# Is AEZ v4.1 Sufficiently Resilient Against Key-Recovery Attacks?

Colin Chaigneau, Henri Gilbert

# Is AEZ v4.1 Sufficiently Resilient Against Key-Recovery Attacks?

Colin Chaigneau and Henri Gilbert

[1] UVSQ, Versailles, France
colin.chaigneau@uvsq.fr
[2] ANSSI, France
henri.gilbert@ssi.gouv.fr

**Abstract.** AEZ is a parallelizable, AES-based authenticated encryption algorithm that is well suited for software implementations on processors equipped with the AES-NI instruction set. It aims at offering exceptionally strong security properties such as nonce and decryption-misuse resistance and optimal security given the selected ciphertext expansion. AEZ was submitted to the authenticated ciphers competition CAESAR and was selected in 2015 for the second round of the competition.
In this paper, we analyse the resilience of the latest algorithm version, AEZ v4.1 (October 2015), against key-recovery attacks. While AEZ modifications introduced in 2015 were partly motivated by thwarting a key-recovery attack of birthday complexity against AEZ v3 published at Asiacrypt 2015 by Fuhr, Leurent and Suder, we show that AEZ v4.1 remains vulnerable to a key-recovery attack of similar complexity and security impact. Our attack leverages the use, in AEZ, of an underlying tweakable block cipher based on a 4-round version of AES.
Although the presented key-recovery attack does not violate the security claims of AEZ since the designers made no claim for beyond-birthday security, it can be interpreted as an indication that AEZ does not fully meet the objective of being an extremely conservative and misuse-resilient algorithm.

**Keywords:** CAESAR competition, cryptanalysis, authenticated encryption, AEZ, key recovery

## 1 Introduction

Authenticated Encryption (AE) algorithms aim at providing encryption and message authentication in a combined way. The purpose of the ongoing international competition CAESAR, that was launched in 2014, is to select a portfolio of AE algorithms that offer better security guarantees and/or improved performances as compared with existing AE standards such as AES-GCM [1].

AEZ [2, 3] is an AES-based AE scheme designed by Hoang, Krovetz, and Rogaway. It is a high-profile CAESAR candidate and was selected in July 2015 for the second round of the competition. The AEZ construction can be viewed as a mode of operation of an underlying block cipher – more precisely of a mixture of AES versions with 4 and 10 rounds denoted `AES4` and `AES10`. AEZ uses secret offsets and round keys derived from the authenticated encryption key $K$. AEZ is parallelizable and particularly well suited for software implementations on processors equiped with the AES-NI instruction set. On such environments, its computational cost is lower than the one of AES-GCM and close to the one of OCB [4]. AEZ aims at providing an unusually strong nonce and decryption misuse resistance and more generally best achievable security given the selected amount


IACR Transactions on Symmetric Cryptology ISSN 2519-173X, Vol. 2016, No. 1, pp. 114–133
DOI:10.13154/tosc.v2016.i1.114-133

of plaintext expansion. These security properties are captured by the notion of *robust authenticated encryption* (RAE), a demanding security notion, not attainable by *online* AE schemes, i.e. AE schemes allowing a single-pass blockwise plaintext encryption with constant memory [5]. The RAE security notion and the security arguments underlying the AEZ construction were detailed in the Eurocrypt 2015 paper [2].

In this paper, we analyze the resilience of the latest algorithm version, AEZ v4.1 (October 2015) [3], against key-recovery attacks. We show that the AEZ modifications introduced in 2015, that were partly motivated by thwarting a key derivation attack with birthday complexity against AEZ v3 published at Asiacrypt 2015 by Fuhr, Leurent and Suder [6], do not prevent the existence of key derivation attacks of birthday complexity against AEZ v4.1. In both cases, the attack rests on the fact that AEZ was designed as to be potentially usable either without nonce[1] or with a repeating nonce values without other security impact than the detectability of repeated (associated data, message) pairs. Unlike the attack of [6], our most efficient key derivation attack relies on the use of `AES4` in AEZ. It can therefore not be transposed to the more conservative but less efficient scaled up version of AEZ where only `AES10` is used instead of a mixture of `AES10` and `AES4`.

Neither the AEZ v3 attack of [6] nor our attack on AEZ v4.1 violates the security claims of AEZ since the designers made no claim for beyond-birthday security. It should also be noted that if one takes into account the limitation of the amount of data processed under the same key to $2^{48}$ bytes required by the designers, their success probability becomes relatively low.

We nevertheless believe that the vulnerability of AEZ4.1 to a key derivation attack of birthday complexity[2] represents an undesirable property, particularly for an algorithm that otherwise aims at satisfying a very strong notions of security and at being exceptionally resilient in various misuse situations. Indeed, even though the existence of distinguishers of birthday complexity against modes of operation of block ciphers is not so unusual, the existence of full key derivation attacks of birthday complexity is far less frequent and raises in the case of AEZ v4.1 the following resilience questions (exactly the same as those raised by the attack of [6] in the case of AEZ v3). First, our attack allows to recover the whole key material with a much higher success probability than the one that would result from generic attacks for typical key sizes, e.g. 128, 256, or 384 bits even if the below-birthday data limitation of $2^{48}$ bytes imposed by the designers is respected. Second, this probability can become arbitrarily close to 1 in the algorithm misuse case where the data limitation of $2^{48}$ bytes cannot be enforced and "birthday" amounts of data can be processed.

Our results are summarized in Table 1. Our most efficient attack essentially consists of two phases. In a first phase, 128 bits of key material used for pre-whitening the inputs to some `AES4` and `AES10` computations are derived, using a birthday attack. We show that the universal hashing part of the AEZ computations, on which the attack of [6] concentrated, can still be targeted by some birthday attacks. However the key material information this provides is less suited for continuing the attack than a 128-bit sub-key that can be derived by targeting instead the encipherment part of the AEZ computations. This sub-key determines the pre-whitening of some `AES4` computations also involved the encipherment procedure. In a second phase of the attack, we encrypt particular plaintext structures and detect plaintext pairs leading to a special differential behaviour in the last three rounds of these `AES4` computations. This allows to recover the remaining of the key

[1] This is allowed by the specification, with the warning that "a nonce must be used unless one has certitude that, even in the presence of the adversary, all encrypted [(associated data, message)] pairs will be distinct[...]" [3].

[2] and to a resulting below-birthday attack of abnormally high success probability, as discussed below

material.

Table 1: AEZ attacks complexities.

| AEZ version | Data complexity (blocks)[3] | Success prob. | Ref. |
|---|---|---|---|
| AEZ v4.1 | $2^{66.5}$ | 0.5 | **This paper** |
| AEZ v4.1 | $2^{44}$ | $2^{-45.7}$ | **This paper** |
| AEZ v3 | $2^{66.6}$ | 1 | [6] |
| AEZ v3 | $2^{44}$ | $2^{-45.2}$ | [6] |

The paper is organized as follows. Section 2 outlines the parts of the AEZ v4.1 specifications that are useful for our attack and the main differences between AEZ v4.1 and AEZ v3. Section 3 first describes partial attacks of birthday complexity allowing to recover a 128-bit piece of the key material (Section 3.1). The combination of these partial attacks can be viewed as a suboptimal key derivation attack of birthday time and data complexity. Then we detail our most efficient attack on AEZ v4.1 (Section 3.2), that exploits the use of `AES4` in AEZ. The attack of Section 3.2 has the property (not shared by the combined attack of Section 3.1) that its success probability remains abnormally high if the amount of data processed under the same key is limited to the below-birthday threshold of $2^{48}$ bytes.

## 2 Description of AEZ

The following input and output arguments are used in AEZ:

- a plaintext $P$ of *plen* bits;
- a key $K$ of arbitrary length *klen* bits. The default value of *klen* is 384 bits and *klen* values of at least 128 bits are recommended;
- a nonce $N$ of length *nlen* bits. The use of nonce values of length at most 128 bits is recommended and $nlen = 0$ is allowed, as well as the use of several nonce lengths for authenticated encryptions under the same key;
- a string-valued or more generally vector-valued associated data $A = (A_1, \cdots, A_m)$ of $m$ strings, of total length *alen* bits. A string-valued associated data can be viewed as a vector with $m = 1$ components;
- a ciphertext $C$ of *clen* bits.

Although their lengths are defined in bits, all these arguments are required to consist of an integer number of bytes.

AEZ is also parametrized by the authenticator byte length `Abytes` of default value 16. The corresponding number of bits $\tau = 8 \times$ `Abytes` represents the plaintext expansion $clen - plen$ and also the number of zero bits that shall be appended to the plaintext $P$ before encipherment if $P$ is not the empty string. The augmented plaintext $(P \,||\, 0^\tau)$ is denoted by $\overline{P}$ in the sequel and the binary representation of $\tau$ as a 128-bit word is denoted by $[\tau]_{128}$.

[3] Chosen plaintexts

The AEZ authenticated encryption process can be viewed as follows. First a vector-valued tweak $T = ([\tau]_{128}, N, A_1, \cdots, A_m)$, that encodes the triplet $(\tau, N, A)$ is derived. Then, depending on the the plaintext length $plen$, different encipherment functions are applied:

- $\texttt{AEZ-prf}(K, T, \tau)$ is returned if $plen = 0$,
- $\texttt{AEZ-tiny}(K, T, \overline{P})$ is returned if $0 < plen < 256 - \tau$,
- $\texttt{AEZ-core}(K, T, \overline{P})$ is returned if $256 - \tau \leq plen$.

The way the tweak argument $T$ is processed in all these functions consists of deriving an associated universal hash value $\Delta = \texttt{AEZ-hash}(K, T)$ of length 128 bits and then using $\Delta$ as an offset in some parts of the encipherment computations.

Since we do not use `AEZ-tiny` in our attack, we only describe `AEZ-prf` and `AEZ-core`.

## 2.1 Tweaked Instances of AES4 and AES10 Used in AEZ

AEZ uses AES-based tweakable block ciphers [7] (TBC) using the XE and XEX constructions. Three sub-keys $I$, $J$, and $L$, of length 128 bits each, are used in these TBC, which are derived from the key $K$ in a way that depends of the key length $klen$ :

- if $klen = 384$, then $I \,||\, J \,||\, L = K$;
- if $klen \neq 384$, then $I \,||\, J \,||\, L = \texttt{BLAKE2b}(K)$, using an instance of the cryptographic hash function BLAKE2b [8] that produces 384-bit hash values.

Given two input tweaks $i, j$, the TBC $E_K^{i,j}$ is defined as follows:

| $i$ | $j$ | $E_K^{i,j}$ | $k$ |
|---|---|---|---|
| $-1$ | $\mathbb{N}$ | $E_K^{i,j} = \texttt{AES10}_k(X \oplus jJ)$ | $(0, I, J, L, I, J, L, I, J, L, I)$ |
| $0$ | $\mathbb{N}$ | $E_K^{i,j} = \texttt{AES4}_k(X \oplus jI)$ | $(0, J, I, L, 0)$ |
| $1$ | $\mathbb{N}$ | $E_K^{i,j} = \texttt{AES4}_k(X \oplus \delta_j I)$ | $(0, J, I, L, 0)$ |
| $2$ | $\mathbb{N}$ | $E_K^{i,j} = \texttt{AES4}_k(X \oplus \delta_j I)$ | $(0, L, I, J, L)$ |
| $\geq 3$ | $0$ | $E_K^{i,j} = \texttt{AES4}_k(X \oplus \delta_i L) \oplus \delta_i L$ | $(0, J, I, L, 0)$ |
| $\geq 3$ | $\mathbb{N}^*$ | $E_K^{i,j} = \texttt{AES4}_k(X \oplus \delta_i L \oplus \delta_j J) \oplus \delta_i L \oplus \delta_j J$ | $(0, J, I, L, 0)$ |

where $\delta_i = 2^{i-3}$ and $\delta_j = 8^{\lfloor (j-1)/8 \rfloor} + (j-1) \bmod 8$. In the former table, `AES4` (resp. `AES10`) are AES variants that consist of 4 (resp. 10) full AES rounds parametrized by 5 (resp. 11) independent sub-keys. Thus, if we denote the composition of `SubBytes`, `ShiftRows`, and `MixColumns` by `aesr` we get

$$\texttt{AES4}_k(X) = \texttt{aesr}(\texttt{aesr}(\texttt{aesr}(\texttt{aesr}(X \oplus k_0) \oplus k_1) \oplus k_2) \oplus k_3) \oplus k_4),$$

with $k = (k_0, k_1, k_2, k_3, k_4)$, `AES10` can be defined in the same way.

## 2.2 AEZ-hash universal hashing

To describe $\texttt{AEZ-hash}(K, T)$, we assume that the tweak $T = (\tau, N, A_1, \ldots, A_m)$, where $(A_1, \ldots, A_m)$ is a $m$-component vector. This allows to cover both the cases of string- and vector-valued associated data. Let us rewrite $T$ as $T = (T_1, \ldots, T_t)$, where $t = m + 2$. For each $m_i$-block component $T_i = B_{i,1} \ldots B_{i,m_i}$ of $T$, whose last block $B_{i,m_i}$ can be complete or incomplete, a partial hash value $\Delta_i$ is computed as follows:

$$\Delta_i = \begin{cases} E_K^{i+2,1}(B_{i,1}) \oplus E_K^{i+2,2}(B_{i,2}) \oplus \cdots \oplus E_K^{i+2,m_i-1}(B_{i,m_i-1}) \oplus E_K^{i+2,m_i}(B_{i,m_i}) & \text{if } |B_{m_i}| = 128 \\ E_K^{i+2,1}(B_{i,1}) \oplus E_K^{i+2,2}(B_{i,2}) \oplus \cdots \oplus E_K^{i+2,m_i-1}(B_{i,m_i-1}) \oplus E_K^{i+2,0}(B_{i,m_i} \,||\, 10^*) & \text{if } |B_{m_i}| < 128 \end{cases}.$$

Finally, $\texttt{AEZ-hash}(K, T) = \Delta \stackrel{\text{def}}{=} \Delta_1 \oplus \cdots \oplus \Delta_t$.

## 2.3 PRF Function

`AEZ-prf` is designed with the purpose to provide a PRF of settable output length $\tau$, that can be viewed as an encipherment of the empty plaintext. The output of `AEZ-prf` is the $\tau$-bit string given by

$$\texttt{AEZ-prf}(K, T, \tau) = (E_K^{-1,3}(\Delta) \,||\, E_K^{-1,3}(\Delta \oplus [1]_{128}) \,||\, E_K^{-1,3}(\Delta \oplus [2]_{128}) \,||\, \ldots)[1..\tau],$$

with $\Delta = \texttt{AEZ-hash}(K, T)$.

## 2.4 AEZ Core

`AEZ-core` is the encipherment function used to process augmented plaintexts of at least 256 bits. It takes as input the key $K$, the tweak vector $T$ and the augmented plaintext $\overline{P}$. The vector $T$ is first preprocessed by computing the universal hash value:

$$\Delta = \texttt{AEZ-hash}(K, T),$$

which will be used as an offset value at some subsequent steps of the encipherment computation.

Then, the augmented plaintext is split as follows into (in)complete 128-bit blocks:

$$\overline{P} = P_1 P_1' \,||\, P_2 P_2' \,||\, \cdots \,||\, P_m P_m' \,||\, P_u P_v \,||\, P_x P_y,$$

where $|P_*| = |P_*'| = 128$ except for at least one of the values $P_u$ and $P_v$, that satisfy $|P_u| + |P_v| < 256$. In detail, to split $\overline{P}$ one needs to

1. Take the 256 last bits of $\overline{P}$ to form $P_x P_y$. This is always possible since $|\overline{P}| \geq 256$;
2. For every remaining pair of entire blocks if any form $P_i P_i'$ (starting from the beginning of $\overline{P}$);
3. Letting $r = \mathit{plen} + \tau \bmod 256$, if $r \neq 0$, the remaining bits form
   - $P_u$ if $r < 128$,
   - $P_u P_v$ with an empty block $P_v$ if $r = 128$
   - $P_u P_v$ with $|P_v| = r \bmod 128$ if $128 < r < 256$.

The ciphertext blocks are then computed as shown on Figure 1, up to the fact that in the first case ($r < 128$), the $v$-column is omitted and the paddings and compressions represented by trapezoids on the $v$-column are moved to the $u$-column.

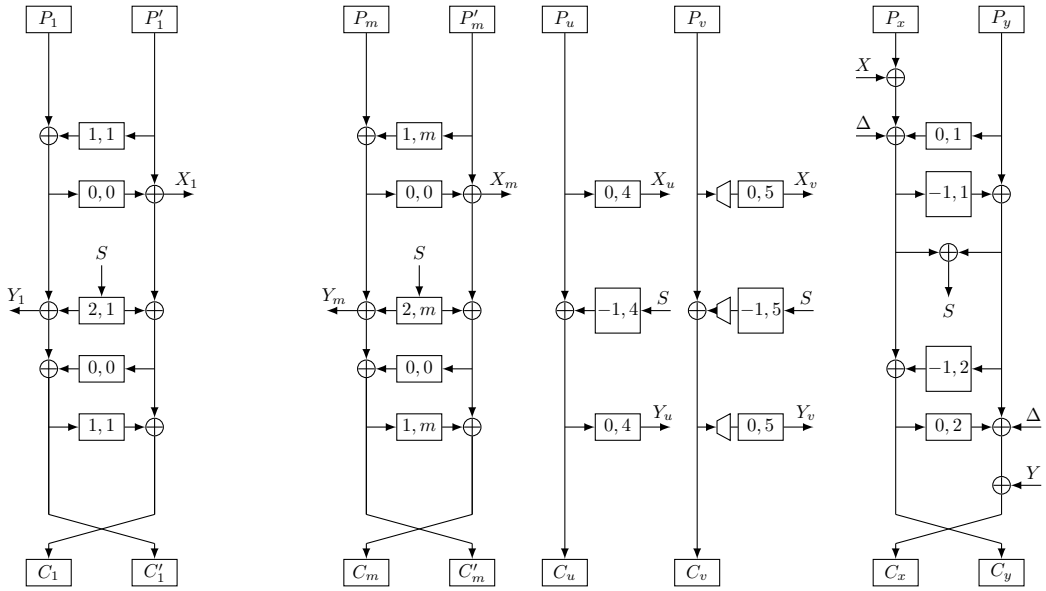


Figure 1: `AEZ-core` scheme.

$$
\begin{aligned}
\boxed{i,j} &= E_K^{i,j}(X) \\
X_i &= E_K^{0,0}(P_i \oplus E_K^{1,1}(P'_i)) \; i \in [1..m] \\
X_u &= E_K^{0,4}(P_u) \\
X_v &= E_K^{0,5}(P_v 10^*) \\
X &= X_1 \oplus \cdots \oplus X_m \oplus X_u \oplus X_v \\
S &= \Delta \oplus X \oplus E_K^{0,1}(P_y) \oplus E_K^{-1,1}(\Delta \oplus X \oplus E_K^{0,1}(P_y)) \oplus P_y \\
Y_i &= P_i \oplus E_K^{1,1}(P'_i) \oplus E_K^{2,1}(S) \; i \in [1..m] \\
Y_u &= E_K^{0,4}(P_u \oplus E_K^{-1,4}(S)) \\
Y_v &= E_K^{0,5}(P_v \oplus E_K^{-1,5}(S)[1..|P_v|] \;||\; 10^*) \\
Y &= Y_1 \oplus \cdots \oplus Y_m \oplus Y_u \oplus Y_v
\end{aligned}
$$

For a more detailed description of `AEZ-core` and more generally on AEZ v4.1, we refer to the AEZ v4.1 specification [3].

## 2.5 Tweaks from AEZ v3

In a nutshell, the main differences between AEZ v3 and AEZ v4.1 are the following:

– the procedure for deriving the subkeys $I$, $J$, and $L$ from the key $K$ was entirely modified. The AEZ v3 derivation procedure, that did not involve the BLAKE2b

hash function, had indeed the undesirable property that for key lengths such as $|K|$=128 bits, the knowledge of one of the subkeys implied the knowledge of the key $K$. Moreover, while a key length of at least 128 bits is recommended in both AEZ v3 and AEZ v4.1, a default key length of 384 bits was introduced in AEZ v4.1;

– the tweakable block ciphers involved in the `AEZ-hash` universal hashing use the XEX construction in AEZ v4.1, whereas they were using the XE construction in AEZ v3. Moreover the offset values used in the definition of the various tweakable block ciphers $E_K^{i,j}$ used in AEZ were modified.

One of the motivations for these changes was to thwart the birthday attack on AEZ v3 introduced by Fuhr, Leurent, and Suder in 2015 [6]. This attack indeed recovered one of the subkeys (namely $J$) by leveraging its use in the pre-whitening keys of the XE construction of the `AEZ-hash` computation underlying the `AEZ-prf` function. It then took advantage from the undesirable property of the AEZ v3 subkey derivation procedure mentioned above to recover the key $K$.

While the attack described by Fuhr et al. does not work anymore on AEZ v4.1, we will see in Section 3.1 that the use of the XEX construction in `AEZ-hash` does not prevent birthday attacks, and Section 3.2 will show that the knowledge of $I$ can be leveraged for recovering the other subkeys $J$ and $L$.

# 3 Attacks on AEZ

In this section, we describe two key derivation attacks:

– First, a combination of three independent birthday attacks allowing to retrieve one of the sub-keys $I$, $J$, and $L$ each. One limitation of this combined attack comes from the fact that the amount of data that can be processed under one single key is limited to $2^{48}$ bytes, below the $2^{64}$ blocks birthday bound. Its success probability, equal to the product of the success probabilities of the underlying birthday attacks, becomes in the case of a 128-bit key lower than the one of a generic attack.

– Second, a more efficient attack that consists of two phases. In the first phase, of birthday complexity, one of the three former partial attacks is applied to retrieve the value of $I$. In the second phase, the knowledge of $I$ is leveraged to mount a differential attack against some of the `AES4` instances of the encipherment computations. For any reasonable key length, e.g. at least 128 bits, its success probability remains abnormally high (i.e. higher than the one of a generic attack) if the amount of data that can be processed under one single key is limited to $2^{48}$ bytes.

## 3.1 Birthday Attacks

We describe in this subsection three partial attacks of birthday complexity each allowing to recover one of the three sub-keys.

All these partial attacks are based on the following informal observation. Let $F$ and $G$ denote two one-block to one-block functions parametrized by secret keys, $\delta_1$ and $\delta_2$ denote two secret one-block offset values and $n$ denote the block length. Let us assume that an adversary is able to access $H(x) = G(F(x \oplus \delta_1) \oplus F(x \oplus \delta_2))$ for sufficiently many chosen block values $x$. Let us show that if a small multiple of $2^{\frac{n}{2}}$ values of $x$ are tried this allows (under mild conditions on $F$ and $G$ that we will not detail here) to determine the secret

offset difference $\delta_1 \oplus \delta_2$ with an overwhelming probability. Indeed, with overwhelming probability, there exists a pair $(x, x')$ such that $x \oplus x' = \delta_1 \oplus \delta_2$. It is easy to see that for such a pair, $H(x) = H(x')$ since the single difference between the computations of $H(x)$ and $H(x')$ is that the entries of the first and second invocations of $F$ are swapped. Conversely, if $H(x) = H(x')$, $x \oplus x'$ provides a candidate value for $x \oplus x' = \delta_1 \oplus \delta_2$ that is easy to test using a few extra $H$ computations.

Note that all attacks presented below can be conducted under the assumption of a fixed nonce length, i.e. $nlen = 128$. We emphasize that the attacks can be transposed, with slight adjustements, to a situation where the nonce is omitted (i.e. $nlen = 0$).[4]

### 3.1.1 Collisions in `AEZ-hash`

Detecting suitable collisions in the `AEZ-hash` function allows to recover the two sub-keys $J$ and $L$ by birthday attacks. While `AEZ-hash` is an internal procedure whose ouput is not directly available to the adversary, such collisions on `AEZ-hash` can nevertheless be detected by collisions they induce in some `AEZ-prf` output blocks.[5] Let $\Delta$ be the output of `AEZ-hash` under an unknown key and a chosen entry

$$\Delta = \texttt{AEZ-hash}(K, T) \text{ with } T = (\tau, N, A)$$

where we assume that $|A| = 128$. For simplicity, we also assume $\tau = 128$ bits, but the attack also applies to others value of $\tau$.

Following the description of `AEZ-hash`, we get:

$$\Delta \quad = \quad E_K^{3,1}(\tau) \oplus E_K^{4,1}(N) \oplus E_K^{5,1}(A).$$

By replacing $E_K^{i,j}(X)$ by its expression we obtain:

$$\Delta \;=\; \texttt{AES4}_K(\tau \oplus L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(A \oplus 3L \oplus 8J) \oplus 8J.$$

If we restrict ourselves to $(A, N)$ pairs of blocks such that $A = N$, the former expression becomes

$$\Delta \;=\; \texttt{AES4}_K(\tau \oplus L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus 3L \oplus 8J) \oplus 8J.$$

With this expression, we are able to create a collision on hash values $\Delta$ associated to $(N, N)$ pairs and this can be used to retrieve the difference $L$ between the first and second offset values applied to $N$. Indeed, if $N' = N \oplus L$, let us denote by $\Delta'$ the associated hash value. We have:

$$\begin{aligned} & \texttt{AES4}_K(N' \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(N' \oplus 3L \oplus 8J) \\ = \;& \texttt{AES4}_K(N \oplus L \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus L \oplus 3L \oplus 8J) \\ = \;& \texttt{AES4}_K(N \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus 3L \oplus 8J). \end{aligned}$$

[4] The assumption that $nlen = 0$ was used in the AEZ v3 attack of [6].

[5] Collisions on `AEZ-hash` could alternatively be detected by collisions they induce on some `AEZ-core` output blocks. We will however not detail this slight variant here.

Hence, when $N' = N \oplus L$, we have $\Delta = \Delta'$. Note that this can be viewed as a direct consequence from the former observation. Indeed, in the former expressions of $\Delta$, $N$ is added with the offsets $\delta_1 = 2L \oplus 8J$ and $\delta_2 = 3L \oplus 8J$, of difference $L$, before being input to the function $F = \texttt{AES4}_K$.

#### Recovering the Sub-key $L$

The former remark allows to build the following birthday attack:

1. Collect $H(N) = \texttt{AEZ-prf}(K, T, \tau)$ with $T = (\tau, N, N)$ for $2^{64}$ values of $N$,
2. If a collision occurs, this implies $H(N') = H(N)$ since `AEZ-prf` is just an overencryption of the `AEZ-hash` output and therefore $L$ is likely to be equal to $N \oplus N'$.

For this attack we need about $2^{64.2}$ pairs $(N, A = N)$ of input blocks to succeed with a probability of about 0.5. If the amount of input data is restricted to the limit of $2^{44}$ blocks imposed by the designers, the success probability drops to $2^{-43}$.

#### Recovering the Sub-key $J$

The previous method can be used to retrieve $J$ in a nonce-misuse scenario where a fixed nonce value $N$ is repeated (which should result in no security degradation if the other AEZ input data are not repeated since an optimal nonce-misuse resistance is claimed). Indeed one can remark that using the previous notation except letting

$$T \quad = \quad (\tau, N, A, A),$$

where $N$ is a fixed nonce value of length 128 bits, $A$ is a variable one-block string, and the default value of 128 bits is assumed for $\tau$. Using the description of `AEZ-hash` one can write

$$\begin{aligned}\Delta \quad = \quad & \texttt{AES4}_K(\tau \oplus L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(A \oplus 3L \oplus 8J) \\ & \oplus \texttt{AES4}_K(A \oplus 3L \oplus 9J) \oplus 3L \oplus J.\end{aligned}$$

.

Hence if $A' = A \oplus J$, one obtains

$$\begin{aligned} & \texttt{AES4}_K(A' \oplus 3L \oplus 8J) \oplus \texttt{AES4}_K(A' \oplus 3L \oplus 9J) \\ = \quad & \texttt{AES4}_K(A \oplus J \oplus 3L \oplus 8J) \oplus \texttt{AES4}_K(A \oplus J \oplus 3L \oplus 9J) \\ = \quad & \texttt{AES4}_K(A \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(A \oplus 3L \oplus 9J), \end{aligned}$$

which implies $\Delta = \Delta'$. One can then build the following attack to recover $J$.

1. Collect $H(A) = \texttt{AEZ-prf}(K, T, \tau)$ with $T = (\tau, N, A, A)$ for $2^{64}$ values of $A$,
2. If a collision happens, this implies $H(A') = H(A)$ and therefore $J$ is likely to be equal to $A \oplus A'$.

To reach $2^{64.2}$ queries and a probability of success of about 0.5 we need to run the algorithm with $2^{65.8}$ blocks of data. If the amount of input data is restricted to $2^{44}$ blocks, the recovery of $J$ succeeds with probability $2^{-44.2}$.

### 3.1.2 Collision in AEZ-core

The last sub-key that remains unknown is $I$. We show how to recover it using a birthday attack within the AEZ-core function.

Let us consider 6-block augmented plaintexts:

$$\overline{P} = \overbrace{0^{128} \,||\, B}^{P_1, P_1'} \,||\, \overbrace{0^{128} \,||\, B}^{P_2, P_2'} \,||\, \overbrace{0^{128} \,||\, 0^{\tau}}^{P_x, P_y},$$

where $B$ denotes a one-block string and $\tau = 128$ bits. In this partial attack, we assume that a fixed nonce value $N$ and no associated data are used. Note that plaintext messages of length only five blocks are being used since the last block $0^{\tau}$ corresponds to the ciphertext expansion.

Next, we denote by $X$ the intermediate value associated with the two first pairs of blocks, that is used as an offset in the $P_x, P_y$ part. We remind that $X = X_1 \oplus \cdots \oplus X_m$ in general. In our case, we have $X = X_1 \oplus X_2$ which, once developed, becomes

$$X = E_K^{0,0}(E_K^{1,1}(B)) \oplus E_K^{0,0}(E_K^{1,2}(B)) \oplus B \oplus B.$$

We can rewrite this expression as

$$X = \mathtt{AES4}_K(\mathtt{AES4}_K(B \oplus 8I)) \oplus \mathtt{AES4}_K(\mathtt{AES4}_K(B \oplus 9I)),$$

and notice that if $B' = B \oplus I$ then $X = X'$. This leads to a similar attack to the one on $J$ and $L$. Indeed one can remark that collisions on the value of $X$ induce collisions in the value of $C_y$ since the single difference affecting the $(P_x, P_y)$ part of the computation is the introduction of distinct values $Y$ and $Y'$, that only affect the value of $C_x$, not the value of $C_y$.

In summary, we search for collisions on the value of $C_y$ to detect collisions on $X$ (as shown in Figure 2).

The following steps describe the attack exploiting the preceding remark.

1. Collect $C_{y,B}$ from the encipherments $\mathtt{AEZ\text{-}core}(K, T, \overline{P})$ associated to $2^{64}$ values of $B$.
2. If a collision occurs, i.e. $C_{y,B} = C_{y,B'}$, then $I$ is likely to be equal to $B \oplus B'$ and this is easy to test using another value of $B$.

We need to encrypt $2^{66.3}$ blocks of data to expect a collision with probability 0.5. If the amount of input data is restricted to $2^{44}$ blocks, the recovery of $I$ succeeds with probability $2^{-45.6}$.

### 3.1.3 Summary of Birthday Attacks

We have presented three partial attacks, each allowing to recover one of the three sub-keys $I$, $J$, and $L$. The following table summarizes the data complexity required by each attack for a success probability of 0.5 and their success probabilities if the amount of input data is limited to $2^{44}$ blocks. The time complexity of these attacks is equal to the time complexity for a single query multiplied by the query complexity.

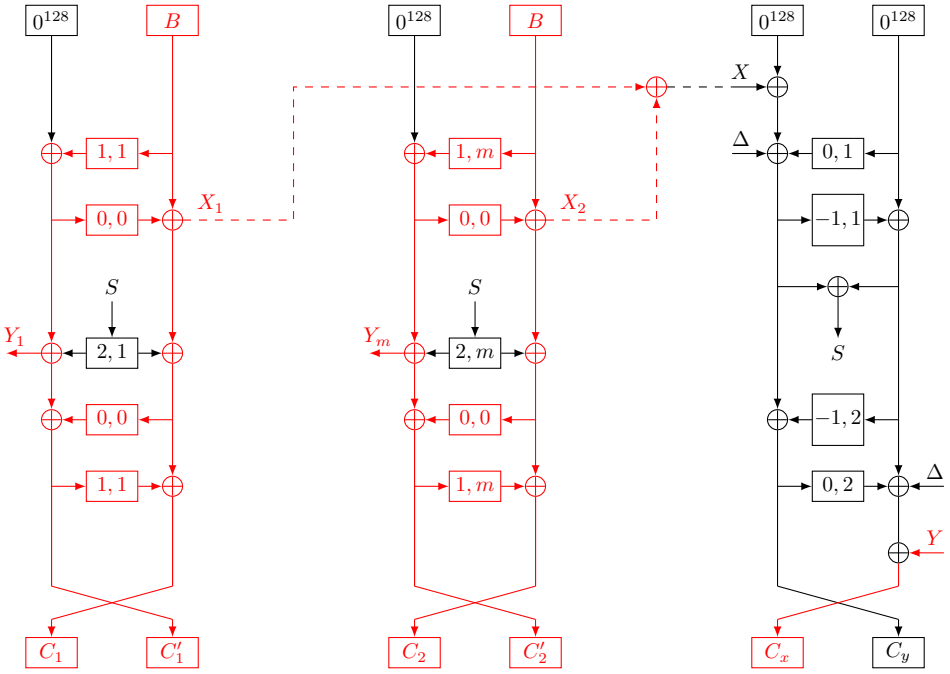


Figure 2: Difference propagation in the birthday attack to retrieve $I$.

The former partial attacks can be combined to recover the three sub-keys. However, when restricted to the encryption of $2^{44}$ blocks, this combined attack succeeds, in the case of a 128-bit key, with a lower probability than a classical brute-force attack.

Table 2: Birthday attacks complexities.

| **Retrieved key** | **Data complexity (blocks)**[6] | **Queries** | **Success probability** |
|---|---|---|---|
| $I$ | $2^{66.5}$ | $2^{64.2}$ | 0.5 |
| $I$ | $2^{44}$ | $2^{41.7}$ | $2^{-45.6}$ |
| $J$ | $2^{65.8}$ | $2^{64.2}$ | 0.5 |
| $J$ | $2^{44}$ | $2^{42.4}$ | $2^{-44.2}$ |
| $L$ | $2^{65.2}$ | $2^{64.2}$ | 0.5 |
| $L$ | $2^{44}$ | $2^{43}$ | $2^{-43}$ |
| $I, J, L$[7] | $2^{68.1}$ | $2^{65.8}$ | 0.5 |
| $I, J, L$ | $2^{44}$ | $2^{42.5}$ | $2^{-142.4}$ |

[6] Chosen plaintexts

[7] In this row, the data and query complexities were derived as the sum of the data (resp. query) complexities for recovering $I$, $J$, and $L$ with a probability of $0.5^{1/3}$ in three birthday attacks, as to ensure that the success probability is 0.5 for the combined attack.

## 3.2 AES4 Cryptanalysis

We previously described partial attacks allowing to recover one of the three sub-keys used in AEZ and a resulting combined attack. We now describe an attack which, assuming the sub-key $I$ has been retrieved using the last partial attack presented before, allows to efficiently recover the two others sub-keys $J$ and $L$.

As in the partial attack allowing to recover $I$, we assume that $\tau = 128$ and we use a fixed nonce value $N$ and no associated data in all considered encryptions.

### 3.2.1 Conducting Idea

Mounting a differential attack that targets the first AES4 encryption of the $P_u$ part of the AEZ-core function allows to leverage the knowledge of $I$ to recover $J$ and $L$. Since $I$ is known, this eventually boils down to attacking only three AES rounds instead of four. Although the differential cryptanalysis of 3-round AES is simple and well studied, the context of the attack for AEZ is more constrained and requires dedicated analysis. We therefore describe in some detail how to take these constraints into account.

Let

$$\overline{P} = P_u \,||\, \underbrace{0^{128} \,||\, 0^{\tau}}_{P_x, P_y},$$

where $P_u$ denotes a 128-bit block. Since $(plen + \tau) \bmod 256 = 128$, an empty block $P_v$ is introduced in the computation of $X$, that we denote by $R = E_K^{0,5}(1 \,||\, 0^{127})$.The resulting offset value $X$ is:

$$\begin{aligned} X &= E_K^{0,4}(P_u) \oplus R \\ &= \texttt{AES4}_K(P_u \oplus 4I) \oplus R. \end{aligned}$$

The detailed computation of $X$ is summarized in Figure 3.

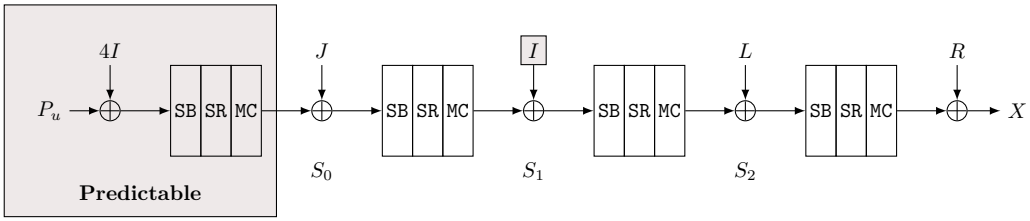


Figure 3: AES4 scheme.

To obtain information on $J$ and $L$, one can search pairs of $P_u$ values whose differential behaviour in the three last rounds of the AES4 computation follows a 4-1-4 differential characteristic. In other words, at the input to the second, third, and fourth rounds, we want the associated state values to only differ on 4 bytes, resp. 1 and 4 bytes, as shown in Figure 4.[8]

[8]We would like to acknowledge the work of Jérémy Jean with his repository "TikZ for Cryptographers" [9] for providing inspiration for the AES figures.

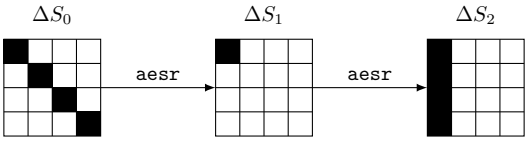


Figure 4: Differential path.

We are using the numbering convention of Figure 5 for the AES state bytes.

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

Figure 5: Bytes numbering in AES state.

Let us denote by $\Delta(x_1, \ldots, x_l)$ the vector space of difference values equal to zero everywhere outside from the byte positions $x_1, \ldots, x_l$. The expected differential behaviour at rounds 2, 3, and 4 is the following.

$$\begin{aligned}
\text{AES round 2}: &\quad \Delta(0,5,10,15) \xrightarrow{\texttt{SB SR}} \Delta(0,1,2,3) \xrightarrow{\texttt{MC}} \Delta(0) \\
\text{AES round 3}: &\quad \Delta(0) \xrightarrow{\texttt{SB SR MC}} \Delta(0,1,2,3) \\
\text{AES round 4}: &\quad \Delta(0,1,2,3) \xrightarrow{\texttt{SB SR}} \Delta(0,7,10,13) \xrightarrow{\texttt{MC}} \\
\text{Output}: &\quad \texttt{MC}(\Delta(0,7,10,13)).
\end{aligned}$$

Let $(S_0, S'_0)$ denote a pair of chosen second round input values before the addition of $J$, of difference $S_0 \oplus S'_0 \in \Delta(0,5,10,15))$ and $\delta_x$ denote a difference value from $\texttt{MC}(\Delta(0,7,10,13))$. $(S_0, S'_0)$ can be derived from the chosen pair of $P_u$ values $(P_u = \texttt{aesr}^{-1}(S_0) \oplus 4I, P'_u = \texttt{aesr}^{-1}(S'_0)) \oplus 4I)$ and one can test whether the differential behaviour of this pair is the desired one and the resulting `AES4` output difference is equal to $\delta_x$.

Indeed, let $(P_x, P'_x)$ be a pair of $P_x$ blocks of difference $P_x \oplus P'_x = \delta_x$ and $P_y = 0$, and let

$$\begin{aligned}
(C_v, C_x, C_y) &= \texttt{AEZ-core}(K, T, (P_u, P_x, P_y)) \\
(C'_v, C'_x, C'_y) &= \texttt{AEZ-core}(K, T, (P'_u, P'_x, P_y)).
\end{aligned}$$

Then we get $C_y = C'_y$ since the differences on the $X$ offset values and on the $P_x$ values cancel out. In Figure 6, the difference propagation is represented by the red pattern. Note that $C_y = C'_y$ happens with a negligible probability of about $2^{-128}$ if the tested condition is not met.

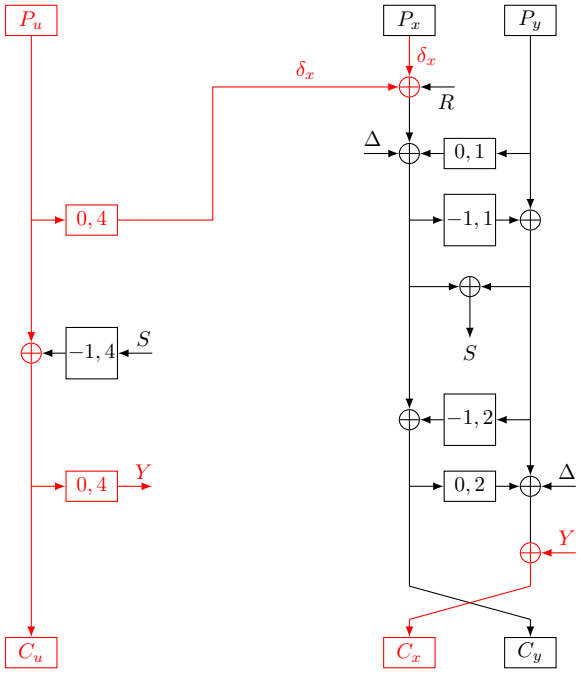


Figure 6: Difference propagation within `AEZ-core`.

We now show that the use of appropriate structures of $(P_u, P_x)$ values – obtained as the cartesian product of smaller structures of $P_u$ and $P_x$ values – allows to efficiently get $C_y = C'_y$ collision.

### 3.2.2 Structure of $P_u$ values

We want to test at least one pair $(S_0, S'_0)$ of difference value $S_0 \oplus S'_0 \in \Delta(0, 5, 10, 15)$ that leads after the second round to a difference value that belongs to the set $\Delta(0)$. A simple heuristic argument indicates that testing about $2^{24}$ $(S_0, S'_0)$ pairs should suffice for this to happen. Since picking $S_0$ and $S'_0$ from a subset of $\mathcal{S}$ of $\Delta(0, 5, 10, 15)$ of size $2^m$, $m \leq 16$, allows to cover approximately $2^{2m-1}$ such pairs, selecting a structure $\mathcal{S}$ of $2^{12.5}$ such values can be expected to suffice to obtain a good pair with a sufficient probability.

The resulting structure of $P_u$ values that we use in the sequel is $\mathcal{U} = \texttt{aesr}^{-1}(\mathcal{S}) \oplus 4I$. We expect at least one pair of $\mathcal{U}$ elements to have the expected differential behaviour.

### 3.2.3 Structures of $P_x$ and $P'_x$ values

At the output of the `AES4` function we want the difference to belong to the image of $\Delta(0, 7, 10, 13)$ by `MixColumns`. Since `MixColumns` is a linear operation, such a difference, denoted $\delta_x$, can be expressed as follows,

$$\delta_x = \delta_{x,(0,7)} \oplus \delta_{x,(10,13)} \quad \text{with} \quad \delta_{x,(0,7)} \in \texttt{MC}(\Delta(0,7)) \quad \text{and} \quad \delta_{x,(10,13)} \in \texttt{MC}(\Delta(10,13)).$$

This decomposition, in combination with the previous structure, allows to reduce by a squared factor the sets of tested $P_x$ and $P'_x$ values. This is explained in the next section.

### 3.2.4 How to Find a Good Pair

We can use cartesian products of the structures defined above to find a collision with an improved data complexity.

We encrypt the plaintexts associated with the two following structures of $(P_u, P_x)$ pairs:

$$\begin{aligned}(P_u, P_x) &\in \mathcal{U} \times \texttt{MC}(\Delta(0,7))\\ (P'_u, P'_x) &\in \mathcal{U} \times \texttt{MC}(\Delta(10,13)).\end{aligned}$$

We call *observation* the block $C_{y,P_u,P_x}$ resulting from the encryption of the plaintext $P_u \,||\, P_x \,||\, 0^{128}$. With the previous notations, one can remark that if $C_{y,P_u,P_x} = C_{y,P'_u,P'_x}$ then with overwhelming probability:

$$E_K^{0,4}(P_u) \oplus P_x = E_K^{0,4}(P'_u) \oplus P'_x$$

or equivalently

$$E_K^{0,4}(P_u) \oplus E_K^{0,4}(P'_u) = P'_x \oplus P'_x.$$

By construction, $P_u$ and $P'_u$ values are selected in such a way that the resulting round 2 input difference $\delta_{in} = \texttt{aesr}(P_u \oplus 4I) \oplus \texttt{aesr}(P_u \oplus 4I)$ belongs to $\Delta(0,5,10,15)$, and $P_x$ and $P'_x$ values were selected in such a way that their difference $\delta_{out} = P_x \oplus P'_x$ can take any value from $\Delta(0,7) \oplus \Delta(10,13) = \Delta(0,7,10,13)$.

Therefore we can expect at least one equality $C_{y,P_u,P_x} = C_{y,P'_u,P'_x}$ to happen and with overwhelming probability the underlying $(P_u, P'_u)$ pair is a good pair of second round input difference $\delta_{in}$ and fourth round output difference $\delta_{out}$.

On can also note that this method can be extended to the following differential patterns.

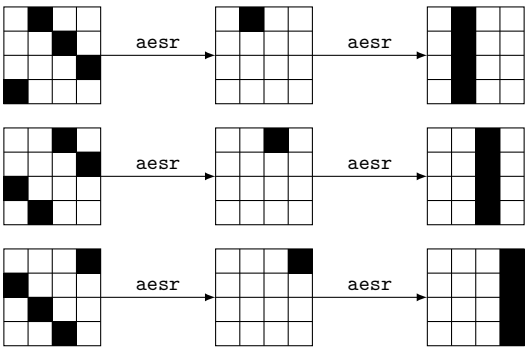


Figure 7: Other possible differential characteristics.

### 3.2.5 Sub-keys Recovery

Once a collision $C_{y,P_u,P_x} = C_{y,P'_u,P'_x}$ occurs, we obtain a good pair $(P_u, P'_u)$ with a known AES4 output difference $\delta_{out} = P_x \oplus P'_x$. This can be used to retrieve information on the sub-keys $J$ and $L$.

We know that the sub-key $J$ has the property to allow a differential transition

$$\Delta(0,5,10,15) \xrightarrow{\texttt{SB SR MC}} \Delta(0)$$

in the second round. Let us denote by $\overline{J}$ a candidate value for $J$ which leads to such a differential transition.

To each of the possible differences in $\Delta(0)$ we can associate a difference in $\Delta(0,5,10,15)$ by $\texttt{MC}^{-1} \circ \texttt{SR}^{-1}$. We denote such a difference by $\delta_{mid}$. There are 255 possible values $\delta_{mid}$.

Let $S_0 = \texttt{aesr}(P_u \oplus 4I)$, and $S'_0 = \texttt{aesr}(P'_u \oplus 4I) = S_0 \oplus \delta_{in}$. For a given trial value $\delta_{mid}$ we want to find $\overline{J}$ values that satisfy

$$\texttt{SB}(S_0 \oplus \overline{J} \oplus \delta_{in}) = \texttt{SB}(S_0 \oplus \overline{J}) \oplus \delta_{mid}.$$

With a variable substitution $X = S_0 \oplus \overline{J}$ this amounts to finding X such that

$$\texttt{SB}(X \oplus \delta_{in}) = \texttt{SB}(X) \oplus \delta_{mid}.$$

Let us denote by $B_i$ the $i$-th byte of a block $B$ and by $\texttt{sbox}$ the AES S-box. The former conditions amount to finding $X_0, X_5, X_{10}, X_{15}$ such that

$$\texttt{sbox}(X_i \oplus \delta_{in,i}) \quad = \quad \texttt{sbox}(X_i) \oplus \delta_{mid,i}, i = 0,5,10,15.$$

We know that at least one $\overline{J}$, the actual sub-key $J$, fulfils these conditions. But one can expect a larger set of candidates to fulfil these conditions, 275 in average (as shown in Appendix A) and another step is thus required to retrieve the right candidate. We expect to test about $275^4 = 2^{32.4}$ values to find $J$.

Assuming that we have collected the candidates for the four 4-byte parts of $J$, namely $\overline{J_i}$, we can retrieve the right value of $J$ by the following method (similar to the method used in the birthday attacks)

1. Compute all the $\Delta_i = \texttt{AEZ-prf}(K, (\tau, N, \overline{J_i}, \overline{J_i}), \tau)$ and the reference value $\Delta = \texttt{AEZ-prf}(K, (\tau, N, 0^{128}, 0^{128}), \tau)$.
2. Find the value $\Delta_m$ such that $\Delta_m = \Delta$. The right value for $J$ is then $J = \overline{J_m}$. This is due to the former observation used for the birthday attacks.

Once $J$ is recovered, one can apply a similar one-round differential technique to recover $L$. Indeed, for good pairs, the input values to the fourth round $S_2 = \texttt{aesr}(\texttt{aesr}(\texttt{aesr}(P_u \oplus 4I) \oplus J) \oplus I$ and $S'_2 = \texttt{aesr}(\texttt{aesr}(\texttt{aesr}(P'_u \oplus 4I) \oplus J) \oplus I$ are known, their difference $\delta_{mid} = S_2 \oplus S'_2 \in \Delta(0,1,2,3)$ is known, and the $\texttt{AES4}$ output difference $\delta_{out}$ is known. The latter difference induces a known difference value $(\texttt{SR}^{-1} \circ \texttt{MC}^{-1})(\delta_{out})$ after the fourth round $\texttt{SubBytes}$. Since the differences before and after $\texttt{SubBytes}$ are completely fixed, only about 16 candidates values in average will satisfy

$$\texttt{SB}(X \oplus \delta_{mid}) = \texttt{SB}(X) \oplus (\texttt{SR}^{-1} \circ \texttt{MC}^{-1})(\delta_{out}).$$

By using the other differential transitions the remaining 12 bytes of $L$ can be completely recovered and the sub-key $L$ is found by testing about $2^{16}$ candidates for $L$. A similar

method to the one used to find the right value of $J$, based on the former observation used for the birthday attack, can be used in order to find the right value of $L$.

### 3.2.6 Algorithm and Complexity to Recover $J$ and $L$

In summary, the following algorithm allows to find the values of $J$ and $L$ assuming that $I$ is known.

1. Compute the observations $C_{y,P_u,P_x}$ associated with all pairs $(P_u, P_x) \in \mathcal{U} \times \texttt{MC}(\Delta(0,7))$.
2. Compute the observations $C_{y,P'_u,P'_x}$ associated with all pairs $(P'_u, P'_x) \in \mathcal{U} \times \texttt{MC}(\Delta(10,13))$.
3. Find $(P_u, P'_u, P_x, P'_x)$ such that $C_{y,P_u,P_x} = C_{y,P'_u,P'_x}$ and compute $\delta_{in} = \texttt{aesr}(P_u \oplus 4I) \oplus \texttt{aesr}(P_u \oplus 4I)$, $\delta_{out} = P_x \oplus P'_x$.
4. Repeat Steps 1,2 and 3 for the three other differential transitions as to finally either get $(\delta^1_{in}, \delta^1_{out}), (\delta^2_{in}, \delta^2_{out}), (\delta^3_{in}, \delta^3_{out})$ or $(\delta^4_{in}, \delta^4_{out})$ for each good pair.
5. For each good pair, compute the about 275 candidate quartets of $J$ bytes that are compatible with $\delta^i_{in}$.
6. Test with `AEZ-prf` all the candidate values to find $J$.
7. For each good pair, compute the candidate quartets of $L$ bytes that are compatible with the $\delta^i_{out}$.
8. Test with `AEZ-prf` all the candidate values to find $L$.

To compute the complexity of our attack, we need to compute the cost of each step

- **Step 1 & 2** : We need to go through $\mathcal{U} \times \texttt{MC}(\Delta(0,7))$ and $\mathcal{U} \times \texttt{MC}(\Delta(10,13))$ to compute all the observations. This costs $2 \times |\mathcal{U}| \times |\texttt{MC}(\Delta(0,7))| = 2 \times 2^{12.5} \times 2^{16} = 2^{29.5}$ queries of 2 blocks i.e. $2^{31.5}$ blocks have to be encrypted.
- **Step 3** : Finding a collision can be achieved with a time complexity of about $2^{33.3}$. This a computational cost, so the query complexity is not affected.
- **Step 4** : $4 \times 2^{31.5} = 2^{33.5}$ blocks have to be encrypted.
- **Step 5** : With pre-computation of all solutions for $\texttt{SB}(X \oplus \delta_{in}) = \texttt{SB}(X) \oplus \delta_{mid}$ with any $\delta_{in}, \delta_{mid}$ the candidates are easily computed with a time complexity of $2^{24}$. As for Step 3 this phase does not require additional queries.
- **Step 6** : We need to compute $\texttt{AEZ-prf}(K, (\tau, X, X), \tau)$ for $275^4 = 2^{32.4}$ values of $X$ i.e. $2 \times 2^{32.4} = 2^{33.4}$ blocks have to be encrypted.
- **Step 7** : No more cost since the pre-computation used in Step 5 can be reused here.
- **Step 8** : We need to compute $\texttt{AEZ-prf}(K, (\tau, N, X, X), \tau)$ for $2^{16}$ values of $X$ i.e. $3 \times 2^{16} = 2^{16.6}$ blocks have to be encrypted.

The final cost to find $J$ and $L$ is given in Table 3 below.

Table 3: `AES4` attack complexities.

| Data complexity (bytes) | Offline time complexity | Queries complexity |
|---|---|---|
| $2^{34.6}$ | $2^{33.3}$ | $2^{32.1}$ |

This part of the attack was successfully validated on the public implementation of AEZ v4.1. This allowed to confirm that $J$ and $L$ can be recovered once $I$ has been recovered.

### 3.3 Results of Our Attack

As described our attack works in two phases : first, find the sub-keys $I$ by a birthday attack, and then, recover the two other sub-keys $J$ and $L$ by attacking `AES4`. Since the number of queries needed in the attack is far greater than the offline time complexity, the latter is insignificant in comparison of other costs and so, not included in the complexity of our attack. The final cost of our attack, depending on whether the data limit of $2^{48}$ bytes is respected or not, is given in the following Table 4

Table 4: Full attack complexities.

| Data complexity (blocks)[9] | Queries complexity | Success probability |
|---|---|---|
| $2^{44}$ | $2^{41.7}$ | $2^{-45.6}$ |
| $2^{66.5}$ | $2^{64.2}$ | 0.5 |

## 4 Conclusion

One of the purposes of the modifications between AEZ v3 and AEZ v4.1 was to fix an undesirable property allowing to recover the whole key from one of the sub-keys used in AEZ. Our paper shows that this property remains despite the changes. We show a key-recovery attack that allows to recover the three sub-keys from the knowledge of only one.

These modifications were also partly motivated by thwarting an attack of birthday complexity allowing to recover one of the subkeys. We described three birthday attacks on AEZ v 4.1 allowing to retrieve one of the three sub-keys.

Even though no claim for beyond birthday security has been made and our attack does not violate the security claims for AEZ, it raises some doubts regarding the resilience of AEZ against key-recovery attacks when the amount of processed data approaches the birthday bound.

[9] Chosen plaintexts

## Acknowledgements

The authors would like to thank Thomas Fuhr and Jérémy Jean for useful discussions on AEZ and insightful comments on former versions of this paper. This work was partially supported by the French Agence Nationale de la Recherche through the BRUTUS project under Contract ANR-14-CE28-0015.

## References

[1] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[2] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015.

[3] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. AEZ v4.1: Authenticated encryption by enciphering. http://web.cs.ucdavis.edu/~rogaway/aez/aez.pdf, 2015.

[4] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205. ACM, 2001.

[5] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 493–517, 2015.

[6] Thomas Fuhr, Gaëtan Leurent, and Valentin Suder. Collision attacks against CAESAR candidates - forgery and key-recovery against AEZ and Marble. In *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 510–532. Springer, 2015.

[7] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. *J. Cryptology*, 24(3):588–613, 2011.

[8] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. In *ACNS*, volume 7954 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2013.

[9] Jérémy Jean. TikZ for Cryptographers. http://www.iacr.org/authors/tikz/.

## A Computation of the Average Number of Candidates for a Quartet of Bytes of Sub-key $J$

If we let $a, b$ represent two non-zero random one-byte differences, then the equation $\texttt{sbox}(X \oplus a) = \texttt{sbox}(X) \oplus b$ may have 0, 2 or 4 solutions (`sbox` corresponds to the AES S-box). These number of solutions stand with their respective probabilities which are:

$$\#\{X \mid \texttt{sbox}(X \oplus a) = \texttt{sbox}(X) \oplus b\} = \begin{cases} 0 & \text{with } p_0 = \frac{128}{255} \\ 2 & \text{with } p_2 = \frac{126}{255} \\ 4 & \text{with } p_4 = \frac{1}{255} \end{cases}.$$

For a random pair $\delta = (\delta_1, \delta_2, \delta_3, \delta_4), \delta' = (\delta'_1, \delta'_2, \delta'_3, \delta'_4)$ of quartets of non-zero difference bytes, the average number of solutions of

$$\texttt{sbox}(X \oplus \delta_i) = \texttt{sbox}(X) \oplus \delta'_i \quad \text{for} \quad i = 1, 2, 3, 4$$

is given by

$$(2p_2 + 4p_4)^4 \simeq 1.015.$$

Hence, out of the 255 possible pairs $(\delta_{in}, \delta_{mid})$, we can expect 254 of them to bring an average of $254 \times 1.015 = 257.8$ candidates since they are not expected to exhibit the right guess on $J$. For the last one we know it will bring the right guess of $J$, at least one solution will be obtained. The previous expression for the average number of solutions has to be slightly modified and becomes

$$\left(2 \times \frac{126}{127} + 4 \times \frac{1}{127}\right)^4 \simeq 16.5.$$

The former heuristic reasoning shows that we can expect to have to test an average of about 275 candidates for each differential transition.