New Representations of the AES Key Schedules

Clara Pernot

To cite this version:

Clara Pernot. New Representations of the AES Key Schedules. Cryptography and Security [cs.CR]. 2020. hal-03135597

HAL Id: hal-03135597

https://hal.inria.fr/hal-03135597

Submitted on 9 Feb 2021

New Representations of the AES Key Schedules

Application to mixFeed and ALE

université de BORDEAUX

Master Thesis

Master in *Sciences and Technologies*,
Specialty in *Mathematics*,
Track *Cryptology and Computer Security.*

Author: Clara Pernot
Supervisor: Gaëtan Leurent
Tutor: Gilles Zémor

Friday, 28th August 2020

Declaration of authorship of the document

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of the Master in *Sciences and Technologies*, Specialty in *Mathematics* or *Computer Science*, Track *Cryptology and Computer Security*, is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Date and Signature

28/08/2020

Pernot

Abstract

In this master thesis we present new representations of the AES key schedules, 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, up to linear transformation. Surprisingly, this property has not been described in the literature after more than 20 years of analysis of AES.

As a consequence, 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, completely breaking the scheme. The same observation also leads to a novel attack on ALE, another AES-based AEAD scheme.

Acknowledgements

First of all, I would like to thank Gaëtan Leurent for his involvement during my internship: I enjoyed the work we have done together during these 6 months, the results of which are beyond my initial expectations.

Even if this internship took place in an atypical way with only 2 months of presence out of the 6 planned, I was still able to get to know the members of the COSMIQ team. I would like to thank them for their warm welcome, and the pleasant atmosphere they give to the 2nd floor.

Because of these reasons, I am very happy to be able to continue as a PhD student under the supervision of Gaëtan and within this team.

As this master thesis is the conclusion of the 2 years of the master, I also take this opportunity to thank the teachers of the University of Bordeaux for their investments, and all the students for the 2 very good years spent during the CSI master.

Table of contents

Introduction

Cryptology is the science of secrecy and data protection: it ensures confidentiality, data integrity, authentication, and non-repudiation. It combines cryptography, which consists in building primitives, and cryptanalysis, which aims to attack them. Nowadays, cryptography splits into two parts. Symmetric or private-key cryptography, in which the secret information (encryption key) is known to both communicating parties. And asymmetric, or public-key cryptography, which requires the use of a key pair: a public key, which, as its name suggests, is known to all, and a private key, which is known only to the owner.

In symmetric cryptography, the main standards are published by the NIST (National Institute of Standards and Technology), and selected after an international competition. Between 1997 and 2000, NIST held a competition called Advanced Encryption Standard (AES) to select a new cryptographic standard, as the previous standard algorithm DES had become obsolete. The winner of this competition was the block cipher Rijndael [DR99, AES01], now called AES and it is still the most widely used block cipher today. The AES follows an iterative process described in Figure 1, which alternates XOR with subkeys and application of a round function. The key schedule allows to generate the subkeys from a master key, in an iterative way. After twenty years of cryptanalysis, many different attacks have been applied to the AES, and we have a strong confidence in its security. However, the key schedule is arguably the weakest part of the design, and it is known to cause issues in the related-key setting [BKN09, BK09, BDK+10].

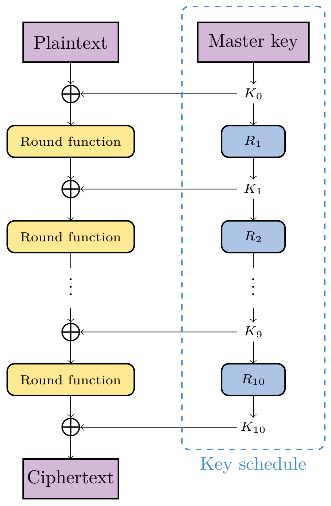


Figure 1: Description of the AES-128.

More recently, the NIST has proposed another competition to obtain lightweight encryption standards. Indeed, the emergence of new objects (IoT, contactless smart cards, ...) requiring the use of cryptographic algorithms in a constrained environment has led the community to think about new standards that meet these needs, since the usual encryption functions were designed to be used by computers and servers. Consequently, lightweight cryptography concerns symmetric-key encryption algorithms that are optimized for one or more of these parameters: RAM, ROM, time-area product, throughput, circuit area, power, latency, energy consumption... Within the scope of this competition, 57 cryptographic algorithms were submitted, 56 were selected during the first round. Currently, the second phase is underway and there are still 32 ciphers competing.

Out of the 32 candidates of the second round, we focus on mixFeed ([CN19a]), an AEAD (Authenticated Encryption with Associated Data) scheme, and especially on the observation concerning this cipher made by Khairallah [Kha19]: when the 11-round AES-128 key schedule is iterated there are apparently many short cycles of length roughly 2^{34}. Understanding the origin of this observation led us to show a surprising alternative representation of the AES-128 key schedule, where the key schedule is split into four independent parts, and the actual subkeys are just linear combinations of the four parts. Our new representation explains Khairallah's observation, and proves that his forgery attack against mixFeed actually succeeds with a very high probability, making it a practical break of the scheme.

ALE [BMR+14] (Authenticated Lightweight Encryption) is an AES-based lightweight authenticated encryption algorithm. An observation similar to the one on mixFeed can be made on ALE. With the new key schedule representation, the presence of a large number of short length cycles on key states in ALE can be shown. This allows us to find a new attack on ALE.

Organization. In section I), the new representation of the key schedule of AES-128 is depicted. In sections II) and III), attacks against mixFeed and ALE which exploit deductions related to the new representation are respectively presented. In section IV), the new representation is extended to 192 and 256 bits versions of the AES key schedule. In section V), a new property on the AES key schedule is demonstrated.

I) A New Representation of the AES-128 Key Schedule

A - 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 $0 \leq i \leq 3$. The following notations are used within the algorithm:

RotWord performs a cyclic permutation of one byte to the left.

SubWord applies the Sbox of AES 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 [AES01]. For simplicity, we denote x^{i-1} as c_i.

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

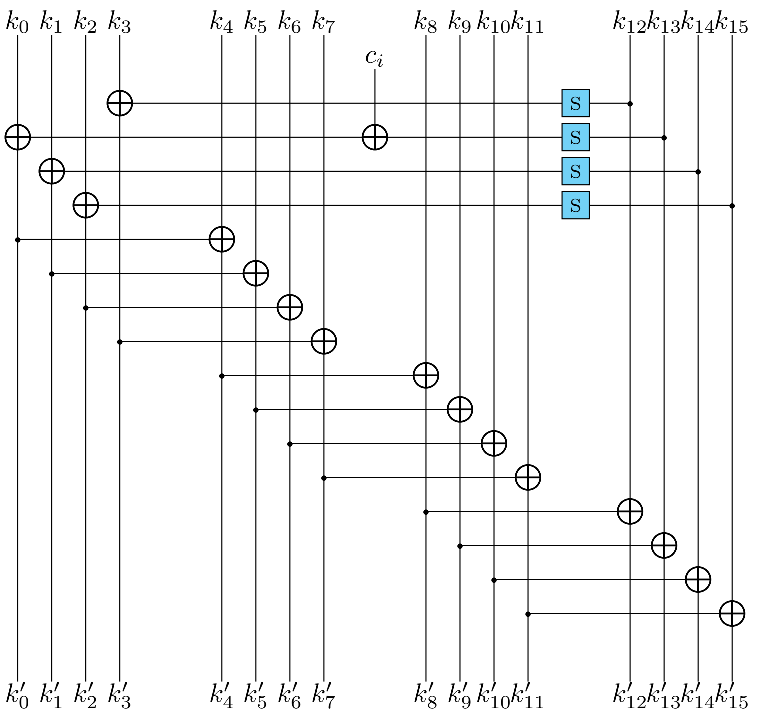
- if $i \equiv 0 \bmod 4$, $w_i = \mathsf{SubWord}(\mathsf{RotWord}(w_{i-1})) \oplus \mathsf{RCon}(i/4) \oplus w_{i-4}$.
- else, $w_i = w_{i-1} \oplus w_{i-4}$.

The subkey at round i is the concatenation of the words w_{4i} to w_{3+4i}. We can also express the key schedule at the byte level, using k_i with $0 \leq i < 16$ to denote the key-schedule state (with the AES byte ordering), and k'_i for the state after one round of key schedule. The key schedule can be written as follows, where S is the AES S-Box(See Figure 2):

$$\begin{aligned}
k'_0 &= k_0 \oplus S(k_{13}) \oplus c_i & k'_1 &= k_1 \oplus S(k_{14}) \\
k'_2 &= k_2 \oplus S(k_{15}) & k'_3 &= k_3 \oplus S(k_{12}) \\
k'_4 &= k_4 \oplus k_0 \oplus S(k_{13}) \oplus c_i & k'_5 &= k_5 \oplus k_1 \oplus S(k_{14}) \\
k'_6 &= k_6 \oplus k_2 \oplus S(k_{15}) & k'_7 &= k_7 \oplus k_3 \oplus S(k_{12}) \\
k'_8 &= k_8 \oplus k_4 \oplus k_0 \oplus S(k_{13}) \oplus c_i & k'_9 &= k_9 \oplus k_5 \oplus k_1 \oplus S(k_{14}) \\
k'_{10} &= k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{15}) & k'_{11} &= k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{12}) \\
k'_{12} &= k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{13}) \oplus c_i & k'_{13} &= k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{14}) \\
k'_{14} &= k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{15}) & k'_{15} &= k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{12})
\end{aligned} \tag{1}$$

B - The new representation of the AES-128 Key Schedule

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.* [LAAZ11]; the basic idea is to identify a linear subspace A and an offset u such that the round function F of a cipher satisfies $F(u + A) = F(u) + A$. At Eurocrypt 2015, Leander, Minaud and Rønjom [LMR15] 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.



k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14} k_{15}
c_i
S
S
S
S
k'_0 k'_1 k'_2 k'_3 k'_4 k'_5 k'_6 k'_7 k'_8 k'_9 k'_{10} k'_{11} k'_{12} k'_{13} k'_{14} k'_{15}

Figure 2: One round of the AES key schedule.

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

1. Guess an offset $u \in \mathbb{F}_2^n$ and a one-dimensional subspace A_0.
2. Compute $A_{i+1} = span\{(F(u + A_i) - F(u)) \cup A_i\}$.
3. If the dimension of A_{i+1} equals the dimension of A_i, we found an invariant subspace: $F(u + A) = F(u) + A$.
4. Else, we go on step 2.

In the case of the AES-128 key schedule, we use $\mathbb{F}_{2^8}^{16}$ instead of $\mathbb{F}_2^n$. If we apply this algorithm with the permutation F corresponding to 4 rounds of key schedule, with any key state u, and with A_0 the vector space generated by one of the first four bytes, we obtain 4 invariant affine subspaces whose linear parts are:

- $E_0 = \{(a, b, c, d, 0, b, 0, d, a, 0, 0, d, 0, 0, 0, d)$ with $a, b, c, d \in \mathbb{F}_{2^8}\}$
- $E_1 = \{(a, b, c, d, a, 0, c, 0, 0, 0, c, d, 0, 0, c, 0)$ with $a, b, c, d \in \mathbb{F}_{2^8}\}$
- $E_2 = \{(a, b, c, d, 0, b, 0, d, 0, b, c, 0, 0, b, 0, 0)$ with $a, b, c, d \in \mathbb{F}_{2^8}\}$
- $E_3 = \{(a, b, c, d, a, 0, c, 0, a, b, 0, 0, a, 0, 0, 0)$ with $a, b, c, d \in \mathbb{F}_{2^8}\}$

A 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, and we have parallel invariant subspaces for any offset u:

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

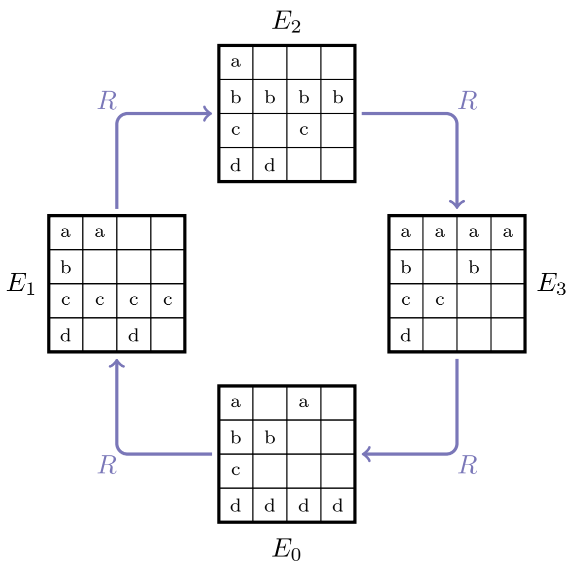


Figure 3: Evolution of the pattern of differences for an invariant subspace of dimension four.

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 $0 \le i < 4$ 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 full computation in four independent 32-bit computations:

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

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 $(1 \le i < 5)$:

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

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 is illustrated in the figure 3).

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 $(k_0, k_1, \ldots, k_{15})$ 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):

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

Therefore, we use:

$$\begin{array}{llll} s_0 = k_{15} & s_1 = k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 & s_2 = k_{13} \oplus k_5 & s_3 = k_{12} \oplus k_8 \\ s_4 = k_{14} & s_5 = k_{13} \oplus k_9 \oplus k_5 \oplus k_1 & s_6 = k_{12} \oplus k_4 & s_7 = k_{15} \oplus k_{11} \\ s_8 = k_{13} & s_9 = k_{12} \oplus k_8 \oplus k_4 \oplus k_0 & s_{10} = k_{15} \oplus k_7 & s_{11} = k_{14} \oplus k_{10} \\ s_{12} = k_{12} & s_{13} = k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 & s_{14} = k_{14} \oplus k_6 & s_{15} = k_{13} \oplus k_9 \end{array} \tag{2}$$

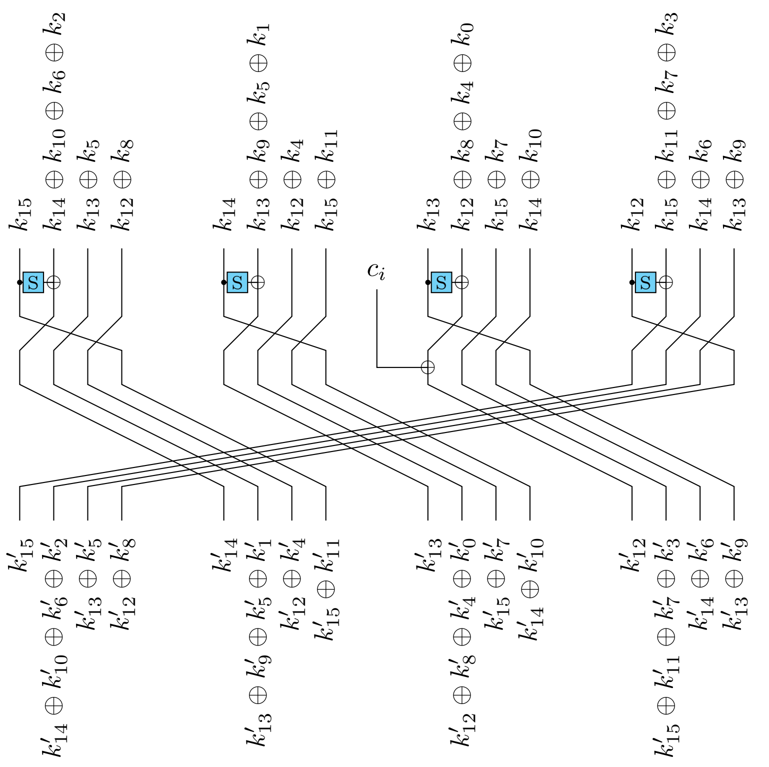


Figure 4: One round of the AES key schedule (alternative representation).

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

$$\begin{aligned}
s'_0 &= k'_{15} = k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{12}) && = s_{13} \oplus S(s_{12}) \\
s'_1 &= k'_{14} \oplus k'_{10} \oplus k'_6 \oplus k'_2 = k_{14} \oplus k_6 && = s_{14} \\
s'_2 &= k'_{13} \oplus k'_5 = k_{13} \oplus k_9 && = s_{15} \\
s'_3 &= k'_{12} \oplus k'_8 = k_{12} && = s_{12} \\
s'_4 &= k'_{14} = k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{15}) && = s_1 \oplus S(s_0) \\
s'_5 &= k'_{13} \oplus k'_9 \oplus k'_5 \oplus k'_1 = k_{13} \oplus k_5 && = s_2 \\
s'_6 &= k'_{12} \oplus k'_4 = k_{12} \oplus k_8 && = s_3 \\
s'_7 &= k'_{15} \oplus k'_{11} = k_{15} && = s_0 \\
s'_8 &= k'_{13} = k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{14}) && = s_5 \oplus S(s_4) \\
s'_9 &= k'_{12} \oplus k'_8 \oplus k'_4 \oplus k'_0 = k_{12} \oplus k_4 && = s_6 \\
s'_{10} &= k'_{15} \oplus k'_7 = k_{15} \oplus k_{11} && = s_7 \\
s'_{11} &= k'_{14} \oplus k'_{10} = k_{14} && = s_4 \\
s'_{12} &= k'_{12} = k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{13}) \oplus c_i && = s_9 \oplus S(s_8) \oplus c_i \\
s'_{13} &= k'_{15} \oplus k'_{11} \oplus k'_7 \oplus k'_3 = k_{15} \oplus k_7 && = s_{10} \\
s'_{14} &= k'_{14} \oplus k'_6 = k_{14} \oplus k_{10} && = s_{11} \\
s'_{15} &= k'_{13} \oplus k'_9 = k_{13} && = s_8
\end{aligned} \tag{3}$$

This is represented by Figure 4. 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:

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

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 rotation of 4 bytes to the right (see Appendix AES-128). In particular $C_0 = A^{-1}$.

In this new representation, there are clearly 4 independant parts each acting on 4 bytes, and the subkeys are reconstructed with linear combinations of the alternative key-schedule state.

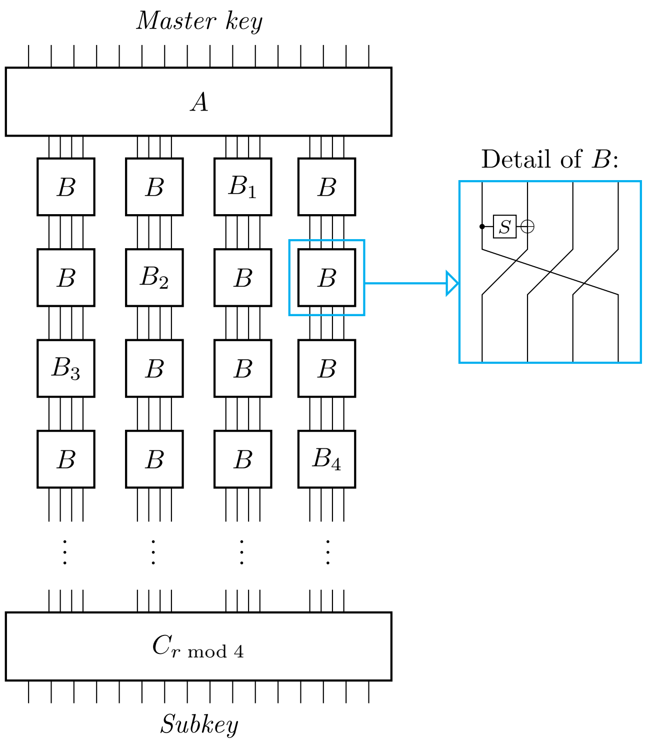


Figure 5: r rounds of the key schedule in the new representation. B_i is similar to B but the round constant c_i is XORed to the output of the S-box. The matrices of C_i for $0 \leq i \leq 3$ are given in Appendix.

II) Application to mixFeed

MixFeed ([CN19a]) is a second-round candidate in the NIST Lightweight Standardization Process, submitted by Bishwajit Chakraborty and Mridul Nandi, and based on the AES block cipher. However, mixFeed uses the AES in a peculiar way, with different keys for each AES call. More precisely, each key is computed by applying a permutation P to the previous key. In order to minimize the internal state, the permutation P just corresponds to eleven round of the AES key schedule, so the subkeys for all the AES calls just correspond to running the AES key schedule indefinitely.

In [Kha19], 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 section, we show that the new representation of the key schedule explains the existence of these short cycle, and we characterize the keys belonging to such cycles. This shows that the permutation P cannot be considered as a random permutation.

A - Description of mixFeed

A global scheme of mixFeed is given in Figure 7, and the full description can be found in [CN19a].

Notations: A, M and C are respectively associated data, plaintext and ciphertext. For the sake of simplicity, their lengths are considered to be multiples of 128 bits. So, A and M can respectively be decomposed into a and m blocks of 128 bits. We define $t = a + m$. For any string B, b_i denotes the i-th bit of B: $B = b_{|B|-1} \dots b_0$ (b_0 is the least significant bit). The chops functions are defined as follows : for $k \leq n$, $B \in 0, 1^n$, $\lfloor B \rfloor_k = B_{k-1} \dots B_0$, and $\lceil B \rceil_k = B_{n-1} \dots B_{n-k}$. When the index k is not specified, this means that $k = 64$.

The following functions are used in mixFeed:

- **E**: it is a modified version of the AES-128 in which the MixColumns of the last round is present.
- **P**: it is the permutation corresponding to eleven rounds of AES-128 key schedule.
- Feed: D represents either the block M, or the block A. Y is the input state, and X the output state. Feed is designed as follows:

  $$\mathsf{Feed}(Y, D) = (X, C)$$

  with $X = \lceil Y \oplus pad(D \oplus \lfloor Y \rfloor_{|D|} \rceil \parallel \lfloor Y \oplus pad(D) \rfloor$, and $C = D \oplus \lfloor Y \rfloor_{|D|}$.

  In the case where D is of length 128, the state and the ciphertext returned by Feed for the entry (Y, D) are respectively $\lceil D \rceil \parallel \lfloor D \oplus Y \rfloor$ and $D \oplus Y$ (see Figure 6).

We can distinguish 3 parts in the cipher mixFeed (each corresponding to a line in Figure 7):

Initialization of the key state and the data state from nonce N and master key K. We define $\widetilde{N} = N \parallel x$, with N a 120-bit nonce, and x a constant depending on the mode: authentication of A, encryption of M, both, or neither. $\widetilde{N}$ is encrypted under the master key K using E. The obtained ciphertext is our initial key state Z. Then, $N \| 0^8$ is encrypted under the key state Z using E. The obtained ciphertext is the data state.

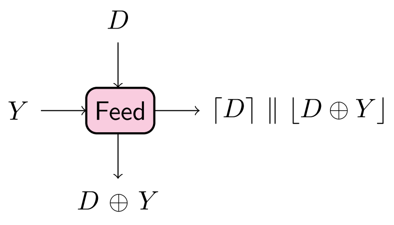


Figure 6: Function Feed in the case where $|D| = 128$

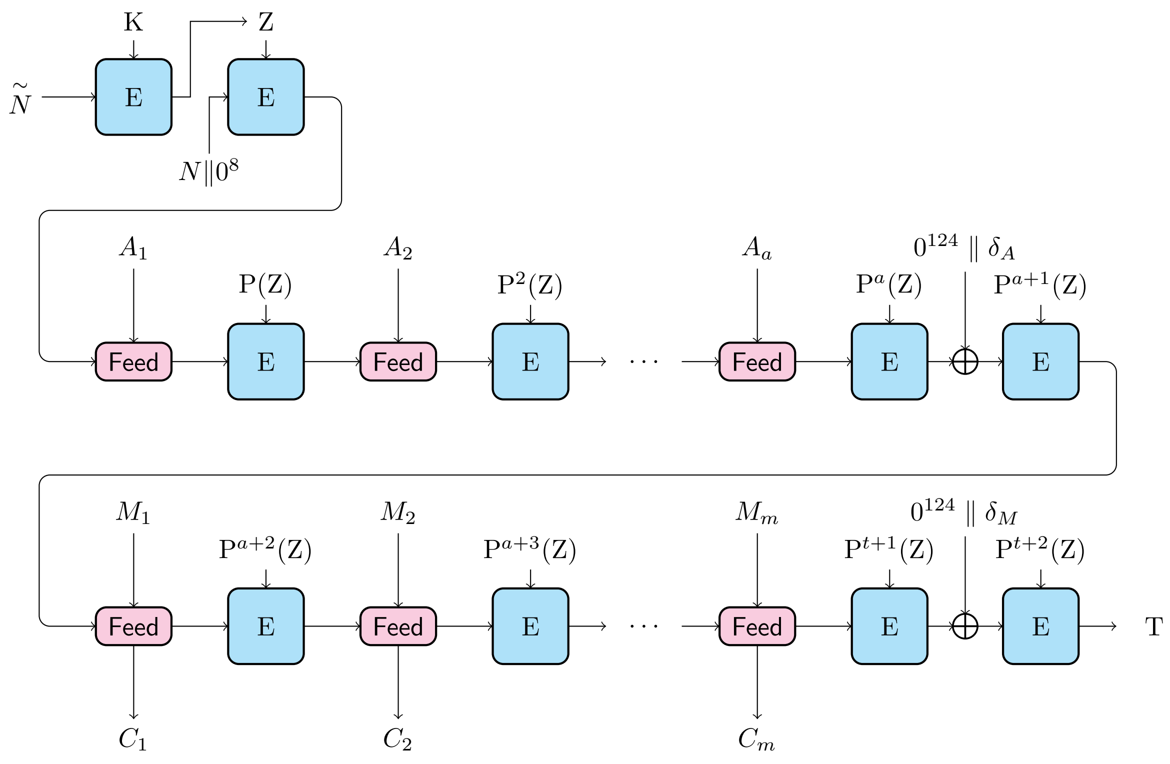


Figure 7: Authenticated encryption with mixFeed.

Authentication of the associated data A. For i from 1 to a, the Feed function is applied with the current data state as input state, and with the data block A_i. The output state of Feed is encrypted under the key $P^i(Z)$ using E. At the end of this step, a XOR is done between the current data state and $0^{124}\|\delta_A$, with δ_A a 4-bits constants defined according to the completeness of the last blocks of A. Then, this state in encrypted under the key $P^{a+1}(Z)$ using E.

Encryption and authentication of the message M. For i from 1 to m, the Feed function is applied with the current data state as input state, and with the data block M_i. Feed returns the cipher block C_i, and an output state which is then encrypted under the key $P^{a+1+i}(Z)$ using E. At the end of this step, a XOR is done between the current data state and $0^{124}\|\delta_M$, with δ_A a 4-bits constants defined according to the completeness of the last blocks of M. Then, this state in encrypted under the key $P^{t+2}(Z)$ using E: the obtained ciphertext is the authentication tag T.

B - Short Cycles of P

In [Kha19], 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 [CN19b], under the assumption that the number of keys in a short cycle is sufficiently small. More precisely, they wrote:

Assumption 1 ([CN19b])**.** *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 [CN19b], and allows a practical forgery attack.

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

- Linear transform A
- 4 parallel 32-bit functions that we denote $f_1 \| f_2 \| f_3 \| f_4$, with

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

  (the functions differ only on the position and the value of the round constants)

- Linear transform $C_3 = A^{-1} \times \mathsf{SR}^{-1}$

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).

Indeed, 4 iterations of $\widetilde{P}$ can be decomposed into 4 parallel functions, because of to the left rotation induced by SR^{-1} (see Figure B -):

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

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

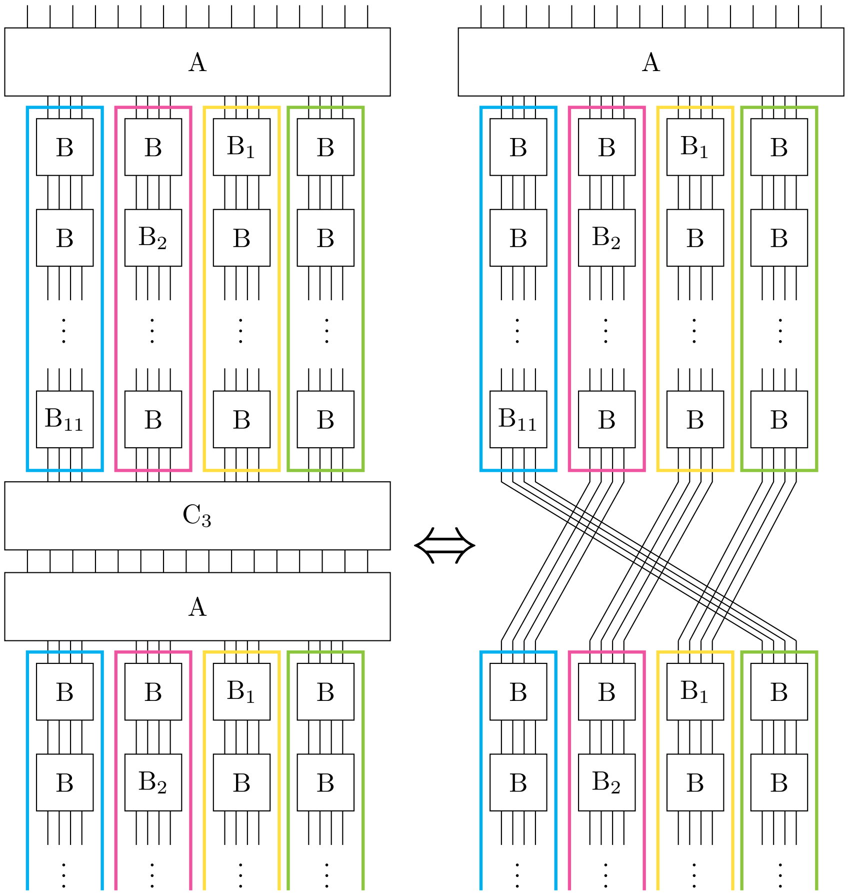


Figure 8: Two iterations of 11 rounds of the key schedule in the new representation.

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 $\phi_1(a) = f_2 \circ f_3 \circ f_4 \circ f_1(a)$, as shown in Table 1. The largest cycle has length $\ell = 3504665256$. Consequently, with probability $(3504665256 \times 2^{-32})^4 \simeq 0.44$, we have a, b, c and d in a

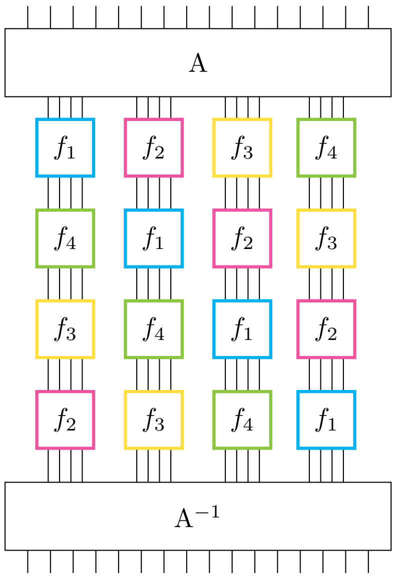


Figure 9: 4 iterations of P in the new model.

Table 1: Cycle structure of ϕ_1 for 11-round AES-128 key schedule

Length	# cycles	Proba	Smallest element
3504665256	1	0.82	`00 00 00 01`
255703222	1	0.05	`00 00 00 0b`
219107352	1	0.05	`00 00 00 1d`
174977807	1	0.04	`00 00 00 00`
99678312	1	0.02	`00 00 00 21`
13792740	1	0.003	`00 00 00 75`
8820469	1	$2^{-8,93}$	`00 00 00 24`
7619847	1	$2^{-9,14}$	`00 00 00 c1`
5442633	1	$2^{-9,63}$	`00 00 02 78`
4214934	1	2^{-10}	`00 00 05 77`
459548	1	$2^{-13,2}$	`00 00 38 fe`
444656	1	$2^{-13,24}$	`00 00 0b 68`
14977	1	$2^{-18,13}$	`00 06 82 5c`
14559	1	$2^{-18,18}$	`00 04 fa b1`
5165	1	$2^{-19,67}$	`00 0a d4 4e`
4347	1	$2^{-19,92}$	`00 04 94 3a`
1091	1	$2^{-21.91}$	`00 21 4b 3b`
317	1	$2^{-23,7}$	`00 28 41 36`
27	1	$2^{-27,25}$	`01 3a 0d 0c`
6	1	$2^{-29,42}$	`06 23 25 51`
5	3	$3 \cdot 2^{-29,68}$	`06 1a ea 18`
4	2	$2 \cdot 2^{-30}$	`23 c6 6f 2b`
2	3	$3 \cdot 2^{-31}$	`69 ea 63 75`
1	2	$2 \cdot 2^{-32}$	`7e be d1 92`

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 [Kha19], and clearly contradicts the assumption of [CN19b].

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 $\ell = LCM(\ell_1, \ell_2, \ell_3, \ell_4)$, and we can compute the associated probability from Table 1. 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 $f_2 \circ f_3 \circ f_4 \circ f_1(a) = a$, if we apply $\widetilde{P}$ to (a, b, c, d), we obtain: $(f_2(b), f_3(c), f_4(d), f_1(a)) = (a, b, c, d)$. Since $\widetilde{P} = A \circ P \circ A^{-1}$, the corresponding key in the original representation is:

$$A^{-1} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix}
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0\\
0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&1\\
0&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0\\
1&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix} \times \begin{pmatrix} \texttt{7e}\\ \texttt{be}\\ \texttt{d1}\\ \texttt{92}\\ \texttt{de}\\ \texttt{d4}\\ \texttt{b7}\\ \texttt{cc}\\ \texttt{9f}\\ \texttt{95}\\ \texttt{88}\\ \texttt{26}\\ \texttt{d4}\\ \texttt{b9}\\ \texttt{79}\\ \texttt{91} \end{pmatrix} = \begin{pmatrix} \texttt{64}\\ \texttt{0b}\\ \texttt{3f}\\ \texttt{83}\\ \texttt{63}\\ \texttt{4e}\\ \texttt{a7}\\ \texttt{f6}\\ \texttt{46}\\ \texttt{0e}\\ \texttt{f8}\\ \texttt{b2}\\ \texttt{d4}\\ \texttt{9f}\\ \texttt{de}\\ \texttt{7e} \end{pmatrix}$$

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 $\ell = 1 \bmod 4$:

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

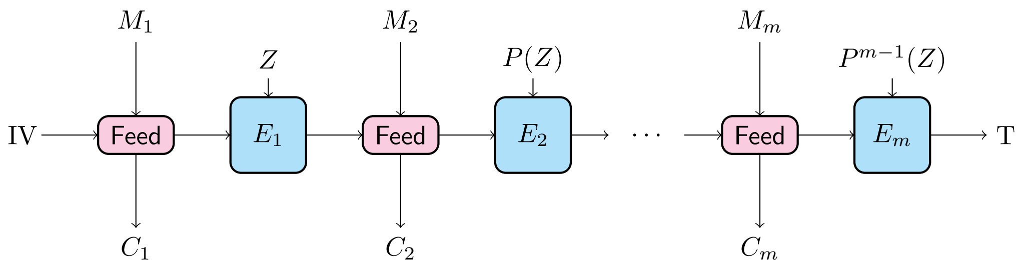


Figure 10: Simplified scheme of mixFeed encryption.

- if $\ell = 3 \bmod 4$:

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

C - Forgery attack against mixFeed

This attack was proposed by Mustafa Khairallah in [Kha19]. The goal of a forgery attack is to forge a valid tag for a new ciphertext using the ciphertext and the tag of a choosen plaintext. Let's consider the simplified scheme for the encryption presented in figure 10. The input and output of the encryption function E_r are respectively represented by $S_i[r]$ and $S_o[r]$. E_r is an AES encryption with key $P^{r-1}(Z)$.

Assuming that Z belongs to a cycle of length ℓ, we have the following attack considering a message M made of m blocks, with $m > \ell$:

1. Encrypt the message M, and obtain the corresponding ciphertext C and tag T.
2. Calculate $S_o[0] = IV$ and $S_i[\ell + 1]$ using M_r and C_r for $r = 1$ and $r = \ell + 1$.
3. Choose M_x and C_x such that $(S_i[\ell + 1], \mathrm{C}_x) = \mathsf{Feed}\ (S_o[0], M_x)$.
4. The T tag will also authenticate the new ciphertext $C' = C_x \| C_{\ell+2} \| \cdots \| C_m$.

The computations required for the forge are negligible: following the definition of the Feed function (see Figure 6), Step 2 requires only to an XOR of 128-bits blocks and Step 3 consists of 2 XORs of 64-bits blocks. 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 key cycle. The probability of success is approximately 0.44, using the cycles of length $\ell = 14018661024$ described above. Knowing that the success or not of the attack is based on the fact that Z belongs to a short cycle, for a fixed key, one can vary the nonce N and repeat the attack until success.

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 the theoretical reasoning.

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

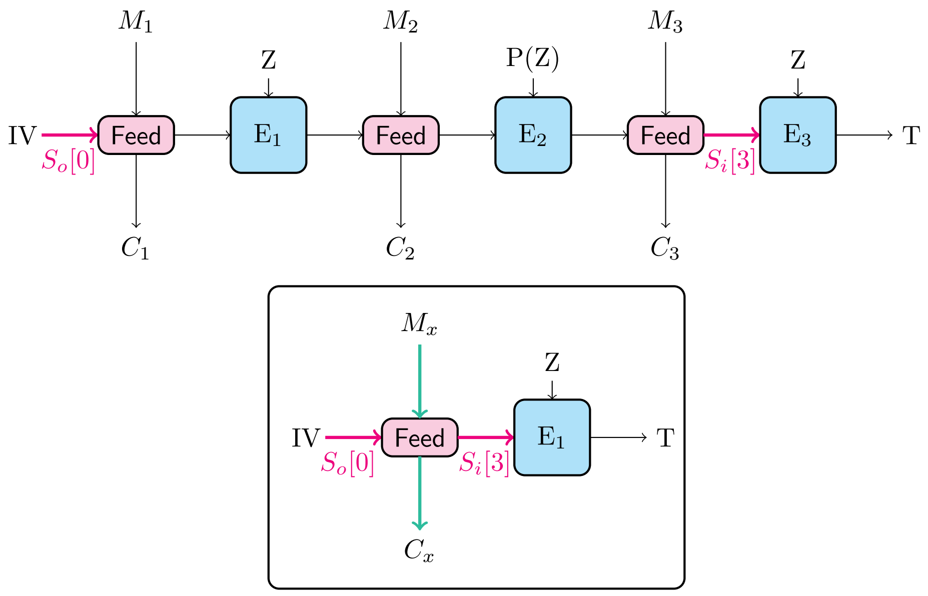


Figure 11: Forgery attack when Z belongs to a cycle of length 2.

III) Application to ALE

ALE [BMR+14] is an earlier authenticated encryption scheme based on the AES round function, strongly inspired by LEX [Bir07] (for the encryption part) and Pelican-MAC [DR05] (for the authentication part). Attacks have already been presented against ALE [KR14, WWH+13] but the new representation of the key schedule gives new types of attacks, based on previous attacks against LEX [DK08a, BDF11].

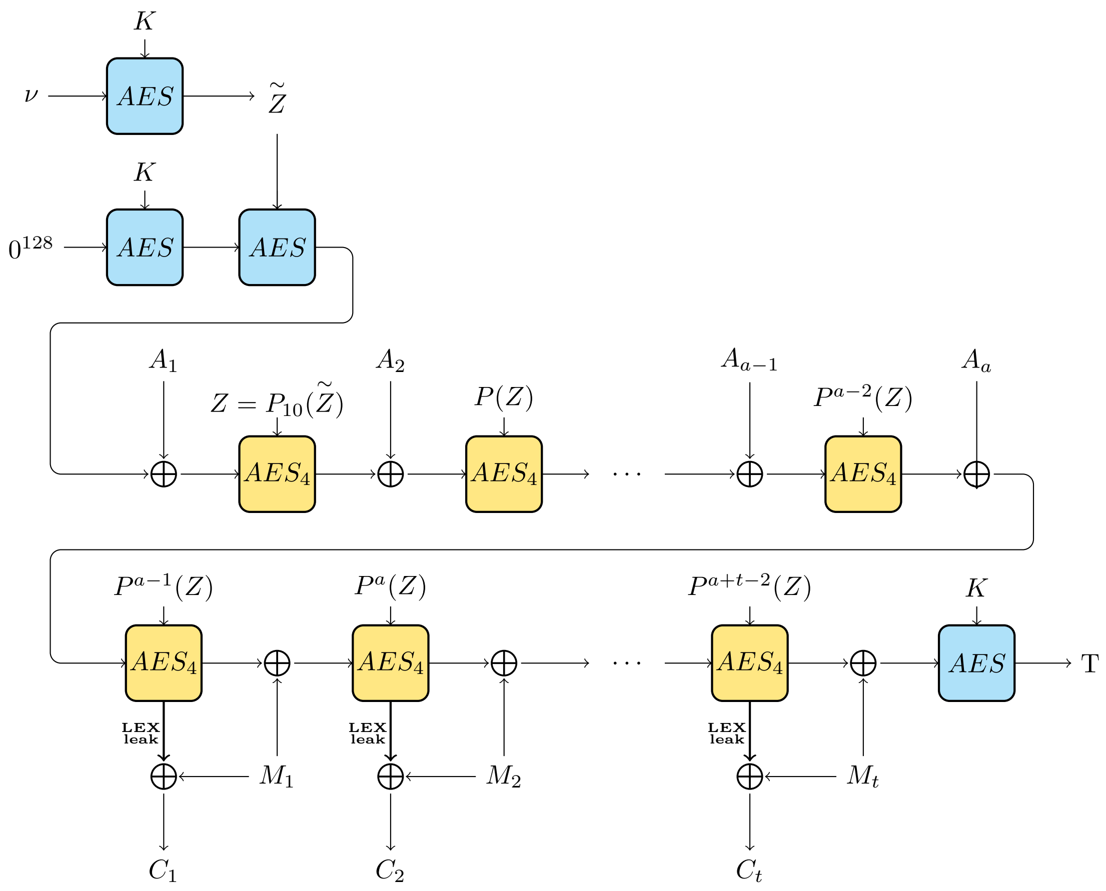


Figure 12: Authenticated encryption with ALE.

A - Description of ALE

For the sake of simplicity, we will only consider blocks of 16 bytes for both associated data and plaintext. This allows us to ignore the padding. ALE can be decomposed into 4 steps.

Initialization of the key state and the data state. A 128-bits nonce ν is encrypted under the master key K using AES-128. The obtained ciphertext is used as the initial key state. In order to intialize the data state, we first encrypt the message 0^{128} under the master key K using AES-128, and then the output is encrypted once again using AES-128, but this time under the intial key state previously obtained. The key state is updated by applying the round key schedule of AES-128 to the final round key of last AES encryption with round constant x^{10} in $\mathbb{F}_{2^8}$.

Associated data phase. The first block of associated data is xored to the current data state. Then, for each of the other data blocks, we do the following operations. We modify the data state by replacing it with the encryption of the previous state using four rounds of AES, and with the state key as key. And, in order to get the new key state, the final round subkey is udpated one more time using AES round key schedule with round constant x^4 in $\mathbb{F}_{2^8}$. Finally, the next block of associated data is xored to the current data state.

Message processing phase. For each block of M, the data state is replaced by the encryption of the previous data state using four rounds of AES, and with the key state as key. 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. The bytes are leaked at the end of each round (after the AddRoundKey operation). The ciphertext block is the XOR of the leak and the message block. The message block is also xored to the current data state. In order to get the new key state, the final round subkey is updated one more time using AES round key schedule with round constant x^4 in $\mathbb{F}_{2^8}$.

Finalization. The data state is encrypted with the full AES-128 using the master key K. The output of this encryption is returned as the authentication tag T for the message and the associated data.

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

As for mixFeed, the elements of the key state can be seen as the successive applications of a permutation P. In the case of ALE, the P permutation corresponds to 5 rounds of the AES key schedule. Five is also an odd number, so the same phenomenon occurs as for mixFeed: the permutation has many short cycles.

B - Cycle Analysis of 5-round AES-128 Key Schedule

We proceed 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. As seen in Table 2, the 32-bit function has a cycle of length 4010800805; therefore the permutation P admits many cycles of length $4 \times 4010800805 = 16043203220$ which are reached with probability $(4010800805 \times 2^{-32})^4 \simeq 0.76$.

Previous results. ALE was designed to thwart attacks against LEX [DK08a, BDF11] 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 [KR14, WWH+13]. We compare the previous attacks in Table 3. When an attack has a low success rate, we assume it is repeated until it succeeds. For attacks using more than 2^{41} blocks of data, the master key will be rotated.

Table 2: Cycle structure of ϕ_1 for 5-round AES-128 key schedule

Length	# cycles	Proba	Smallest element
4010800805	1	0.93	`00 00 00 00`
131787964	1	0.03	`00 00 00 5d`
49935997	1	0.01	`00 00 00 0e`
34379325	1	0.008	`00 00 00 1d`
33741892	1	0.008	`00 00 00 1e`
14932111	1	0.003	`00 00 01 94`
9654619	1	0.002	`00 00 01 3d`
6188177	1	$2^{-9.44}$	`00 00 07 28`
3087025	1	$2^{-10.44}$	`00 00 02 8a`
117032	1	$2^{-15.16}$	`00 00 63 a3`
110859	1	$2^{-15.25}$	`00 01 21 ca`
74232	1	$2^{-15.82}$	`00 00 a6 8e`
57337	1	$2^{-16.2}$	`00 01 1e 11`
33273	1	$2^{-16.98}$	`00 03 9f 7b`
23808	1	$2^{-17.46}$	`00 02 2d 14`
17227	1	$2^{-17.93}$	`00 01 ab 12`
8853	1	$2^{-18.89}$	`00 08 41 42`
6025	1	$2^{-19.44}$	`00 05 d7 2c`
5042	1	$2^{-19.70}$	`00 05 21 3a`
2516	1	$2^{-20.70}$	`00 1d d2 74`
1920	1	$2^{-21.10}$	`00 3f 0e 58`
906	1	$2^{-22.18}$	`00 22 52 0d`
179	1	$2^{-24.52}$	`01 59 63 a1`
168	1	$2^{-24.61}$	`00 66 2a fd`
3	1	$2^{-30.42}$	`3f 37 c5 3c`
1	1	2^{-32}	`7f 22 aa a7`

Table 3: Comparison of attacks against ALE

Attack		Enc	Verif	Time	Ref
Existential Forgery	Known Plaintext	$2^{110.4}$	2^{102}	$2^{110.4}$	[WWH+13]
Existential Forgery	Known Plaintext	2^{102}	2^{102}	2^{103}	[KR14]
Existential Forgery	Known Plaintext	1	2^{119}	2^{119}	[KR14]
Universal Forgery	Known Plaintext	1	2^{120}	2^{120}	[KR14]
Universal Forgery	Chosen Plaintext	$2^{61.3}$	1	$2^{104.4}$	New

C - Key and State Recovery Attack against ALE

We describe a new attack against ALE, based on the previous analysis of LEX [DK08a, BDF11]. 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.

Previous attacks against LEX 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.

ALE encrypts a plaintext with up to 2^{41} blocks of 16 bytes. We perform a chosen plaintext attack: we choose a message M of 2^{41} blocks which admits cycles of length $4010800805 \times 4 \approx 2^{33.9}$. With probability 0.76, the key cycles after $4010800805 \times 4 \approx 2^{33.9}$ iterations of the permutation P (five rounds of AES-128 key schedule). 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 5 consecutive AES rounds, therefore it is not possible to apply exactly the same attack as on ALE. Consequently, we used the tool developed by Bouillaguet, Derbez, and Fouque [BDF11] in order to find an attack against ALE. This tool found an attack 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 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 $2^{32.4}$ pairs to analyse, so that the time complexity is $2^{32.4} \times 2^{72} = 2^{104.4}$, and the data complexity is $2^{16.3} \times 2^{45} = 2^{61.3}$.

IV) New Representations of the AES-192 and AES-256 Key Schedules

The same techniques can also be applied to other variants of AES.

A - The AES-192 Key Schedule

The AES-192 key schedule allows us to derive 13 subkeys from a 192-bit master key. The operations used are the same as in the AES-128: $\mathsf{RotWord}$, $\mathsf{SubWord}$ and RCon. However, in the 192-bit version, the initialization is different: six 32-bit words w_i $(0 \leq i < 6)$ are filled with the bytes of the master key.

The way of constructing the words w_i $(i \geq 6)$ also differs:

- if $i \equiv 0 \bmod 6$, $w_i = \mathsf{SubWord}(\mathsf{RotWord}(w_{i-1})) \oplus \mathsf{RCon}(i/6) \oplus w_{i-6}$.
- else, $w_i = w_{i-1} \oplus w_{i-6}$.

The subkey at round i is the concatenation of the words w_{4i} to w_{3+4i}. In AES-192, the key schedule state corresponds to one and a half consecutive subkeys. As for the AES-128, we can also express the key schedule at byte level using k_i $(0 \leq i < 24)$ to denote the key-schedule state, and k'_i for the state after one round of key schedule. This corresponds to the following equations:

$$\begin{aligned}
k'_0 &= k_0 \oplus S(k_{21}) \oplus c_i & k'_1 &= k_1 \oplus S(k_{22}) \\
k'_2 &= k_2 \oplus S(k_{23}) & k'_3 &= k_3 \oplus S(k_{20}) \\
k'_4 &= k_4 \oplus k_0 \oplus S(k_{21}) \oplus c_i & k'_5 &= k_5 \oplus k_1 \oplus S(k_{22}) \\
k'_6 &= k_6 \oplus k_2 \oplus S(k_{23}) & k'_7 &= k_7 \oplus k_3 \oplus S(k_{20}) \\
k'_8 &= k_8 \oplus k_4 \oplus k_0 \oplus S(k_{21}) \oplus c_i & k'_9 &= k_9 \oplus k_5 \oplus k_1 \oplus S(k_{22}) \\
k'_{10} &= k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{23}) & k'_{11} &= k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{20}) \\
k'_{12} &= k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{21}) \oplus c_i & k'_{13} &= k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{22}) \\
k'_{14} &= k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{23}) & k'_{15} &= k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{20}) \\
k'_{16} &= k_{16} \oplus k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{21}) \oplus c_i & k'_{17} &= k_{17} \oplus k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{22}) \\
k'_{18} &= k_{18} \oplus k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{23}) & k'_{19} &= k_{19} \oplus k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{20}) \\
k'_{20} &= k_{20} \oplus k_{16} \oplus k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{21}) \oplus c_i & k'_{21} &= k_{21} \oplus k_{17} \oplus k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{22}) \\
k'_{22} &= k_{22} \oplus k_{18} \oplus k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{23}) & k'_{23} &= k_{23} \oplus k_{19} \oplus k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{20})
\end{aligned}$$

B - The New Representation of the AES-192 Key Schedule

As for the key schedule of AES-128, the algorithm of Leander, Minaud and Rønjom [LMR15] can be used in order to extract invariant subspaces for the AES-192 key schedule. When we apply this algorithm with the permutation F corresponding to two rounds of key schedule, we obtained two invariant subpaces of dimension 12. In the following, we are going to make them appear more clearly.

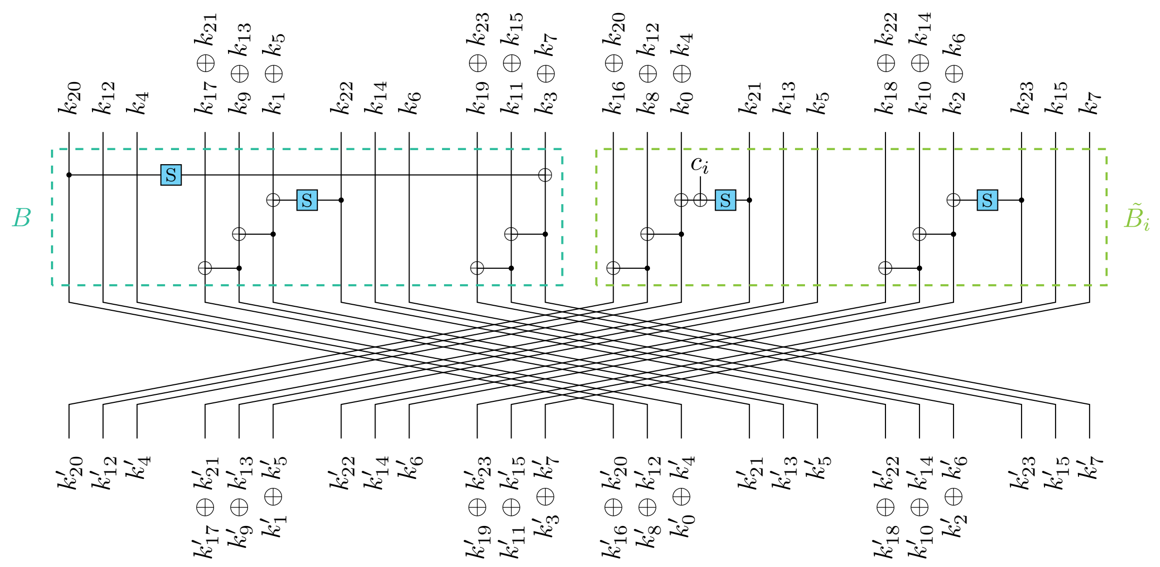


Figure 13: One round of the AES-192 key schedule (alternative representation).

Instead of the usual representation of the AES key schedule, we now describe an alternative representation, with a linear change of variables for the key schedule state:

$$\begin{aligned}
s_0 &= k_{20} & s_1 &= k_{12} & s_2 &= k_4 \\
s_3 &= k_{17} \oplus k_{21} & s_4 &= k_9 \oplus k_{13} & s_5 &= k_1 \oplus k_5 \\
s_6 &= k_{22} & s_7 &= k_{14} & s_8 &= k_6 \\
s_9 &= k_{19} \oplus k_{23} & s_{10} &= k_{11} \oplus k_{15} & s_{11} &= k_3 \oplus k_7 \\
s_{12} &= k_{16} \oplus k_{20} & s_{13} &= k_8 \oplus k_{12} & s_{14} &= k_0 \oplus k_4 \\
s_{15} &= k_{21} & s_{16} &= k_{13} & s_{17} &= k_5 \\
s_{18} &= k_{18} \oplus k_{22} & s_{19} &= k_{10} \oplus k_{14} & s_{20} &= k_2 \oplus k_6 \\
s_{21} &= k_{23} & s_{22} &= k_{15} & s_{23} &= k_7
\end{aligned}$$

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

$$\begin{aligned}
s'_0 &= k'_{20} = k_{20} \oplus k_{16} \oplus k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{21}) \oplus c_i &&= s_{12} \oplus s_{13} \oplus s_{14} \oplus S(s_{15}) \oplus c_i \\
s'_1 &= k'_{12} = k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{21}) \oplus c_i &&= s_{13} \oplus s_{14} \oplus S(s_{15}) \oplus c_i \\
s'_2 &= k'_4 = k_4 \oplus k_0 \oplus S(k_{21}) \oplus c_i &&= s_{14} \oplus S(s_{15}) \oplus c_i \\
s'_3 &= k'_{17} \oplus k'_{21} = k_{21} &&= s_{15} \\
s'_4 &= k'_9 \oplus k'_{13} = k_{13} &&= s_{16} \\
s'_5 &= k'_1 \oplus k'_5 = k_5 &&= s_{17} \\
s'_6 &= k'_{22} = k_{22} \oplus k_{18} \oplus k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{23}) &&= s_{18} \oplus s_{19} \oplus s_{20} \oplus S(s_{21}) \\
s'_7 &= k'_{14} = k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{23}) &&= s_{19} \oplus s_{20} \oplus S(s_{21}) \\
s'_8 &= k'_6 = k_6 \oplus k_2 \oplus S(k_{23}) &&= s_{20} \oplus S(s_{21}) \\
s'_9 &= k'_{19} \oplus k'_{23} = k_{23} &&= s_{21} \\
s'_{10} &= k'_{11} \oplus k'_{15} = k_{15} &&= s_{22}
\end{aligned}$$

$$
\begin{aligned}
s'_{11} &= k'_3 \oplus k'_7 = k_7 && = s_{23}\\
s'_{12} &= k'_{16} \oplus k'_{20} = k_{20} && = s_0\\
s'_{13} &= k'_8 \oplus k'_{12} = k_{12} && = s_1\\
s'_{14} &= k'_4 \oplus k'_0 = k_4 && = s_2\\
s'_{15} &= k'_{21} = k_{21} \oplus k_{17} \oplus k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{22}) && = s_3 \oplus s_4 \oplus s_5 \oplus S(s_6)\\
s'_{16} &= k'_{13} = k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{22}) && = s_4 \oplus s_5 \oplus S(s_6)\\
s'_{17} &= k'_5 = k_5 \oplus k_1 \oplus S(k_{22}) && = s_5 \oplus S(s_6)\\
s'_{18} &= k'_{18} \oplus k'_{22} = k_{22} && = s_6\\
s'_{19} &= k'_{10} \oplus k'_{14} = k_{14} && = s_7\\
s'_{20} &= k'_2 \oplus k'_6 = k_6 && = s_8\\
s'_{21} &= k'_{23} = k_{23} \oplus k_{19} \oplus k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{20}) && = s_9 \oplus s_{10} \oplus s_{11} \oplus S(s_0)\\
s'_{22} &= k'_{15} = k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{20}) && = s_{10} \oplus s_{11} \oplus S(s_0)\\
s'_{23} &= k'_7 = k_7 \oplus k_3 \oplus S(k_{20}) && = s_{11} \oplus S(s_0)
\end{aligned}
$$

This is represented by Figure 13. To further simplify the description, we write the output as

$$(s'_{12}, s'_{13}, s'_{14}, s'_{15}, s'_{16}, s'_{17}, s'_{18}, s'_{19}, s'_{20}, s'_{21}, s'_{22}, s'_{23}, \quad s'_0, s'_1, s'_2, s'_3, s'_4, s'_5, s'_6, s'_7, s'_8, s'_9, s'_{10}, s'_{11}).$$

This corresponds to "untwisting" the rotation of the 12-byte blocks, so that each block of 12 output bytes depend on the same 12 input bytes. This gives us our alternate representation of the AES-192 key schedule.

1. We first apply the linear transformation A to the state, corresponding to the change of variable above:

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

2. Then the rounds of the key schedule are seen as the concatenation of 2 functions B and $\tilde{B}_i$ each acting on 96-bit words (12 bytes), as seen in Figure 14.

3. In order to extract the subkey of round r, another linear transformation $C_{r \bmod 2}$ is applied to the state, depending of the round number modulo 2. C_i is defined as $C_i = A^{-1} \times \mathsf{SR}^i$, with SR the matrix corresponding to rotation of 12 bytes to the right. In particular $C_0 = A^{-1}$.

In this new representation, there are clearly 2 independant parts each acting on 12 bytes, and the subkeys are reconstructed with linear combinations of the alternative key-schedule state.

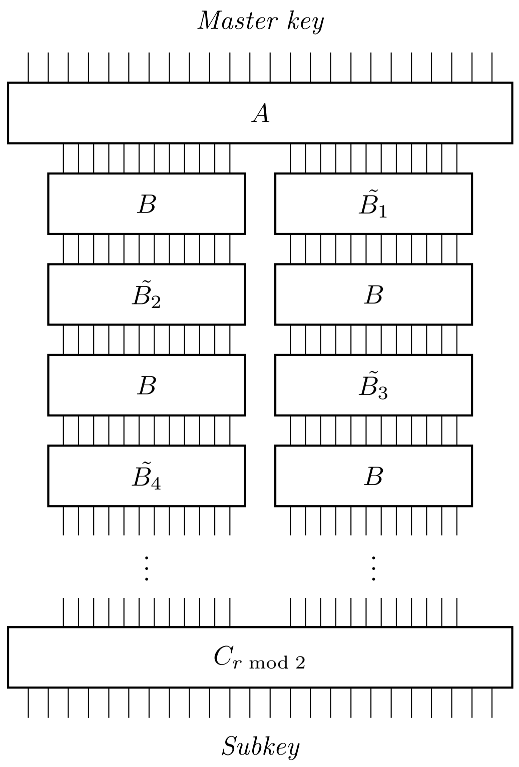


Figure 14: r rounds of the AES-192 key schedule in the new representation. B_i and $\tilde{B}$ are defined in Figure 13.

C - The AES-256 Key Schedule

The AES-256 key schedule allows us to derive 15 subkeys from a 256-bit master key. The operations used are the same as in the AES-128 and the AES-192: RotWord, SubWord and RCon. However, in the 256-bit version, the initialization is different: eight 32-bit words w_i $(0 \le i < 8)$ are filled with the bytes of the master key.

The way of constructing the words w_i $(i \ge 8)$ also differs:

- if $i \equiv 0 \bmod 8$, $w_i = \mathsf{SubWord}(\mathsf{RotWord}(w_{i-1})) \oplus \mathsf{RCon}(i/8) \oplus w_{i-8}$.
- else if $i \equiv 0 \bmod 4$, $w_i = \mathsf{SubWord}(w_{i-1}) \oplus w_{i-8}$.
- else, $w_i = w_{i-1} \oplus w_{i-8}$.

The subkey at round i is the concatenation of the words w_{4i} to w_{3+4i}. So in AES-256, the key schedule state corresponds to two consecutive subkeys. As for the AES-128, we can also express the key schedule at byte level using k_i $(0 \le i < 32)$ to denote the key-schedule state, and k'_i for the state after one round of key schedule. This corresponds to the following equations:

$$
\begin{aligned}
k'_0 &= k_0 \oplus S(k_{29}) \oplus c_i \\
k'_1 &= k_1 \oplus S(k_{30}) \\
k'_2 &= k_2 \oplus S(k_{31}) \\
k'_3 &= k_3 \oplus S(k_{28})
\end{aligned}
$$

$$
\begin{aligned}
k'_4 &= k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i \\
k'_5 &= k_5 \oplus k_1 \oplus S(k_{30}) \\
k'_6 &= k_6 \oplus k_2 \oplus S(k_{31}) \\
k'_7 &= k_7 \oplus k_3 \oplus S(k_{28}) \\
k'_8 &= k_8 \oplus k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i \\
k'_9 &= k_9 \oplus k_5 \oplus k_1 \oplus S(k_{30}) \\
k'_{10} &= k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{31}) \\
k'_{11} &= k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{28}) \\
k'_{12} &= k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i \\
k'_{13} &= k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{30}) \\
k'_{14} &= k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{31}) \\
k'_{15} &= k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{28}) \\
k'_{16} &= k_{16} \oplus S(k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i) \\
k'_{17} &= k_{17} \oplus S(k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{30})) \\
k'_{18} &= k_{18} \oplus S(k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{31})) \\
k'_{19} &= k_{19} \oplus S(k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{28})) \\
k'_{20} &= k_{20} \oplus k_{16} \oplus S(k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i) \\
k'_{21} &= k_{21} \oplus k_{17} \oplus S(k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{30})) \\
k'_{22} &= k_{22} \oplus k_{18} \oplus S(k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{31})) \\
k'_{23} &= k_{23} \oplus k_{19} \oplus S(k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{28})) \\
k'_{24} &= k_{24} \oplus k_{20} \oplus k_{16} \oplus S(k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i) \\
k'_{25} &= k_{25} \oplus k_{21} \oplus k_{17} \oplus S(k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{30})) \\
k'_{26} &= k_{26} \oplus k_{22} \oplus k_{18} \oplus S(k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{31})) \\
k'_{27} &= k_{27} \oplus k_{23} \oplus k_{19} \oplus S(k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{28})) \\
k'_{28} &= k_{28} \oplus k_{24} \oplus k_{20} \oplus k_{16} \oplus S(k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i) \\
k'_{29} &= k_{29} \oplus k_{25} \oplus k_{21} \oplus k_{17} \oplus S(k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{30})) \\
k'_{30} &= k_{30} \oplus k_{26} \oplus k_{22} \oplus k_{18} \oplus S(k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{31})) \\
k'_{31} &= k_{31} \oplus k_{27} \oplus k_{23} \oplus k_{19} \oplus S(k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{28}))
\end{aligned}
$$

D - The New Representation of the AES-256 Key Schedule

As for the AES-128 key schedule, the algorithm of Leander, Minaud and Rønjom [LMR15] can be used in order to extract invariant subspaces for the AES-256 key schedule. It allows us to find four invariant subpaces of dimension 8. In the following, we are going to make them appear more clearly.

Instead of the usual representation of the AES key schedule, we now describe an alternative representation, with a linear change of variables for the key schedule state:

$$
\begin{array}{llll}
s_0 = k_{15} & s_1 = k_{31} & s_2 = k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 & s_3 = k_{30} \oplus k_{26} \oplus k_{22} \oplus k_{18} \\
s_4 = k_{13} \oplus k_5 & s_5 = k_{29} \oplus k_{21} & s_6 = k_{12} \oplus k_8 & s_7 = k_{28} \oplus k_{24} \\
s_8 = k_{14} & s_9 = k_{30} & s_{10} = k_{13} \oplus k_9 \oplus k_5 \oplus k_1 & s_{11} = k_{29} \oplus k_{25} \oplus k_{21} \oplus k_{17} \\
s_{12} = k_{12} \oplus k_4 & s_{13} = k_{28} \oplus k_{20} & s_{14} = k_{15} \oplus k_{11} & s_{15} = k_{31} \oplus k_{27}
\end{array}
$$

$$\begin{array}{llll}
s_{16} = k_{13} & s_{17} = k_{29} & s_{18} = k_{12} \oplus k_8 \oplus k_4 \oplus k_0 & s_{19} = k_{28} \oplus k_{24} \oplus k_{20} \oplus k_{16} \\
s_{20} = k_{15} \oplus k_7 & s_{21} = k_{31} \oplus k_{23} & s_{22} = k_{14} \oplus k_{10} & s_{23} = k_{30} \oplus k_{26} \\
s_{24} = k_{12} & s_{25} = k_{28} & s_{26} = k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 & s_{27} = k_{31} \oplus k_{27} \oplus k_{23} \oplus k_{19} \\
s_{28} = k_{14} \oplus k_6 & s_{29} = k_{30} \oplus k_{22} & s_{30} = k_{13} \oplus k_9 & s_{31} = k_{29} \oplus k_{25}
\end{array}$$

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

$$\begin{aligned}
s'_0 &= k'_{15} = k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{28}) \\
&= s_{26} \oplus S(s_{25}) \\
s'_1 &= k'_{31} = k_{31} \oplus k_{27} \oplus k_{23} \oplus k_{19} \oplus S(k_{15} \oplus k_{11} \oplus k_7 \oplus k_3 \oplus S(k_{28})) \\
&= s_{27} \oplus S(s_{26} \oplus S(s_{25}))) \\
s'_2 &= k'_{14} \oplus k'_{10} \oplus k'_6 \oplus k'_2 = k_{14} \oplus k_6 \\
&= s_{28} \\
s'_3 &= k'_{30} \oplus k'_{26} \oplus k'_{22} \oplus k'_{18} = k_{30} \oplus k_{22} \\
&= s_{29} \\
s'_4 &= k'_{13} \oplus k'_5 = k_{13} \oplus k_9 \\
&= s_{30} \\
s'_5 &= k'_{29} \oplus k'_{21} = k_{29} \oplus k_{25} \\
&= s_{31} \\
s'_6 &= k'_{12} \oplus k'_8 = k_{12} \\
&= s_{24} \\
s'_7 &= k'_{28} \oplus k'_{24} = k_{28} \\
&= s_{25} \\
s'_8 &= k'_{14} = k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{31}) \\
&= s_2 \oplus S(s_1) \\
s'_9 &= k'_{30} = k_{30} \oplus k_{26} \oplus k_{22} \oplus k_{18} \oplus S(k_{14} \oplus k_{10} \oplus k_6 \oplus k_2 \oplus S(k_{31})) \\
&= s_3 \oplus S(s_2 \oplus S(s_1))) \\
s'_{10} &= k'_{13} \oplus k'_9 \oplus k'_5 \oplus k'_1 = k_{13} \oplus k_5 \\
&= s_4 \\
s'_{11} &= k'_{29} \oplus k'_{25} \oplus k'_{21} \oplus k'_{17} = k_{29} \oplus k_{21} \\
&= s_5 \\
s'_{12} &= k'_{12} \oplus k'_4 = k_{12} \oplus k_8 \\
&= s_6 \\
s'_{13} &= k'_{28} \oplus k'_{20} = k_{28} \oplus k_{24} \\
&= s_7 \\
s'_{14} &= k'_{15} \oplus k'_{11} = k_{15} \\
&= s_0 \\
s'_{15} &= k'_{31} \oplus k'_{27} = k_{31} \\
&= s_1 \\
s'_{16} &= k'_{13} = k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{30}) \\
&= s_{10} \oplus S(s_9)
\end{aligned}$$

$$\begin{aligned}
s'_{17} &= k'_{29} = k_{29} \oplus k_{25} \oplus k_{21} \oplus k_{17} \oplus S(k_{13} \oplus k_9 \oplus k_5 \oplus k_1 \oplus S(k_{30})) \\
&= s_{11} \oplus S(s_{10} \oplus S(s_9)) \\
s'_{18} &= k'_{12} \oplus k'_8 \oplus k'_4 \oplus k'_0 = k_{12} \oplus k_4 \\
&= s_{12} \\
s'_{19} &= k'_{28} \oplus k'_{24} \oplus k'_{20} \oplus k'_{16} = k_{28} \oplus k_{20} \\
&= s_{13} \\
s'_{20} &= k'_{15} \oplus k'_7 = k_{15} \oplus k_{11} \\
&= s_{14} \\
s'_{21} &= k'_{31} \oplus k'_{23} = k_{31} \oplus k_{27} \\
&= s_{15} \\
s'_{22} &= k'_{14} \oplus k'_{10} = k_{14} \\
&= s_8 \\
s'_{23} &= k'_{30} \oplus k'_{26} = k_{30} \\
&= s_9 \\
s'_{24} &= k'_{12} = k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i \\
&= s_{18} \oplus S(s_{17}) \oplus c_i \\
s'_{25} &= k'_{28} = k_{28} \oplus k_{24} \oplus k_{20} \oplus k_{16} \oplus S(k_{12} \oplus k_8 \oplus k_4 \oplus k_0 \oplus S(k_{29}) \oplus c_i) \\
&= s_{19} \oplus S(s_{18} \oplus S(s_{17}) \oplus c_i) \\
s'_{26} &= k'_{15} \oplus k'_{11} \oplus k'_7 \oplus k'_3 = k_{15} \oplus k_7 \\
&= s_{20} \\
s'_{27} &= k'_{31} \oplus k'_{27} \oplus k'_{23} \oplus k'_{19} = k_{31} \oplus k_{23} \\
&= s_{21} \\
s'_{28} &= k'_{14} \oplus k'_6 = k_{14} \oplus k_{10} \\
&= s_{22} \\
s'_{29} &= k'_{30} \oplus k'_{22} = k_{30} \oplus k_{26} \\
&= s_{23} \\
s'_{30} &= k'_{13} \oplus k'_9 = k_{13} \\
&= s_{16} \\
s'_{31} &= k'_{29} \oplus k'_{25} = k_{29} \\
&= s_{17}
\end{aligned}$$

To further simplify the description, we write the output as

$$\begin{aligned}
&(s'_8, s'_9, s'_{10}, s'_{11}, s'_{12}, s'_{13}, s'_{14}, s'_{15}, \quad s'_{16}, s'_{17}, s'_{18}, s'_{19}, s'_{20}, s'_{21}, s'_{22}, s'_{23}, \\
&s'_{24}, s'_{25}, s'_{26}, s'_{27}, s'_{28}, s'_{29}, s'_{30}, s'_{31}, \quad s'_0, s'_1, s'_2, s'_3, s'_4, s'_5, s'_6, s'_7).
\end{aligned}$$

This corresponds to “untwisting” the rotation of the 8-byte blocks, so that each block of 8 output bytes depend on the same 8 input bytes. This results in our alternate representation of the AES-256 key schedule:

1. We first apply the linear transformation A to the state, corresponding to the change of variable above:

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

2. Then the rounds of the key schedule are seen as the concatenation of 4 functions each acting on 64-bit words (8 bytes), as seen in Figure 15.

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 rotation of 8 bytes to the right. In particular $C_0 = A^{-1}$.

In this new representation, there are clearly 4 independant parts each acting on 8 bytes, and the subkeys are reconstructed with linear combinations of the alternative key-schedule state.

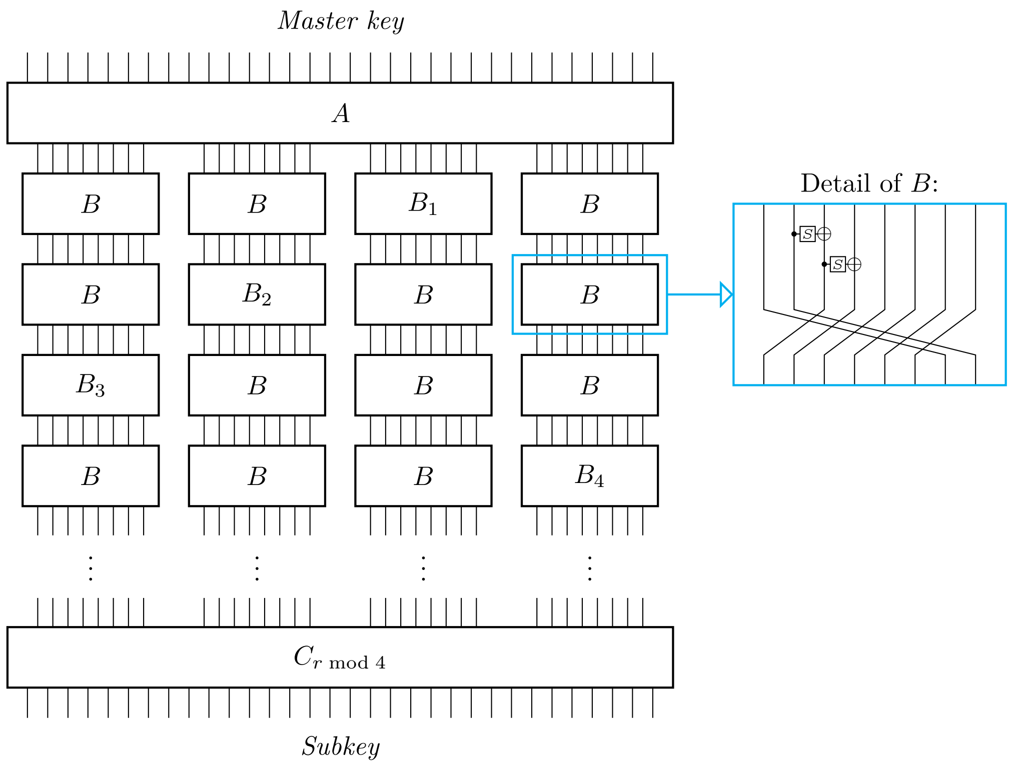


Figure 15: r rounds of the AES-256 key schedule in the new representation. B_i is similar to B but the round constant c_i is XORed to the output of the first S-box. The matrices of C_i for $0 \leq i \leq 3$ are given in Appendix.

V) 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:

Proposition 1. *Let P_r and P'_r defined in one of the following ways:*

- *AES-128:* $P_r = (k_r[5], k_r[7], k_r[13], k_r[15])$, *and* $P'_r = (k_r[4], k_r[6], k_r[12], k_r[14])$.
- *AES-128:* $P_r = (k_r[0] \oplus k_r[4], k_r[2] \oplus k_r[6], k_r[8] \oplus k_r[12], k_r[10] \oplus k_r[14])$, *and* $P'_r = (k_r[1] \oplus k_r[5], k_r[3] \oplus k_r[7], k_r[9] \oplus k_r[13], k_r[11] \oplus k_r[15])$.
- *AES-256:* $P_r = (k_r[5], k_r[7], k_r[13], k_r[15], k_r[21], k_r[23], k_r[29], k_r[31])$, *and* $P'_r = (k_r[4], k_r[6], k_r[12], k_r[14], k_r[20], k_r22], k_r[28], k_r[30])$.
- *AES-256:* $P_r = (k_r[0] \oplus k_r[4], k_r[2] \oplus k_r[6], k_r[8] \oplus k_r[12], k_r[10] \oplus k_r[14])$, *and* $P'_r = (k_r[1] \oplus k_r[5], k_r[3] \oplus k_r[7], k_r[9] \oplus k_r[13], k_r[11] \oplus k_r[15])$.

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 knowledge of P_{r_0} and $P'_{r_0 \pm 1}$ allows us to know two non-consecutive parts in our new representation. Based on the matrices given in the appendix, we notice that this knowledge allows us to extract one quarter of the bytes of the key state after any number of rounds. ☐

This proposition is a generalization of the observations made for AES-128 by Dunkelman and Keller:

Observation 3 ([DK08b]). *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 ([DK08b]). *For each $0 \le i \le 3$, the subkeys of AES satisfy the relation:*

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

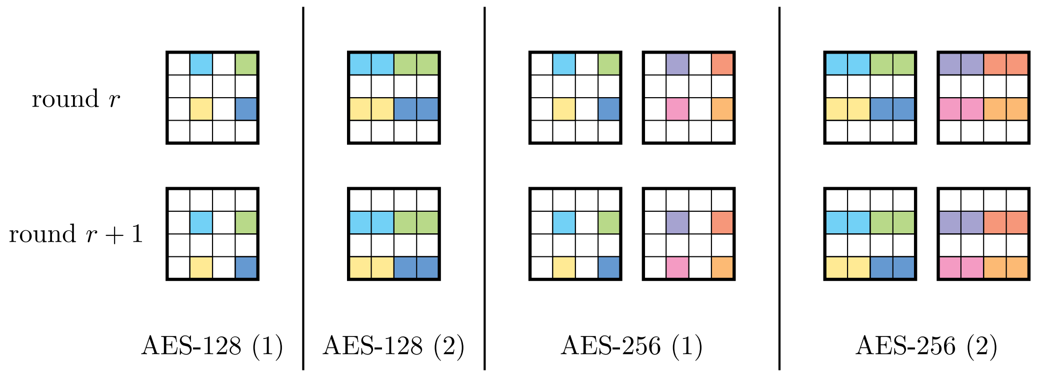


Figure 16: Representation of the position of the bytes of the proposition. In AES-128 (2) and AES-256 (2), only the XOR of the two bytes of the same color must be known.

Conclusion

We found new representations of the AES key schedule for each of the versions: 128, 192 and 256 bits. We have shown that, up to linear transformations, any number of rounds of key schedule can be seen as the concatenation of 4 functions each acting on 4 bytes in the case of AES-128, 2 functions each acting on 12 bytes in the case of AES-192, and 4 functions each acting on 8 bytes in the case of AES-256. This explains the presence of many short length cycles when we iterate an odd number of rounds of key schedule, and it also allows us to find new properties on the key schedule. The key schedule was already considered as the least secure part of the AES: our observation confirms this idea. These representations also show that the AES key schedule cannot and should not be considered as a random permutation, even after a large number of rounds.

The mixFeed and ALE ciphers have the common feature that they iterate the key schedule. The purpose of the creators of these ciphers was that each AES call uses a different key, in order to prevent attacks that use states encrypted under the same key. However, our analysis demonstrates that short length cycles are obtained when an odd number of rounds of key schedule are iterated. By exploiting this fact, we prove that the probability of success of the forgery attack of Khairallah against mixFeed([Kha19]) is approximately 44%, and we found a new attack on the cipher ALE. The attack against mixFeed only requires to encrypt a known plaintext of length $2^{37.7}$ bytes, and has negligible memory and time complexity. MixFeed is currently a second round candidate of NIST Lightweight Standardiastion Process, but will probably be eliminated because of this analysis. Regarding ALE, the short length cycles cause some keys to be reused. Therefore, we can locate states encrypted under the same key, and adapt previous attacks against LEX, with the help of the tool created by Bouillaguet, Derbez, and Fouque [BDF11].

Our new representations and the resulting properties have not yet allowed us to improve cryptanalysis of the AES, but it's still an interesting line of research.

References

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001.

[BDF11] Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic search of attacks on round-reduced AES and applications. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 169–187. Springer, Heidelberg, August 2011.

[BDK+10] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 299–319. Springer, Heidelberg, May / June 2010.

[Bir07] Alex Biryukov. The design of a stream cipher LEX. In Eli Biham and Amr M. Youssef, editors, *SAC 2006*, volume 4356 of *LNCS*, pages 67–75. Springer, Heidelberg, August 2007.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 1–18. Springer, Heidelberg, December 2009.

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and related-key attack on the full AES-256. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 231–249. Springer, Heidelberg, August 2009.

[BMR+14] Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-based lightweight authenticated encryption. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 447–466. Springer, Heidelberg, March 2014.

[CN19a] Bishwajit Chakraborty and Mridul Nandi. 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`.

[CN19b] Bishwajit Chakraborty and Mridul Nandi. Security proof of mixFeed, 2019. `https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2019/documents/papers/security-proof-of-mixfeed-lwc2019.pdf`.

[DK08a] Orr Dunkelman and Nathan Keller. A new attack on the LEX stream cipher. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 539–556. Springer, Heidelberg, December 2008.

[DK08b] Orr Dunkelman and Nathan Keller. Treatment of the initial value in time-memory-data tradeoff attacks on stream ciphers. Cryptology ePrint Archive, Report 2008/311, 2008. `http://eprint.iacr.org/2008/311`.

[DR99] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. Submission to the AES competition, 1999. `https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf`.

[DR05] Joan Daemen and Vincent Rijmen. The Pelican MAC function 2.0. Cryptology ePrint Archive, Report 2005/088, 2005. http://eprint.iacr.org/2005/088.

[Kha19] Mustafa Khairallah. Weak keys in the rekeying paradigm: Application to COMET and mixFeed. *IACR Trans. Symm. Cryptol.*, 2019(4):272–289, 2019.

[KR14] Dmitry Khovratovich and Christian Rechberger. The LOCAL attack: Cryptanalysis of the authenticated encryption scheme ALE. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 174–184. Springer, Heidelberg, August 2014.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of PRINTcipher: The invariant subspace attack. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 206–221. Springer, Heidelberg, August 2011.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of robin, iSCREAM and Zorro. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 254–283. Springer, Heidelberg, April 2015.

[WWH$^+$13] Shengbao Wu, Hongjun Wu, Tao Huang, Mingsheng Wang, and Wenling Wu. Leaked-state-forgery attack against the authenticated encryption algorithm ALE. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 377–404. Springer, Heidelberg, December 2013.

Appendix

A - AES-128

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

$$C_1 = A^{-1} \times SR = \begin{pmatrix}
0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&1\\
0&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0\\
1&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0\\
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0
\end{pmatrix}$$

$$C_2 = A^{-1} \times SR \times SR = \begin{pmatrix}
0&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0\\
1&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0\\
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0\\
0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0
\end{pmatrix}$$

$$C_3 = A^{-1} \times SR \times SR \times SR = \begin{pmatrix}
1&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0\\
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0\\
0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&1\\
0&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0\\
1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

B - AES-192

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

$$C_1 = \begin{pmatrix}
0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

C - AES-256

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

$$C_1 = \begin{pmatrix}
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&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&1&0&0&0&0&0&1&0&0&0&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&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 \\
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&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&0&0&1&0&0&0&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&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&1&0 \\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\
1&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&0 \\
0&0&0&0&0&0&1&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&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&0&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&0&0&0 \\
1&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&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&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&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&1&0&0&0&0&0&1&0&0&0&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&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&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&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&0&0&1&0&0&0&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&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&1 \\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\
0&1&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 \\
0&0&0&0&0&0&0&1&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&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&0&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&0&0 \\
0&1&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&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0
\end{pmatrix}$$

$$C_2 = \begin{pmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 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 & 0 & 1 & 0 & 0 & 0 & 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 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 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 & 0 & 0 & 1 & 0 & 0 & 0 & 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 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 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 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 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 & 1 & 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 & 0 & 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 & 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 & 1 & 0 & 0 & 0 & 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 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 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 & 0 & 1 & 0 & 0 & 0 & 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 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
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 & 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 & 0 & 0 & 1 & 0 & 0 & 0 & 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 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 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 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 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 & 1 \\
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 & 0 & 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 & 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 & 1 & 0 & 0 & 0 & 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 & 0 & 0 & 0
\end{pmatrix}$$

$$C_3 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 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 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 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 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 0 & 0 & 1 & 0 & 0 & 0 & 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 \\
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 & 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 & 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 & 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 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 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 & 1 & 0 & 0 & 0 & 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 & 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 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 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 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 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 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 0 & 0 & 1 & 0 & 0 & 0 & 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 & 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 & 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 & 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 & 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 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 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 & 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 & 1 & 0 & 0 & 0 & 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 & 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 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$