length $\ell = 3504665256$. Consequently, with probability $(3504665256 \times 2^{-32})^4 \approx 0.44$, we have a, b, c and d in a cycle of length ℓ, resulting in a cycle of length ℓ for $\widetilde{P}^4$, and a cycle of length at most $4\ell = 14018661024$ for $\widetilde{P}$ and P. This explains the observation of Khairallah [29], and clearly contradicts the assumption of [15].

More generally, when a, b, c, d belong to a cycle of length ℓ_i, the corresponding cycle for $\widetilde{P}^4$ is of length $\ell = \mathrm{lcm}(\ell_1, \ell_2, \ell_3, \ell_4)$, and we can compute the associated probability. In most cases, a cycle of length ℓ of $\widetilde{P}^4$ corresponds to a cycle of $\widetilde{P}$ of length 4ℓ. However, the cycle of $\widetilde{P}$ is of length ℓ when $\widetilde{P}^\ell(a, b, c, d) = (a, b, c, d)$, and of length 2ℓ when $\widetilde{P}^{2\ell}(a, b, c, d) = (a, b, c, d)$ (this can only be the case with odd ℓ, by definition of ℓ). This is unlikely for short cycles, but as an example we can construct a fixed-point for $\widetilde{P}$ and P from a fixed-point of ϕ_1:

- $a =$ `7e be d1 92`
- $b =$ `de d4 b7 cc` $= f_3 \circ f_4 \circ f_1(a)$
- $c =$ `9f 95 88 26` $= f_4 \circ f_1(a)$
- $d =$ `d4 b9 79 91` $= f_1(a)$

Since $f_2 \circ f_3 \circ f_4 \circ f_1(a) = a$, we have $\widetilde{P}(a, b, c, d) = (f_2(b), f_3(c), f_4(d), f_1(a)) = (a, b, c, d)$. Since $\widetilde{P} = A \circ P \circ A^{-1}$, the corresponding key in the original representation is:

$$A^{-1} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} \texttt{64 0b 3f 83 63 4e a7 f6 46 0e f8 b2 d4 9f de 7e} \end{pmatrix}^\top$$

This results in a fixed point of P.

We can generalize this construction for all odd cycle lengths ℓ. We choose w an element of a cycle of length ℓ, and then we can build an element which belongs to a cycle of length ℓ for the permutation P:

- if $\ell = 1 \bmod 4$:

$$\begin{aligned} a &= w \\ b &= f_3 \circ f_4 \circ f_1 \circ \ldots \circ f_1(w), && \text{with } 3\ell \text{ terms } f_i \\ c &= f_4 \circ f_1 \circ f_2 \circ \ldots \circ f_1(w), && \text{with } 2\ell \text{ terms } f_i \\ d &= f_1 \circ f_2 \circ f_3 \circ \ldots \circ f_1(w), && \text{with } \ell \text{ terms } f_i \end{aligned}$$

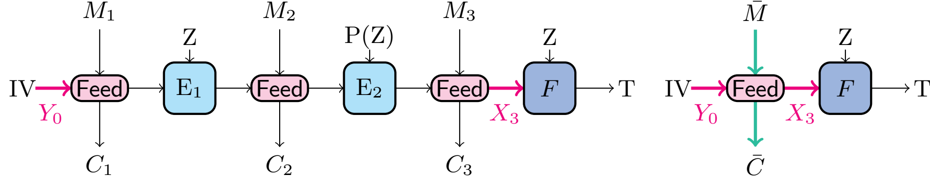

Fig. 8. Forgery attack when Z belongs to a cycle of length 2.

– if $\ell = 3 \bmod 4$:

$$\begin{aligned} a &= w \\ b &= f_3 \circ f_4 \circ f_1 \circ \ldots \circ f_1(w), && \text{with } \ell \text{ terms } f_i \\ c &= f_4 \circ f_1 \circ f_2 \circ \ldots \circ f_1(w), && \text{with } 2\ell \text{ terms } f_i \\ d &= f_1 \circ f_2 \circ f_3 \circ \ldots \circ f_1(w), && \text{with } 3\ell \text{ terms } f_i \end{aligned}$$

3.3 Forgery attack against mixFeed

Khairallah [29] proposed a forgery attack assuming that Z belongs to a cycle of length ℓ, considering a message M made of m blocks, with $m > \ell$:

1. Encrypt the message M to obtain the ciphertext C and tag T.
2. Compute Y_0 using M_1 and C_1 and $X_{\ell+1}$ using $M_{\ell+1}$ and $C_{\ell+1}$.
3. Compute $\bar{M}$ and $\bar{C}$ such that $(X_{\ell+1}, \bar{C}) = \mathsf{Feed}(Y_0, \bar{M})$.
4. The T tag will also authenticate the new ciphertext $C' = \bar{C} \| C_{\ell+2} \| \cdots \| C_m$.

The computations required for the forge are negligible with only a few XORs to invert the Feed function. Therefore the complexity of the attack is just the encryption of a message with at least $(\ell+1)$ blocks, with ℓ the length of the cycle. As explained above, the probability of success is approximately 0.44, using $\ell =$ 14018661024. When the forgery fails, we can repeat the attack with a different nonce, because the internal key Z depends on the nonce; for each master key K, the attack works on 44% of the nonces.

We have verified this attack using the reference implementation provided by the designers. We take a message of $\ell + 1 = 14018661025$ blocks of 16 bytes (220 Gbytes[2]), choose a random key and nonce, and encrypt the message with mixFeed. We modify the ciphertext according to the previous explanation, and we check if the new ciphertext is accepted. We obtained 41% of success over 100 attempts. This result is close to the expected 44% success rate, and confirms our analysis.

[2] Note that there is no need to store the plaintext or ciphertext in memory if we have access to an online implementation of mixFeed.

4 Application to ALE

ALE [8] is an earlier authenticated encryption scheme based on the AES round function, strongly inspired by LEX [4] (for the encryption part) and Pelican-MAC [16] (for the authentication part). Attacks have already been presented against ALE [30,34] but the new representation of the key schedule gives new types of attacks, based on previous attacks against LEX [11,20].

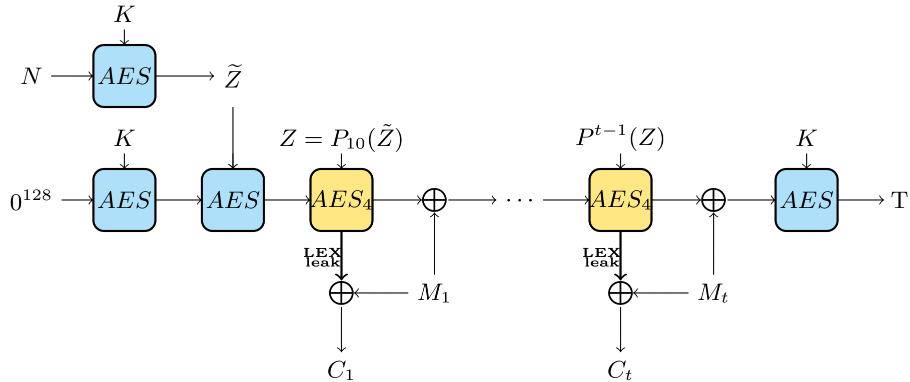

Fig. 9. Simplified authenticated encryption with ALE.

4.1 Description of ALE

For the sake of simplicity, we will consider ALE without associated data, and we only consider blocks of 16 bytes for the plaintext (to ignore the padding). ALE maintains a state composed of an internal state and an internal key, and operates with 3 steps (cf Figure 9). As for mixFeed, the internal key is updated with iterative applications of a permutation P corresponding to AES key schedule rounds. In the case of ALE, P corresponds to 5 rounds of key schedule rather than 11, but we have again many short cycles because 5 is also an odd number.

Initialization. The state is initialized from the key K and a nonce N, using a session key $\widetilde{Z} = E_K(N)$. The internal state is initialized to $IV = E_{\widetilde{Z}}(E_K(0))$, and the internal key is initialized to $P_{10}(Z)$, where P_{10} correspond to 11 rounds of AES key schedule.

Message processing phase. For each block of message, the internal state is encrypted with 4-round AES, and the internal key is updated by five rounds of AES key schedule. During the encryption, four bytes are leaked in each AES round according to the LEX specification (bytes 0, 2, 8, 10 for odd rounds, and bytes 4, 6, 12 and 14 for even rounds), and used as keystream to encrypt the message. Then the message block is xored to the current internal state, following the Pelican-MAC construction.

Finalization. Finally, the internal state is encrypted with the full AES using the master key K to generate the tag T.

Rekeying. The designers of ALE require that the master key is changed after processing 2^{48} bits (*i.e.* 2^{41} blocks).

Previous results. ALE was designed to thwart attacks against LEX [11,20] that use a pair of partially-colliding internal states to recover the key. Indeed, each AES call uses a different key, which prevents those attacks. Other attacks have been proposed against LEX, based on differential trails between two message injections [30,34]. We compare the previous attacks in Table 1. To make the results comparable, we assume that attacks with a low success rate are repeated until they succeed. For attacks using more than 2^{41} blocks of data, the master key will be rotated.

4.2 Internal Key Recovery

We describe a new attack against ALE, based on previous analysis of LEX. The key update of ALE was supposed to avoid these attacks, but since the update function has small cycles, there is a large probability that the key state is repeated, which makes the attack possible.

We analyze cycles of P in the same way as for mixFeed: four iterations of the 5-round key schedule are equivalent to the application in parallel of four 32-bit functions. The study of one of these functions gives us information about the cycle structure of the permutation P. The 32-bit function has a cycle of length $\ell = 4010800805 \approx 2^{31.9}$; therefore the permutation P admits many cycles of length $4 \times \ell \approx 2^{33.9}$ which are reached with probability $(\ell \times 2^{-32})^4 \approx 0.76$.

Previous attacks against LEX [11,12,20] are based on the search for a pair of internal states that partially collides, with two identical columns. This pattern can occur in odd or even round: we use columns 0 and 2 for odd rounds, and columns 1 and 3 for even rounds. The partial collision occurs with probability 2^{-64}, and 32 bits of the colliding state can be directly observed, due to the leak extractions. A candidate pair can be tested with complexity 2^{64} [12, Section 7.1], using the leak extraction of rounds before and after the collision; if it actually corresponds to a partial collision this reveals the internal state and key.

In the case of ALE, we perform a chosen plaintext attack: we choose a message M of 2^{41} blocks (the maximum length allowed by the ALE specification) which admits cycles of length $4 \times \ell$. With probability 0.76, the key cycles after $4 \times \ell \approx 2^{33.9}$ iterations of the permutation P. When this happens, we can split the message into $2^{33.9}$ sets of $2^{7.1}$ blocks encrypted under the same key. In each set we can construct $2^{13.2}$ pairs. In total, from one message M of 2^{41} blocks, we get on average $0.76 \times 2^{13.2} \times 2^{33.9} \approx 2^{46.7}$ pairs encrypted with the same key.

Unfortunately, the attack against LEX uses five consecutive AES rounds, but in ALE, the subkeys used in five consecutive rounds do not follow the exact AES key schedule. It is not possible to apply exactly the same attack on ALE, but we can use the tool developed by Bouillaguet, Derbez, and Fouque [10,12] in

order to find an attack in this setting. This tool found an attack that can test a candidate pair with time complexity 2^{72}, and a memory requirement of 2^{72}, for two different positions of the partial collision:

- when the collision occurs in round 4, the attack uses the leak of rounds 1, 2, 3, 4 and of round 1 of the next 4-round AES.
- when the collision occurs in round 1, the attack uses the leak of rounds 1 and 2, and of rounds 2, 3, 4 of the previous 4-round AES.

Starting with $2^{16.3}$ messages of length 2^{48} (encrypted under different master keys) we obtain $2^{16.3} \times 2^{13.2} \times 2^{33.9} \approx 2^{63.4}$ pairs, such that each pair uses the same key with probability 0.76. Each pair can be used twice, assuming a collision at round 1 or at round 4, so we have in total $2^{64.4}$ pairs to consider, and we expect one of them to actually collide ($0.76 \times 2^{64.4} \approx 2^{64}$). After filtering on 32 bits, we have $2^{32.4}$ candidate pairs to analyse, so that the time complexity is $2^{32.4} \times 2^{72} = 2^{104.4}$, and the data complexity is $2^{16.3} \times 2^{41} = 2^{57.3}$.

This attack recovers the internal state, and we can compute backwards the initial state $E_K(0)$ and the session key $\widetilde{Z} = E_K(N)$. We can also generate almost universal forgeries: when $E_K(0)$ and $\widetilde{Z}$ are known we can compute the internal state and ciphertext corresponding to an arbitrary message, and we can match the value of the final internal state (and hence the tag) by choosing one block of message or associated data appropriately.

5 Application to Impossible Differential Attacks

In 1999, Biham, Biryukov and Shamir introduced Impossible Differential attacks: a new cryptanalysis technique that they applied to Skipjack ([3]). This attack is based on the existence of an impossible differential, *i.e.* a differential occurring with probability 0. If a key guess leads to this differential, then it can be deduced that this guess was wrong. This allows to eliminate key candidates and thus to obtain an attack faster than exhaustive search. Impossible differentials have been applied to various cryptosystems, including reduced versions of AES [2,13,33].

The framework described in [13] is composed of two parts: firstly, combinations of bytes from the first and last subkeys are shown impossible, and secondly, the master keys associated to the remaining candidates are reconstructed and tested. When reconstructing the master key, previous attacks only exploit the subkeys bytes in the first rounds, guess the missing bytes, and evaluate the key schedule to check the bytes in the last subkeys. Our results significantly improve this part, by combining information from the first and the last subkeys. Indeed, the new representation shows that some bytes of a given subkey depend on fewer than 128 bits of information of another subkey, even if the subkeys are separated by many rounds. The complexity of the attack is a trade-off between the first and second parts. After improving the second part we obtain slightly better trade-offs. The improvement is limited because a small increase of the data complexity (corresponding to the cost of the part) leads to a large reduction in the number of remaining candidates (corresponding to the complexity of the second part).

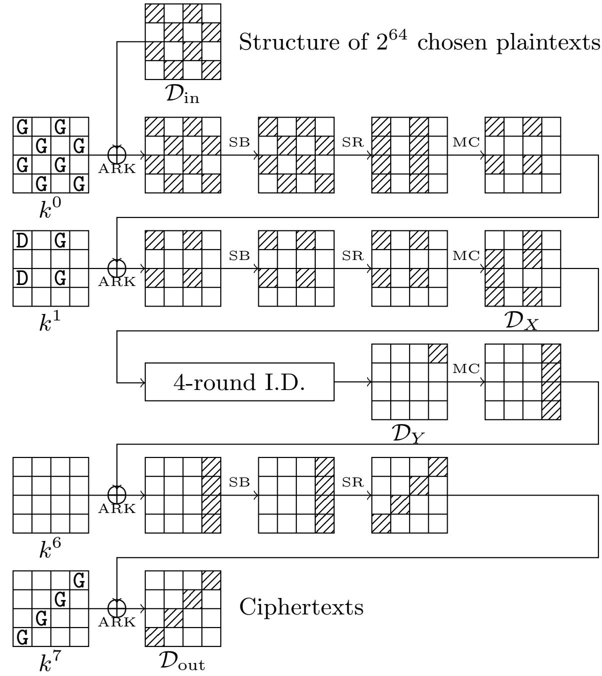

Fig. 10. 7-round impossible differential attack of [33] (figure adapted from [28]). Key bytes marked G and D are respectively guessed, and deduced from guessed bytes.

5.1 The AES round function

The AES state is represented as a 4×4-byte array, and the round function iterates the following operations:

- SubBytes applies an Sbox on each byte of the state;
- ShiftRows shifts by the left the second row of the state by 1 cell, the third row by 2 cells, and the last row by 3 cells;
- MixColumns multiplies each column of the state by an MDS matrix;
- AddRoundKey xors the state with the round key.

Sbox property. During this attack, we will use a well-known property for a n-bit to m-bit Sbox: given an input and an output difference, there is on average 2^{n-m} possible values. For the AES Sbox, $n = m = 8$, so in average one value is expected. We pre-compute those values, and refer to that table as the DDT.

5.2 Previous results

The best impossible differential attacks against AES-128 are variants of an attack from Mala, Dakhilalian, Rijmen and Modarres-Hashemi [33]. Several trade-off are proposed in [13] with four output differentials and using a technique to

reduce the memory by iterating over the possible key bytes values, rather that iterating over the data pairs. In this work, we start from a variant with a single output differential explained in detail below; it is easier to describe than variants considered in [13] and provides an interesting trade-off.

Impossible differential. This attack uses a collection of impossible differentials over 4 rounds, and extends them with two rounds at the beginning and one round at the end (omitting the final MixColumns), as shown in Figure 10. We use a set of impossible differentials over 4-rounds (without the last MixColumns):

$$\mathcal{D}_X \not\to \mathcal{D}_Y$$
$$\mathcal{D}_X = \begin{Bmatrix} (0,?,?,?,0,0,0,0,?,?,0,?,0,0,0,0) \\ (?,0,?,?,0,0,0,0,?,?,?,0,0,0,0,0) \\ (?,?,0,?,0,0,0,0,0,?,?,?,0,0,0,0) \\ (?,?,?,0,0,0,0,0,?,0,?,?,0,0,0,0) \end{Bmatrix}$$
$$\mathcal{D}_Y = \begin{Bmatrix} (0,0,0,0,0,0,0,0,0,0,0,0,x,0,0,0) \\ (0,0,0,0,0,0,0,0,0,0,0,0,0,x,0,0) \\ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,x,0) \\ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,x) \end{matrix} : x \neq 0 \right\}$$

We assume to be given a pair of plaintexts and the corresponding ciphertexts such that the plaintext difference is in a set $\mathcal{D}_{\text{in}}$ corresponding to two active diagonals, and the ciphertext difference is in a set $\mathcal{D}_{\text{out}}$ corresponding to one active anti-diagonal:

$$\mathcal{D}_{\text{in}} = \{(?,0,?,0,0,?,0,?,?,0,?,0,0,?,0,?)\}$$
$$\mathcal{D}_{\text{out}} = \{(0,0,0,?,0,0,?,0,0,?,0,0,?,0,0,0)\}$$

After guessing the values of the key bytes $k^0_{\langle 0,2,5,7,8,10,13,15\rangle}, k^1_{\langle 8,10\rangle}, k^7_{\langle 3,6,9,12\rangle}$, we can deduce that some values result in differences in $\mathcal{D}_X$ and $\mathcal{D}_Y$. Since this transition holds with probability 0, we can discard those key candidates. Eventually with a large number N of pairs of plaintexts, we eliminate most of the key candidates, and we can verify the remaining candidates exhaustively. We now detail how to perform this attack efficiently, following Algorithm 1.

Pre-computation. After the MixColumns of the first round, in column 1 and 3, we want non-zero differences only in the first and the third bytes. There are 2^{16} possible differences; by inverting the linear operations MixColumns and ShiftRows, we obtain 2^{16} possible differences for the diagonal (bytes $\langle 0,5,10,15\rangle$ and $\langle 2,7,8,13\rangle$ respectively) after the SubBytes of the first round. We store these 2^{16} differences in the table T_1. Similarly, we build a table T_2 with the 2^{10} possible differences before the SubBytes of the last round by propagating the 2^{10} differences in $\mathcal{D}_Y$.

Construction of pairs. We start with $2^{37+\epsilon}$ structures of 2^{64} plaintexts such that all the plaintexts in a structure are identical in bytes 1, 3, 5, 7, 9, 11, 13, and 15. For each set, we construct $\binom{2^{64}}{2} \approx 2^{127}$ pairs. We identify the pairs with a ciphertext difference in $\mathcal{D}_{out}$ and store them in a list L_1; we expect to have $N = 2^{127} \times 2^{-96} \times 2^{37+\epsilon} = 2^{68+\epsilon}$ pairs.

Step 1. First, we identify plaintext/ciphertext pairs and values of $k^0_{\langle 0,5,10,15 \rangle}$ that result in a zero difference in bytes 1 and 3 after the first MixColumns. To this end, we sort the list L_1 according to the plaintext difference and value in bytes 0, 5, 10 and 15. We obtain 2^{64} sublists of approximatively $2^{4+\epsilon}$ pairs. From now on, all the steps are repeated for all guesses of the key bytes $k^0_{\langle 0,5,10,15 \rangle}$. For each possible difference δ in bytes 0, 5, 10 and 15 before SubBytes, we confront the difference with each of the possible differences after SubBytes in T_1. Then, using the DDT of the AES Sbox, we extract the input values of the SubBytes operation of the first round, corresponding to this input and output difference. Since the key $k^0_{\langle 0,5,10,15 \rangle}$ has been guessed, we can deduce the value of the plaintext in bytes 0, 5, 10 and 15, and locate the right sublist of L_1 with $2^{4+\epsilon}$ pairs that follow this part of the trail for this key guess. We store those pairs in a list L_2; after iterating over δ and T_1 we have on average $2^{32+16+4+\epsilon} = 2^{52+\epsilon}$ pairs in L_2.

Step 2. During this step, we filter data pairs and values of $k^0_{\langle 2,7,8,13 \rangle}$ leading to a zero difference in bytes 13 and 15 after the first MixColumns. To do this, we consider each pair of L_2, and iterate over the possible differences after SubBytes in bytes 2, 7, 8, 13, stored in T_1. Since we have the input and output differences of those Sboxes, we retrieve the corresponding values from the DDT. By xoring these values with the plaintext, we obtain the associated key bytes $k^0_{\langle 2,7,8,13 \rangle}$ and we add this pair to a list indexed by the key bytes, $L_3[k^0_{\langle 2,7,8,13 \rangle}]$.

The following steps are repeated for each value of $k^0_{\langle 2,7,8,13 \rangle}$; we have a list $L_3[k^0_{\langle 2,7,8,13 \rangle}]$ of $2^{52+\epsilon+16-32} = 2^{36+\epsilon}$ plaintext pairs that satisfy the required difference after the first round.

Step 3. During this step, we associate each pair of $L_3[k^0_{\langle 2,7,8,13 \rangle}]$ to the key bytes k^1_8 and k^1_{10} such that difference after the MixColumns of round 2 is in $\mathcal{D}_X$. We recall that at this point, the bytes $k^0_{\langle 0,2,5,7,8,10,13,15 \rangle}$ have already been guessed. Following the AES-128 key schedule, we can easily deduce bytes k^1_0 and k^1_2. For each pair of $L_3[k^0_{\langle 2,7,8,13 \rangle}]$, we compute the values of the first and the third column of both plaintexts after the MixColumns of the first round. Using $k^1_{\langle 0,2 \rangle}$ We can also compute the values of both states on bytes 0 and 2 after AddRoundKey and SubBytes in the second round, corresponding to bytes 0 and 10 after ShiftRows. Looking at the MixColumns operations in columns 1 and 3 in the second round, we know the difference in 3 input bytes (2 zeros given by the differential trail, and value just recovered) and one output byte (a zero given by the differences in $\mathcal{D}_X$). Therefore we can recover the full input and output difference in those columns by solving a linear system (the solution is unique because of the MDS

property). By inverting the ShiftRows operation, we recover the difference after the SubBytes operation of the second round in bytes 8 and 10. The difference before this operation is also known, therefore we recover the values of bytes 8 and 10 before SubBytes, and deduce the value of $k^1_{\langle 8,10\rangle}$ by xoring the value at the end of the first round. We have to repeat this deduction four time, because we have four different positions of the zero differences in $\mathcal{D}_X$. Each pair of $L_3[k^0_{\langle 2,7,8,13\rangle}]$ suggests on average four candidates for $k^1_{\langle 8,10\rangle}$, and we store the pairs in a list indexed by the key bytes, $L_4[k^1_{\langle 8,10\rangle}]$.

The next steps are repeated for each value of $k^1_{\langle 8,10\rangle}$, using the list $L_4[k^1_{\langle 8,10\rangle}]$ with on average $2^{36+\epsilon+2-16} = 2^{22+\epsilon}$ pairs leading to a difference in $\mathcal{D}_X$.

Step 4. This step determines the key candidates $k^7_{\langle 3,6,9,12\rangle}$ that are ruled out with the available data, for each $k^0_{\langle 0,2,5,7,8,10,13,15\rangle}, k^1_{\langle 8,10\rangle}$. For this purpose, we use a list L_5 of 2^{32} bits to mark impossible key candidates $k^7_{\langle 3,6,9,12\rangle}$. For each pair of $L_4[k^1_{\langle 8,10\rangle}]$, we consider all the differences at the end of the sixth round that correspond to a difference in $\mathcal{D}_Y$, stored in T_2. From the differences before and after the last SubBytes, we compute the value of the output of SBox in bytes 3, 6, 9 and 12 using the DDT. Then, using the ciphertext values, we recover the bytes $k^7_{\langle 3,6,9,12\rangle}$ and mark this value in the list L_5.

On average we mark $2^{22+\epsilon+10} = 2^{32+\epsilon}$ keys as impossible, so that each key remains possible with probability $P = (1-2^{-32})^{2^{32+\epsilon}} \approx e^{-2^\epsilon}$.

Step 5. Finally, we reconstruct the master keys corresponding to the candidates $k^0_{\langle 0,2,5,7,8,10,13,15\rangle}, k^1_{\langle 8,10\rangle}, k^7_{\langle 3,6,9,12\rangle}$ not marked as impossible. Following [33,13], knowing $k^0_{\langle 0,2,5,7,8,10,13,15\rangle}$ and $k^1_{\langle 8,10\rangle}$ is equivalent to knowing $k^0_{\langle 0,2,4,5,6,7,8,10,13,15\rangle}$, but it is hard to combine this with information about the last round. Therefore, for each of the $2^{112} \times P$ candidates, we just consider the 10 known bytes of k^0, do an exhaustive search for the 6 missing bytes and recompute k^7 to see if it matches the candidate. This requires $2^{112} \times P \times 2^{48} = 2^{160} \times P$ evaluations of the key schedule. We verify the $2^{160} \times P \times 2^{-32} = 2^{128} \times P$ remaining candidates with a know plaintext/ciphertext pair, for a cost of $2^{128} \times P$ encryptions.

Complexity. There are three dominant terms in the complexity of the attack. First we need to make $2^{101+\epsilon}$ calls to the encryption oracle. Then, the generation of key candidates (steps 1 to 4) is dominated by step 4. This step is done 2^{80} times (for each guess of $k^0_{\langle 0,2,5,7,8,10,13,15\rangle}$ and $k^1_{\langle 8,10\rangle}$) and during this step we go through the whole list $L_4[k^1_{\langle 8,10\rangle}]$, containing $2^{22+\epsilon}$ pairs. For each pair and for each of the 2^{10} differences in T_2, we use 4 times the DDT. In order to express this complexity using one encryption as the unit, we follow the common practice of counting the number of table look-up. A 7 round AES encryption, requires 20×7 table lookups (including the Sboxes in the key schedule), therefore the cost of 4 DDT lookups is similar to $4/140 = 1/35$ encryptions. In total, the complexity of Step 4 is $2^{80} \times 2^{22+\epsilon} \times 2^{10}/35$. Finally step 5 requires the equivalent

Algorithm 1. Construction of possible key candidates (Steps 1 to 4)

Require: Tables T_1, T_2 and a list L_1 of $2^{68+\epsilon}$ pairs satisfying $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$.
  Sort L_1 according to the plaintext difference and value in bytes 0, 5, 10 and 15.
  Let $L_1[\delta][x]$ be the sub-list with difference δ and value x in those bytes.
  for all $k^0_{\langle 0,5,10,15\rangle}$ **do**
    $L_2 \leftarrow \varnothing$
    for all 32-bits difference δ **do**
      for all difference θ in T_1 **do** ▷ bytes $\langle 0, 5, 10, 15\rangle$
        Compute value(s) $x_{\langle 0,5,10,15\rangle}$ before first SubBytes from DDT.
        Add all pairs of $L_1[\delta][x_{\langle 0,5,10,15\rangle} \oplus k^0_{\langle 0,5,10,15\rangle}]$ to L_2.
    $L_3 \leftarrow \left[\varnothing, \textbf{for all } k^0_{\langle 2,7,8,13\rangle}\right]$
    for all pairs $((p, p'), (c, c'))$ in L_2 **do**
      for all difference θ in T_1 **do** ▷ bytes $\langle 2, 7, 8, 13\rangle$
        Compute value(s) $x_{\langle 2,7,8,13\rangle}$ before first SubBytes from DDT.
        Add pair to $L_3[x_{\langle 2,7,8,13\rangle} \oplus p_{\langle 2,7,8,13\rangle}]$.
    for all $k^0_{\langle 2,7,8,13\rangle}$ **do**
      $L_4 \leftarrow \left[\varnothing, \textbf{for all } k^1_{\langle 8,10\rangle}\right]$
      Compute $k^1_{\langle 0,2\rangle}$ using the AES key schedule.
      for i in $\{0, 1, 2, 3\}$ **do**
        for all pairs in $L_3[k^0_{\langle 2,7,8,13\rangle}]$ **do**
          Deduce $k^1_{\langle 8,10\rangle}$, assuming that diagonal i is inactive at end of round 2.
          Add pair to $L_4[k^1_{\langle 8,10\rangle}]$.
      for all $k^1_{\langle 8,10\rangle}$ **do**
        $L_5 \leftarrow \left[\textbf{True}, \textbf{for all } k^7_{\langle 3,6,9,12\rangle}\right]$
        for all pairs $((p, p'), (c, c'))$ in $L_4[k^1_{\langle 8,10\rangle}]$ **do**
          for all difference θ in T_2 **do** ▷ bytes $\langle 12, 13, 14, 15\rangle$
            Compute value(s) $x_{\langle 15,14,13,12\rangle}$ after last SubBytes from DDT.
            $L_5[x_{\langle 15,14,13,12\rangle} \oplus c_{\langle 3,6,9,12\rangle}] \leftarrow$ **False**.
        for all $k^7_{\langle 3,6,9,12\rangle}$ **do**
          if $L_5[k^7_{\langle 3,6,9,12\rangle}]$ **then**
            Check key candidate $k^0_{\langle 0,2,5,7,8,10,13,15\rangle}, k^1_{\langle 8,10\rangle}, k^7_{\langle 3,6,9,12\rangle}$.

of $e^{-2^\epsilon} \cdot 2^{160}/5 + e^{-2^\epsilon} \cdot 2^{128}$ encryptions, because the cost of the key schedule compared to an encryption[3] is $4/20 = 1/5$. In total, the time complexity is:

$$T = 2^{101+\epsilon} + 2^{112+\epsilon}/35 + e^{-2^\epsilon} \cdot (2^{160}/5 + 2^{128})$$

The best time complexity is obtained by taking $\epsilon = 5.1$, leading to a time complexity of $2^{112.1}$, a data complexity of $2^{106.1}$ chosen plaintexts, and a memory complexity of $N = 2^{73.1}$ words.

[3] This ratio is given as $2^{-3.6} \approx 1/12$ in [13], but we don't see how to achieve this result. In any case the impact on the total complexity is negligible because it is compensated by a very small change of ϵ.

Variant with multiple differentials. Boura, Lallemand, Naya-Plasencia and Suder describe [13] in a variant of this attack using multiple output differentials. More precisely, instead of using a fixed column for $\mathcal{D}_Y$ and a fixed anti-diagonal for $\mathcal{D}_{\text{out}}$, they consider the four possible columns for $\mathcal{D}_Y$ and the four corresponding anti-diagonal for $\mathcal{D}_{\text{out}}$. The attacks is essentially the same, but there are two important differences.

To construct the pairs, they start from only $2^{35+\epsilon}$ structures of 2^{64} plaintexts, but they obtain $2^{68+\epsilon}$ pairs matching $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$ when considering the four anti-diagonal in $\mathcal{D}_{out}$. Steps 1 to 3 of the attack are the same a given above, but in step 4 each pair can give information about different bytes of k^7, depending on which anti-diagonal is active in the ciphertext. For each choice of $k^0_{\langle 0,2,5,7,8,10,13,15\rangle}, k^1_{\langle 8,10\rangle}$, they build a list of possible values for each anti-diagonal of k^7, and each key value remains possible with probability $e^{-2^{\epsilon-2}}$ because one fourth of the data correspond to each diagonal. Finally, in step 5, they merge the 4 lists, for a cost of $2^{80} \times (e^{-2^{\epsilon-2}} \cdot 2^{32})^4 = e^{-2^{\epsilon}} \cdot 2^{208}$.

The total time complexity of this variant is:

$$T = 2^{99+\epsilon} + 2^{112+\epsilon}/35 + e^{-2^{\epsilon}} \cdot (2^{208}/5 + 2^{128})$$

The best time complexity is obtained by taking $\epsilon = 6.1$, leading to a time complexity of 2^{113}, a data complexity of $2^{105.1}$ chosen plaintexts, and a memory complexity of $N = 2^{74.1}$ words.

This attack is listed with a time complexity of $2^{106.88}$ with $\epsilon = 6$ in [13], but this seems to be a mistake. There are not enough details of this attack in [13] to verify where their attack would differ from our understanding, but we don't see how to avoid having $2^{112+\epsilon}$ iterations at step 4, when we are eliminating 112-bit keys. Applying the generic formula (7) from the same paper also gives a term $2^{112+\epsilon}/35$ in the complexity (written as $2^{k_A+k_B}\frac{N}{2^{c_{\text{in}}+c_{\text{out}}}} \cdot C'_E$ in [13]).

Variant with state-test technique. In [13], the authors describe in details a variant using four output differentials and the state-test technique. This allows them to reduce by one byte the number of key bytes to be guessed, but they must use smaller structures, and this increases the data complexity.

The attack requires $N = 2^{68+\epsilon}$ chosen plaintexts, with a time complexity of:

$$T = 2^{107+\epsilon} + 2^{104+\epsilon}/35 + e^{-2^{\epsilon}} \cdot (2^{200}/5 + 2^{128})$$

The optimal time complexity[4] is 2^{113} with $\epsilon = 6$.

5.3 Our improvement

We now explain how to improve the first attack using properties of the key schedule. We keep steps 1 to 4 as given in Algorithm 1, but we improve the reconstruction of the master key from bytes of the first and last round keys (Step 5). With

[4] In [13] they report the complexity as $2^{113.1}$ with $\epsilon = 6.1$.

this improvement, generating the key candidates is actually cheaper than verifying them with a known plaintext/ciphertext pair. We use the following property of the key schedule, in order to guess the missing key bytes of k^0 iteratively, and to efficiently verify whether they match the known bytes of k^7.

Proposition 1. *Let k_i^r a byte of an AES-128 subkey. If the byte is in the last column ($12 \leq i < 16$), then it depends on only 32 bits of information of the master key. If the byte is in the second or third column ($4 \leq i < 12$), then it depends on only 64 bits of information of the master key.*

Proof. Bytes in the last column correspond to basis vectors in the new representation, following Equation (1) (for instance $k_{12}^r = s_{12}^r$). Therefore they depend only on one 32-bit chunk at any given round (k_{12}^7 can be computed from $s_{\langle 0,1,2,3 \rangle}^0$).

Bytes in the second column correspond to the sum of two basis vector in the new representation (for instance $k_6^r = s_{14}^r \oplus s_4^r$). Since the two elements do not belong to the same chunk, the byte depends on two 32-bit chunks at any given round (k_6^7 can be computed from $s_{\langle 0,1,2,3,8,9,10,11 \rangle}^0$).

Similarly, bytes in the third column correspond to the sum of two basis vector in the new representation (for instance $k_9^r = s_{15}^r \oplus s_8^r$). Therefore they depend only on two 32-bit chunks at any given round (k_9^7 can be computed from $s_{\langle 0,1,2,3,12,13,14,15 \rangle}^0$).

Bytes in the first column correspond to the sum of four basis vector from four different chunks, therefore they depend on the full state in general (for instance $k_3^r = s_{13}^r \oplus s_{10}^r \oplus s_7^r \oplus s_0^r$). □

Initially we are given the values of $k_{\langle 0,2,4,5,6,7,8,10,13,15 \rangle}^0$ and $k_{\langle 3,6,9,12 \rangle}^7$. According to the property above, k_{12}^7 can be computed from k_{15}^0, $k_{14}^0 \oplus k_{10}^0 \oplus k_6^0 \oplus k_2^0$, $k_{13}^0 \oplus k_5^0$, $k_{12}^0 \oplus k_8^0$, k_{14}^0, and k_6^7 can be computed from k_{15}^0, $k_{14}^0 \oplus k_{10}^0 \oplus k_6^0 \oplus k_2^0$, $k_{13}^0 \oplus k_5^0$, $k_{12}^0 \oplus k_8^0$, k_{13}^0, $k_{12}^0 \oplus k_8^0 \oplus k_4^0 \oplus k_0^0$, $k_{15}^0 \oplus k_7^0$, $k_{14}^0 \oplus k_{10}^0$. Therefore we can verify their value after guessing $k_{\langle 12,14 \rangle}^0$.

At this point two chunks are completely known: $s_{\langle 0,1,2,3 \rangle}^0$ and $s_{\langle 8,9,10,11 \rangle}^0$ or equivalently $s_{\langle 12,13,14,15 \rangle}^7$ and $s_{\langle 4,5,6,7 \rangle}^7$. In particular, we can deduce the value of $k_{13}^7 = s_8^7 = s_{15}^7 \oplus k_9^7$, which can also be computed from $s_{\langle 12,13,14,15 \rangle}^0$, *i.e.* from k_{12}^0, $k_{15}^0 \oplus k_{11}^0 \oplus k_7^0 \oplus k_3^0$, $k_{14}^0 \oplus k_6^0$, $k_{13}^0 \oplus k_9^0$. Therefore, we only need to guess $k_{11}^0 \oplus k_3^0$ and k_9^0 to verify k_{13}^7.

Finally, we focus of the remaining 32-bit chunk, corresponding to $s_{\langle 4,5,6,7 \rangle}^0$ and $s_{\langle 0,1,2,3 \rangle}^7$. We already have the value of $s_4^0 = k_{14}^0$ and $s_6^0 = k_{12}^0 \oplus k_4^0$, and we can compute $s_0^7 = s_{10}^7 \oplus s_{13}^7 \oplus s_7^7 \oplus k_3^7$. Using a pre-computed table, we recover the 2^8 values of the chunk corresponding to those constraints.

Algorithm 2 describes the full process. The cost of this step is $e^{-2^\epsilon} \times 2^{128}/5$, where $1/5$ is the cost of computing the key schedule compared to a full encryption. Finally the total time complexity of our attack is:

$$T = 2^{101+\epsilon} + 2^{112+\epsilon}/35 + e^{-2^\epsilon} \cdot (2^{128}/5 + 2^{128})$$

Algorithm 2. Improved version of the key candidate checking (Step 5)

Require: A key candidate $k^0_{\langle 0,2,5,7,8,10,13,15\rangle}, k^1_{\langle 8,10\rangle}, k^7_{\langle 3,6,9,12\rangle}$.
  for all $k^0_{\langle 12,14\rangle}$ **do**
    Compute $s^7_{\langle 12,13,14,15\rangle}$ from $s^0_{\langle 0,1,2,3\rangle}$
    if $k^7_{12} = s^7_{12}$ **then**
      Compute $s^7_{\langle 4,5,6,7\rangle}$ from $s^0_{\langle 8,9,10,11\rangle}$
      if $k^7_6 = s^7_4 \oplus s^7_{14}$ **then**
        $T \leftarrow [\varnothing, \textbf{for all } k^7_{15}]$
        for all $k^0_{11}, k^0_1 \oplus k^0_9$ **do**
          Compute $s^7_{\langle 0,1,2,3\rangle}$ from $s^0_{\langle 4,5,6,7\rangle}$
          Add $(k^0_{11}, k^0_1 \oplus k^0_9)$ to $T[s^7_0]$
        for all $k^0_9, k^0_3 \oplus k^0_{11}$ **do**
          Compute $s^7_{\langle 8,9,10,11\rangle}$ from $s^0_{\langle 12,13,14,15\rangle}$
          if $k^7_9 = s^7_8 \oplus s^7_{15}$ **then**
            for all $(k^0_{11}, k^0_1 \oplus k^0_9)$ in $T[s^7_{13} \oplus s^7_{10} \oplus s^7_7 \oplus k^7_3]$ **do**
              Check the master key k^0 with a pair (p, c).

The best time complexity is obtained by taking $\epsilon = 3.9$ leading to a time complexity of $2^{110.9}$, a data complexity of $2^{104.9}$ chosen plaintext, and a memory complexity of $2^{71.9}$ words.

We remark that the improvement is only applicable when the last MixColumns is omitted. In general, it does not affect the complexity of attacks, because removing the last MixColumns defines an equivalent cipher up to a modification of the key schedule. However, when attacks exploit relations between the subkeys, the relations are simpler if the last MixColumns is omitted [22].

6 New Representations of the AES-192 and AES-256 Key Schedules

The same techniques can also be applied to other variants of AES: we apply the algorithm of Leander, Minaud and Rønjom [32] to extract invariant subspaces of the key schedule, and we use a change of variables corresponding to the subspaces to obtain a simplified representation.

AES-192. We find two invariant subpaces of dimension 12, and obtain a simplified representation with 2 independant chunks each acting on 12 bytes, as shown in Figure 11.

AES-256. We find four invariant subpaces of dimension 8, and obtain a simplified representation with 4 independant chunks each acting on 8 bytes, as shown in Figure 12.

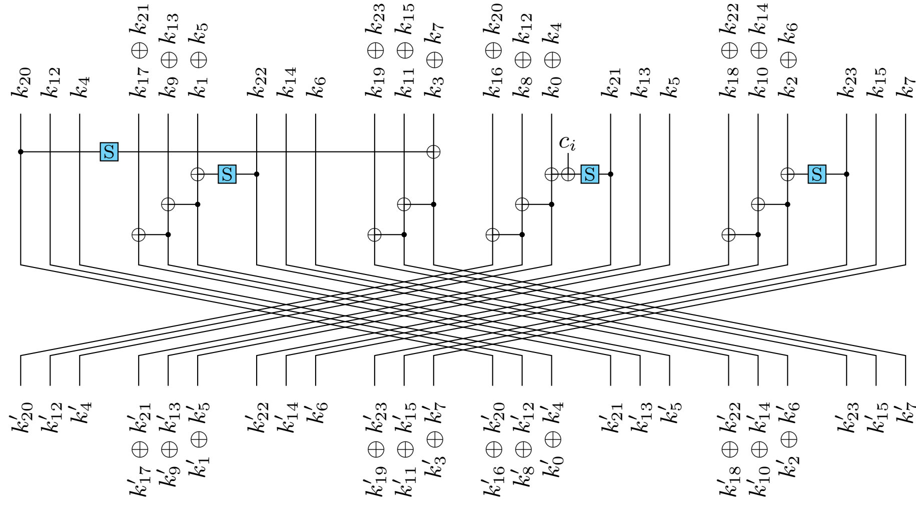

Fig. 11. One round of the AES-192 key schedule (alternative representation).

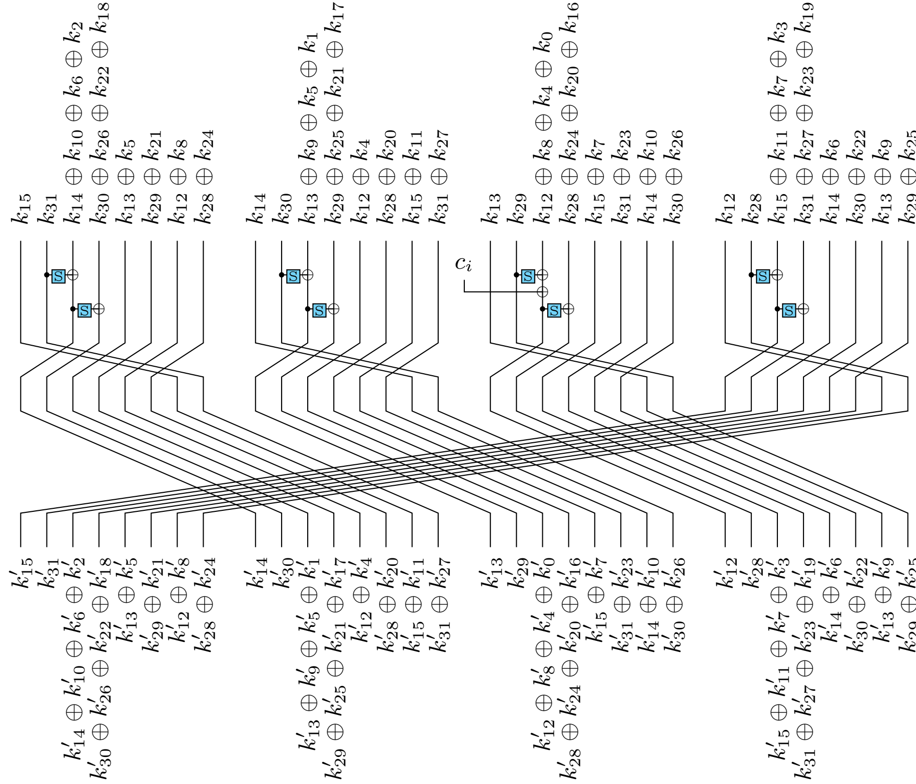

Fig. 12. One round of the AES-256 key schedule (alternative representation).

7 Properties on the AES Key Schedule

In addition to explaining the presence of short length cycles, our new representations of the key schedule also permits us to demonstrate some properties. For conciseness, we use the notation $k^r_{i,j_1 \oplus j_2,\ldots}$ to denote $k^r_i, k^r_{j_1} \oplus k^r_{j_2}, \ldots$

Proposition 2. *Let P_r and P'_r defined in one of the following ways:*

- *AES-128 (1): $P_r = k^r_{\langle 5,7,13,15\rangle}$, and $P'_r = k^r_{\langle 4,6,12,14\rangle}$*
- *AES-128 (2): $P_r = k^r_{\langle 0\oplus 4,2\oplus 6,8\oplus 12,10\oplus 14\rangle}$, and $P'_r = k^r_{\langle 1\oplus 5,3\oplus 7,9\oplus 13,11\oplus 15\rangle}$*
- *AES-192 (1): $P_r = k^r_{\langle 5,7,13,15,21,23\rangle}$, and $P'_r = k^r_{\langle 4,6,12,14,20,22\rangle}$*
- *AES-192 (2): $P_r = k^r_{\langle 0\oplus 4,2\oplus 6,8\oplus 12,10\oplus 14,16\oplus 20,18\oplus 22\rangle}$, and $P'_r = k^r_{\langle 1\oplus 5,3\oplus 7,9\oplus 13,11\oplus 15,17\oplus 21,19\oplus 23\rangle}$*
- *AES-256 (1): $P_r = k^r_{\langle 5,7,13,15,21,23,29,31\rangle}$, and $P'_r = k^r_{\langle 4,6,12,14,20,22,28,30\rangle}$*
- *AES-256 (2): $P_r = k^r_{\langle 0\oplus 4,2\oplus 6,8\oplus 12,10\oplus 14,16\oplus 20,18\oplus 22,24\oplus 28,26\oplus 30\rangle}$, and $P'_r = k^r_{\langle 1\oplus 5,3\oplus 7,9\oplus 13,11\oplus 15,17\oplus 21,19\oplus 23,25\oplus 29,27\oplus 31\rangle}$*

If there exists an r_0 such as P_{r_0} and $P'_{r_0\pm 1}$ are known, then for all $i \in \mathbb{Z}$, the bytes P_{r_0+2i} and P'_{r_0+2i+1} are known (and they are easily computable).

Proof. The AES-128 (1) case is considered here, the other cases are demonstrated in the same way. Knowing $k^r_{\langle 5,7,13,15\rangle}$ and $k^{r+1}_{\langle 4,6,12,14\rangle}$ is equivalent to knowing two chunks of the state: $s^r_{\langle 0,1,2,3\rangle}$ and $s^r_{\langle 8,9,10,11\rangle}$. This can be verified using Equation (2). The knowledge of these 2 chunks allows us to extract the value of the bytes in position $k_{\langle 5,7,13,15\rangle}$ or $k_{\langle 4,6,12,14\rangle}$ at any round. □

This byte position of this proposition is represented in figure 13. This proposition is a generalization of the observations made for AES-128 by Dunkelman and Keller:

Observation 3 ([21]) *For each $0 \le i \le 3$, the subkeys of AES satisfy the relations:*

$$k_{r+2}(i,0) \oplus k_{r+2}(i,2) = k_r(i,2).$$

$$k_{r+2}(i,1) \oplus k_{r+2}(i,3) = k_r(i,3).$$

Observation 4 ([21]) *For each $0 \le i \le 3$, the subkeys of AES satisfy the relation:*

$$k_{r+2}(i,1) \oplus SB(k_{r+1}((i+1) \ mod\ 4,3)) \oplus RCON_{r+2}(i) = k_r(i,1).$$

Another property can also be demonstrated on the AES-128 key schedule, using the value of one byte of the last column per round over 4 consecutive rounds:

Proposition 3. *If there exists $r \in \mathbb{N}$ and $i \in \{0,1,2,3\}$ such that the bytes $k^r_{15-i}, k^{r+1}_{15-(i+1)\%4}, k^{r+2}_{15-(i+2)\%4}, k^{r+3}_{15-(i+3)\%4}$ are known, then for all $j \in \mathbb{Z}$, the value of the byte $k^{r+j}_{15-(i+j\%4)}$ is known.*

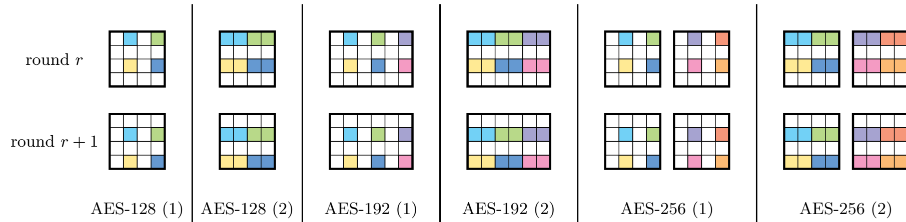

Fig. 13. Representation of the position of the bytes of the proposition. In variants (2), only the XOR of the two bytes of the same color must be known.

Proof. Knowing the bytes $k^r_{15-i}, k^{r+1}_{15-(i+1)\%4}, k^{r+2}_{15-(i+2)\%4}, k^{r+3}_{15-(i+3)\%4}$ is equivalent to knowing one chunk of the state in our representation: $s^r_{\langle 4i,4i+1,4i+2,4i+3\rangle}$. Given that $\forall r \in \mathbb{N}$, $s^r_{4i} = k^r_{15-i}$, we can calculate a byte of the last column at any round because we have the knowledge of a chunk in our new representation. □

The property can also be generalized when bytes at the correct position are known in non-consecutive rounds.

8 Conclusion

Alternative representations of the AES data operations have been used in several previous works; in particular, the super-box property [26] of Gilbert and Peyrin is an alternative representation of two AES rounds that led to several improved cryptanalysis results on AES-based schemes. Gilbert has later shown a more general untwisted representation of the AES data path, resulting in the first known-key attack against the full AES-128 [25].

In this work we use techniques from invariant subspace attacks to discover an equivalent representation of the AES key schedule, and we derive new cryptanalysis results, based on two main observations. First, iterating an odd number of key schedule rounds defines a permutation with short cycles. This undermine the security of AES-based schemes using iterations of the key schedule as a type of tweak to make each encryption call different. More generally, the AES key schedule cannot and should not be considered as a random permutation, even after a large number of rounds. Second, the alternative representation makes it easier to combine information from the first subkeys and from the last subkeys, improving previous key recovery attacks. This topic has been studied before and many attacks use key schedule relations to reduce the complexity (in particular, we can mention the *key bridging* notion of Dunkelman, Keller and Shamir [23,24]). However our alternative representation shows non-linear relations that have not been exploited before. In particular, we show that bytes in the last column of an AES-128 subkey depend on only 32 bits of information from the master key.

We expect that this alternative representation can open the way to further results exploiting properties of the AES key schedule. For instance, the new representation can be used to characterize keys that stay symmetric for two

rounds, as used in [27], but this is easily be done with the standard representation due to the small number of rounds.

Acknowledgement. The second author is funded by a grant from Région Ile-de-France. This work was also supported by the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE48-0017 (project SELECT).

References

1. Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce (Nov 2001)
2. Bahrak, B., Aref, M.R.: Impossible differential attack on seven-round AES-128. IET Inf. Secur. **2**(2), 28–32 (2008), https://doi.org/10.1049/iet-ifs:20070078
3. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (May 1999)
4. Biryukov, A.: The design of a stream cipher LEX. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 67–75. Springer, Heidelberg (Aug 2007)
5. Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., Shamir, A.: Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 299–319. Springer, Heidelberg (May / Jun 2010)
6. Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer, Heidelberg (Dec 2009)
7. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and related-key attack on the full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (Aug 2009)
8. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-based lightweight authenticated encryption. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 447–466. Springer, Heidelberg (Mar 2014)
9. Bonnetain, X., Naya-Plasencia, M., Schrottenloher, A.: Quantum security analysis of AES. IACR Trans. Symm. Cryptol. **2019**(2), 55–93 (2019)
10. Bouillaguet, C., Derbez, P.: AES attacks finder. https://github.com/cbouilla/AES-attacks-finder (2011)
11. Bouillaguet, C., Derbez, P., Fouque, P.A.: Automatic search of attacks on round-reduced AES and applications. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 169–187. Springer, Heidelberg (Aug 2011)
12. Bouillaguet, C., Derbez, P., Fouque, P.A.: Automatic search of attacks on round-reduced AES and applications. Cryptology ePrint Archive, Report 2012/069 (2012), http://eprint.iacr.org/2012/069
13. Boura, C., Lallemand, V., Naya-Plasencia, M., Suder, V.: Making the impossible possible. Journal of Cryptology **31**(1), 101–133 (Jan 2018)
14. Chakraborty, B., Nandi, M.: mixFeed. Submission to the NIST Lightweight Cryptography standardization process (2019), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/mixFeed-spec-round2.pdf
15. Chakraborty, B., Nandi, M.: Security proof of mixFeed (2019), https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2019/documents/papers/security-proof-of-mixfeed-lwc2019.pdf

16. Daemen, J., Rijmen, V.: The Pelican MAC function 2.0. Cryptology ePrint Archive, Report 2005/088 (2005), http://eprint.iacr.org/2005/088
17. Daemen, J., Rijmen, V.: The design of Rijndael: AES – the advanced encryption standard (2013)
18. Demirci, H., Selçuk, A.A.: A meet-in-the-middle attack on 8-round AES. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 116–126. Springer, Heidelberg (Feb 2008)
19. Derbez, P., Fouque, P.A., Jean, J.: Improved key recovery attacks on reduced-round AES in the single-key setting. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 371–387. Springer, Heidelberg (May 2013)
20. Dunkelman, O., Keller, N.: A new attack on the LEX stream cipher. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 539–556. Springer, Heidelberg (Dec 2008)
21. Dunkelman, O., Keller, N.: Treatment of the initial value in time-memory-data tradeoff attacks on stream ciphers. Inf. Process. Lett. **107**, 133–137 (08 2008)
22. Dunkelman, O., Keller, N.: The effects of the omission of last round's mixcolumns on AES. Inf. Process. Lett. **110**(8-9), 304–308 (2010)
23. Dunkelman, O., Keller, N., Shamir, A.: Improved single-key attacks on 8-round AES-192 and AES-256. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 158–176. Springer, Heidelberg (Dec 2010)
24. Dunkelman, O., Keller, N., Shamir, A.: Improved single-key attacks on 8-round AES-192 and AES-256. Journal of Cryptology **28**(3), 397–422 (Jul 2015)
25. Gilbert, H.: A simplified representation of AES. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 200–222. Springer, Heidelberg (Dec 2014)
26. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (Feb 2010)
27. Grassi, L., Leander, G., Rechberger, C., Tezcan, C., Wiemer, F.: Weak-key subspace trails and applications to AES. In: Jacobson Jr., M.J., Dunkelman, O., O'Flynn, C. (eds.) SAC 2020. LNCS, Springer, Heidelberg (Oct 2019)
28. Jean, J.: TikZ for Cryptographers. https://www.iacr.org/authors/tikz/ (2016)
29. Khairallah, M.: Weak keys in the rekeying paradigm: Application to COMET and mixFeed. IACR Trans. Symm. Cryptol. **2019**(4), 272–289 (2019)
30. Khovratovich, D., Rechberger, C.: The LOCAL attack: Cryptanalysis of the authenticated encryption scheme ALE. In: Lange, T., Lauter, K., Lisonek, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 174–184. Springer, Heidelberg (Aug 2014)
31. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A cryptanalysis of PRINTcipher: The invariant subspace attack. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 206–221. Springer, Heidelberg (Aug 2011)
32. Leander, G., Minaud, B., Rønjom, S.: A generic approach to invariant subspace attacks: Cryptanalysis of robin, iSCREAM and Zorro. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 254–283. Springer, Heidelberg (Apr 2015)
33. 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.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 282–291. Springer, Heidelberg (Dec 2010)
34. Wu, S., Wu, H., Huang, T., Wang, M., Wu, W.: Leaked-state-forgery attack against the authenticated encryption algorithm ALE. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 377–404. Springer, Heidelberg (Dec 2013)