Fault analysis of AEZ

Hassan Qahur Al Mahri | Leonie Simpson | Harry Bartlett | Ed Dawson | Kenneth Koon-Ho Wong

Queensland University of Technology, Queensland, Australia

Correspondence
Hassan Qahur Al Mahri, George St, Brisbane 4000, Australia.
Email: hassan.mahri@hdr.qut.edu.au

Funding information
Government of the Sultanate of Oman

Summary
AEZ is a block cipher mode based on AES which uses three 128-bit keys. The algorithm has been updated several times during the three rounds of the CAESAR cryptographic competition. Cryptanalytic results presented on AEZ to date do not breach its security. This paper describes a fault injection analysis on AEZ. We focus on analyzing AEZ v4.2 but also investigate the applicability of these analyses to the recent version AEZ v5. This paper shows that all three 128-bit keys in AEZ v4.2 can be uniquely retrieved using only three random-valued single byte fault injections. A similar approach using four fault injections can uniquely recover all three keys of AEZ v5. The feasibility of this fault injection methodology has been proven against AES in previous works.

KEYWORDS
AEZ, authenticated encryption, block cipher mode, fault attack, side channel analysis

1 | INTRODUCTION

Cryptographic algorithms are designed so that they are secure against mathematical attacks. For all cryptographic algorithms, the brute-force attack, where the entire key space is searched to obtain the correct key, can be applied. A large key size can make this infeasible; where there is no attack faster than exhaustive key search, the algorithm is considered mathematically secure.

However, physical implementations of cryptographic algorithms may leak information about their internal operation. Attacks that exploit such leakage are called side channel analysis (SCA) attacks. Every cryptographic algorithm is vulnerable to such attacks even if the algorithm provides strong mathematical security. One common type of SCA attack is fault attack in which a fault is deliberately introduced into the operation of algorithm and the result observed.

AEZ is a mode of operation based on the AES block cipher, which provides authenticated encryption with associated data (AEAD). Currently, AEZ is a candidate in the third round of the CAESAR cryptographic competition. This scheme claims to provide strong security and usability properties, including flexible ciphertext expansion and nonce-reuse resistance.

Analysis of the security of AEZ against cryptographic attacks such as other works has limitations in relation to feasibility: most existing works require birthday complexity while the others do not invalidate the security claims of this scheme. There is no published analysis of AEZ security against side channel attacks. This article analyzes the security of AEZ against fault attacks as AEZ makes use of AES and fault attacks on AES are well known. Parts of this article were previously published in our paper.

Firstly, the proper place in AEZ scheme to directly apply these existing differential fault attacks is identified. Then, we consider the application of known fault attacks on AES to AEZ. This work shows that an attacker requires six faults to uniquely determine the three 128-bit keys in AEZ or five faults to reduce the key search space to $2^{24}$. After that, the application of the attack is further improved by reducing the number of fault injections required. Reducing the number of required faults saves time and makes the attack more feasible. The article considers different versions of AEZ, especially v4.2 and the most recent work v5.

This article is organised as follows: Section 2 briefly describes the AEZ scheme, while Section 3 gives a brief review of existing analysis on AEZ. Section 4 describes existing differential fault attacks on AES, especially Mukhopadhyay’s attack. Section 5 describes an approach to retrieve all three keys of AEZ v4.2 using minimal fault injections. This section includes also the steps needed to perform an experimental attack and presents a
further improvement to the proposed approach. Section 6 discusses the application of the presented fault approaches to AEZ v5. Section 7 presents the experimental results and compares the total number of fault injections for different approaches. Section 8 describes modifications to make fault attacks on AEZ more difficult to perform. Section 9 draws a conclusion.

2 | DESCRIPTION OF AEZ

In this section, we briefly describe the AEZ scheme, and refer the reader to related works\(^2\)\(^10\) for more technical details. AEZ\(^2\) is a block cipher mode that provides authenticated encryption and was submitted to the ongoing cryptographic competition CAESAR. The work in this article mainly focuses on AEZ v4.2, and v5. v4.2 was the latest version when this work was first conducted.\(^10\) At the moment of writing this article, AEZ is reaching v5. However, the analysis in this article is still applicable to AEZ v4 and v4.1 since these two versions are very similar to AEZ v4.2. We only provide the analysis of AEZ v4.2 and v5 to avoid redundancy. For simplicity, we omit the addendum “v4.2” and “v5” when referring to features common to both versions. We use the following notation and terminology:

- \(|X|\): the length of the string \(X\) in bits.
- \(X \| Y\): the concatenation of strings \(X\) and \(Y\).
- \(M\): plaintext message of arbitrary length.
- \(K\): \(k\)-bit key of default length \(|K| = 384\) and recommended that \(|K| \geq 128\).
- \(N\): the nonce and recommended that \(|N| \leq 128\). However, AEZ allows that \(|N| = 0\) and allows also different nonce lengths for different messages under the same key.
- \(A\): associated data of arbitrary length.
- \(ABYTES\): number of bytes of a fixed authentication block. Default \(ABYTES = 16\) and \(ABYTES \leq 16\) is recommended.
- \(r\): authenticator such that \(r = 8 \times ABYTES\). Authenticator measures the ciphertext expansion in bits over its plaintext.
- \(C\): the ciphertext output such that \(|C| = |M| + r\).
- \(SB()\): the AES SubBytes operation and \(SB^{-1}()\) represents the inverse operation.
- \(MC()\): the AES MixColumns operation and \(MC^{-1}()\) represents the inverse operation.
- \(AES4\): AES function consisting of four rounds and \(AES10\) represents an AES function consisting of ten rounds.

The block cipher encryption function mainly depends on two AES round functions: AES4 and AES10. Let the composition of the three main functions, SubBytes \((SB)\), ShiftRow \((SR)\) and MixColumns \((MC)\), used in the AES round function be defined as follows:

\[
aesr(X) = MC \circ SR \circ SB(X).
\]

AEZ is a four-round function of \(aesr\) as follows:

\[
AES4_X(K) = aesr(aesr(aesr(X \oplus K_0) \oplus K_1) \oplus K_2) \oplus K_3) \oplus K_4,
\]

where \(K = (K_0, K_1, K_2, K_3, K_4)\). AES10 is defined similarly but with ten rounds of \(aesr()\) instead of four. Note that this definition makes AES10 equivalent to the standard AES, except that AES10 applies the MixColumns operation in the last round (round 10) and uses a sequence built from the three 128-bit keys as round sub-keys.

The AEZ scheme provides several important features. Firstly, AEZ claims to provide nonce-reuse/misuse-resistant authenticated encryption (MRAE). According to Rogaway and Shrimpton,\(^11\) the confidentiality of an MRAE scheme is only compromised in that repeated messages of same \((N, A, M)\) can be detected when the nonce is repeated, whereas the integrity assurance is not affected at all. Secondly, AEZ can perform parallel operations such that the performance is comparable to AES in CTR mode. On the other hand, AEZ is not online, according to the definition of Bellare et al.\(^12\) In addition, AEZ is not easy to implement in hardware.\(^10\)

AEZ (see Figure 1) uses four different designs for processing the plaintext message and associated data as follows:

- **AEZ-hash**: this function uses the tweak \(T = (N, A, r)\) formed from the nonce \(N\), associated data \(A\) and authenticator \(r\) to create a mask \(\Delta\) that will be used in other modules, AEZ-prf, AEZ-tiny or AEZ-core.
- **AEZ-prf**: this module is used when \(|M| = 0\). It generates an output of length \(r\) that acts as an authentication tag.
- **AEZ-tiny**: this module is used when \(0 < |M| < 256 - r\). AEZ-tiny uses a balanced Feistel network where the number of rounds is between 8 and 24, depending on the length of the plaintext message. The round function is solely \(AES4\).
- **AEZ-core**: this module is used when \(|M| \geq 256 - r\). This module combines two modes, EME and OTR, in one design using both functions \(AES10\) and \(AES4\). Each pair of blocks requires 5 \(AES4\) calls which is equivalent to 10 AES rounds per block. Neither AEZ-core nor AEZ-tiny requires the AES-inverse operation. The work in this article considers only the AEZ-core scheme that is illustrated in Figure 2.
AEZ works by appending a fixed authentication block of zero-valued bytes (ABYTES) to the plaintext and encrypting the result using a mask \( \Delta \). This mask is generated from the function AEZ-hash that takes as input a mask \( T \) built from a nonce (\( N \)), associated data (\( A \)) and the authenticator (\( \tau \)). Integrity assurance is provided by decrypting the ciphertext and verifying the presence of the all-zero authentication block.

The AEZ secret key (\( K \)) is \( 3 \times 128 = 384 \) bits long and is arranged as: \( I \parallel J \parallel L \), where \(|I| = |J| = |L| = 128 \). However, if \(|K| < 384\), the key is transformed into 384-bit long string using the hash function BLAKE2b\(^{13}\) as follows:

\[
I \parallel J \parallel L = \begin{cases} 
K & \text{if } |K| = 384 \\
\text{BLAKE2b}(K) & \text{if } |K| \neq 384.
\end{cases}
\]

As noted above, AEZ-core uses two block encryption functions, AES4 and AES10. These are represented in Figure 2 as small rectangles (AES4) and larger squares (AES10) and are denoted in the cipher specification as \( E^j_i(X) \), where \( K \) is the key, \( X \) is the input string, and the parameters \( j, i \in \mathbb{N} \) specify which of the two functions (AES4 or AES10) is used and how the AES round subkeys are obtained from \( I, J, \) and \( L \). AEZ has been updated several times since the initial proposal was submitted to CAESAR. Section 2.1 and 2.2 highlight the changes between v4.2 and v5.
2.1 | AEZ v4.2

Version 4.2 is a minor revision from v4 and v4.1. For v4.2, the default key length is set to 384 bits and the key processing mechanism is changed to use the hash function BLAKE2b. Also, the encryption function $E_{K}^{i}(X)$ computes the ciphertext as follows:

$$E_{K}^{i}(X) = \begin{cases} N \times (i, l, l, l, l, l, l, l, l, l) & i = 0 \\ N \times (i, l, l, l, l, 0^{28}) & i = 1 \\ A \times (i, l, l, l, l, l, l, l, l, l) & i \geq 3 \end{cases}$$

where $\Delta_{i} = (2^{32} \cdot j - 1) + (i - 1 \mod 8)$.

2.2 | AEZ v5

This version changes the offsets used in $E_{K}^{i}(X)$. This revision is to avoid a forgery attack that is possible if the same offset is used in two different masks. The new definition of $E_{K}^{i}(X)$ is simplified as follows:

$$E_{K}^{i}(X) = \begin{cases} N \times (i, l, l, l, l, l, l, l, l, l) & i = 0 \\ A \times (i, l, l, l, l, l, l, l, l, l) & i \neq 1 \\ N \times (i, l, l, l, l, l, l, l, l, l) & i = 1 \\ A \times (i, l, l, l, l, l, l, l, l, l) & i \geq 3 \end{cases}$$

where $\Delta = j \cdot J \oplus 2^{28} \cdot l \oplus (i \mod 8) \cdot L$.

3 | REVIEW OF EXISTING ANALYSIS ON AEZ

The existing literature includes cryptanalysis of AEZ. There is a key recovery attack on v2 and v3, which is still applicable to v4. AEZ v4 is vulnerable to weak keys and has a serious bug that results in a forgery attack. These attacks are outlined below.

Fuhr et al. presented a generic key-recovery attack of birthday complexity against AEZ v2 and v3. These attacks are independent of the underlying permutation and work even if the round function is replaced by the full AES. The attack in the aforementioned work is applied to AEZ under the condition that either no nonce is used or the nonce is repeated. More importantly, the key derivation used in AEZ v2 and v3 allows the recovery of the master key ($K$) from the sub-key ($J$). However, this attack does not violate the security claim since the authors of AEZ do not claim beyond-birthday security. Nonetheless, this attack shows that AEZ is not a robust scheme; it collapses completely when the birthday bound is exceeded. Schemes usually present some resilience at birthday bound and do not allow full key recovery. AEZ was updated to v4 and the minor revision v4.1 to thwart the key recovery attack presented in the work of Fuhr et al. However, Chaigneau and Gilbert showed later that v4.1 is still vulnerable to such an attack. They presented a key recovery attack with birthday complexity that exploited the use of the four round function AES4 in AEZ. Assuming $I$ has been retrieved, a particular structure of plaintext in the last three rounds of AES4 allows an attacker to detect plaintext pairs that follow certain differential behaviour. These pairs are exploited to recover the $J$ and $L$ sub-keys.

Further, Mennink shows that AEZ v4 has weak keys, for which a distinguishing attack is possible. This observation on weak keys for AEZ is still valid to date. However, these weak keys represent only a tiny fraction of the entire key space, too small to be considered a practical breach.

AEZ v4 is also vulnerable to a forgery attack as a result of a mistake in mask instantiation. Bonnetain et al. found that the definition of masks in AEZ can lead to an overlap between two masks, which allows a simple forgery attack. Currently, AEZ has been updated to v5, which changes the definition of mask instantiation and simplifies it to avoid the forgery attack in the work Bonnetain et al. However, the key recovery attack with birthday complexity is still applicable and could not be thwarted.

4 | EXISTING FAULT ATTACKS ON AES

To the best of our knowledge, there are no existing fault attacks on AEZ in the public literature. However, there are several well-known differential fault attacks against AES. Differential fault attacks on AES were first introduced by Giraud. Giraud requires 250 pairs of correct and faulty ciphertexts where the fault disturbs one byte at the input of the 9th round. Subsequently, Dusart et al. show that 40 pairs of correct and faulty ciphertexts were sufficient to recover the secret key of AES by inducing a fault in a byte anywhere between the 8th and 9th round. Then, Piret and
Quisquater\textsuperscript{17} were able to reveal the AES key using only two ciphertext pairs, assuming the fault occurs between the 8\textsuperscript{th} round \texttt{MixColumns} and 9\textsuperscript{th} round \texttt{MixColumns}. Similarly, Mukhopadhyay presented an improved differential fault attack in the work of Mukhopadhyay\textsuperscript{7} that showed with two pairs of correct and faulty ciphertexts the AES key can be deduced. Additionally, Mukhopadhyay\textsuperscript{7} showed that one pair of correct/faulty ciphertexts can be used to deduce the AES key coupled with a brute-force search of $2^{32}$. This attack was further improved in the work of Tunstall et al\textsuperscript{8} to reduce the brute-force search from $2^{32}$ to $2^8$, again using one pair of correct and faulty ciphertexts. In this attack, the improvement is obtained by exploiting the linear relationship between AES sub-keys in the key schedule algorithm.

4.1 Mukhopadhyay’s fault attack

In this section, we review Mukhopadhyay’s differential fault attack against AES as we apply this repeatedly in our attack on AEZ. We choose this specifically because of the minimum number of fault injections required by this attack to recover the key and the simple analysis it follows using few algebraic equations. However, we expect similar results could be obtained with other known differential fault attacks on AES, especially in the attack of Piret and Quisquater\textsuperscript{17}. Piret and Quisquater’s attack is very similar to Mukhopadhyay’s attack: both work in last rounds of AES and both require only two faults to uniquely determine the key of AES.

The attack of Mukhopadhyay\textsuperscript{7} is a differential fault attack applied to the AES block cipher. This attack is based on inducing a random-valued single byte fault at the input of the eighth round. That is, a fault $f$, where $f \in \{1, \ldots, 255\}$, is induced into a single byte in the internal state of the block cipher during the encryption process. The XOR difference between the correct/faulty state matrices propagates in the last three rounds of AES as shown in Figure 3. This specific location enables the fault to be distributed to the entire state before producing the ciphertext message.

As Figure 3 shows, the faults at the first column of the state matrix at the input of the tenth round \texttt{SubBytes} is $2F_1, F_1, F_1, F_1$. Let $x_1, x_8, x_{11}$, and $x_{14}$ denote the correct ciphertext bytes and $x'_1, x'_8, x'_{11}$, and $x'_{14}$ represent the faulty ciphertext bytes. The corresponding bytes of the last sub-key are $K_1, K_8, K_{11}, K_{14}$. This pattern gives the following set of equations:

\[
2F_1 = SB^{-1}(x_1 \oplus K_1) \oplus SB^{-1}(x'_1 \oplus K_1) \\
F_1 = SB^{-1}(x_{14} \oplus K_{14}) \oplus SB^{-1}(x'_{14} \oplus K_{14}) \\
F_1 = SB^{-1}(x_{11} \oplus K_{11}) \oplus SB^{-1}(x'_{11} \oplus K_{11}) \\
3F_1 = SB^{-1}(x_8 \oplus K_8) \oplus SB^{-1}(x'_8 \oplus K_8),
\]

where $F_1, K_1, K_8, K_{11}$, and $K_{14}$ are all unknown values $\in \{0, \ldots, 255\}$.

Mukhopadhyay shows that this system of equations reduces the possibilities for $K_1, K_8, K_{11}$, and $K_{14}$ in the last sub-key of AES such that these key bytes can be uniquely determined with a probability of around 99% using only two correct/faulty ciphertext pairs. With only a single correct/faulty pair, the hypotheses for these four bytes in the last sub-key are reduced to $2^8$.

The same technique can be applied to the remaining columns to derive three more sets of equations. These sets cover the remaining bytes in the last sub-key. That is, Mukhopadhyay’s attack can uniquely determine the last sub-key in AES using two correct/faulty pairs or can reduce the key space to $2^{32}$ using only one correct/faulty pair, for which a brute-force search is feasible with current computation power.
Mukhopadhyay’s attack was further improved in the work of Tunstall et al., reducing the key search space to $2^8$ when a single byte fault is used. The improved attack employs the AES key schedule algorithm to reduce key hypotheses from $2^{24}$ to $2^8$. However, this improvement cannot be directly applied to AEZ since it uses three different keys $(I, J, L)$ that are not related.

Mukhopadhyay’s attack is very practical compared to most existing fault attacks. Mukhopadhyay’s attack uses a very weak assumption about the hardware fault required, assuming a random fault is induced to one byte at the input of the eighth round in AES. As a primary step, an attacker can predict the right location by analyzing other side channel information such as counting the number of clock edges or conducting a power scan. This single-byte random fault has been shown to be feasible using low-budget equipment. As an example, such fault was reported in the work of Selmane et al., obtained on experiments on a smart card integrated with an AES co-processor. This experiment successfully retrieved the AES key using Piret and Quisquater’s attack. This attack uses the same fault model as Mukhopadhyay’s attack. Hence, the approach outlined in this article can be easily verified.

5 | FAULT ATTACK ON AEZ V4.2

This work applies Mukhopadhyay’s attack to AEZ. As shown in Section 2, AEZ uses three unrelated 128-bit keys $(I, J, L)$. Thus, revealing one key is not sufficient to determine the other two keys. This suggests that the required number of pairs of correct and faulty ciphertexts for AEZ will be triple that required for attacks on AES. This research is motivated by this fact and investigates the structure of AEZ in order to decrease the required number of fault injections to increase the practicality of the fault attack. However, most of the attacks presented in this article do not apply if the light, efficient function AES4 is replaced by the more secure but heavier AES10 function.

AEZ claims to provide misuse-resistant authenticated encryption (MRAE) in which the scheme preserves optimal security when the nonce is repeated. This article applies fault attacks to AEZ in this specific case when the nonce is re-used. However, the authors of AEZ warn that omitting/ repeating the nonce is not allowed unless all encrypted pairs of associated data and messages under the same key are distinct from each other. Hence, these attacks nor the attacks in the work of Fuhr et al. do not invalidate the scheme security when the recommended specifications are followed.

For the rest of the article, assume that the attacker knows ciphertext messages all generated from the same plaintext. Assume also either the nonce is not used $(|N| = 0)$ or the $N$ value is fixed during the encryption of different messages (nonce repeated) since this is a usual requirement for most differential fault analysis. For simplicity, suppose the associated data and the value of $r$ are fixed. Suppose also $(|M| + r) \mod 256 = 128$. This implies that the $M_r$ block is an empty block and has a value of $(1 \parallel 0^{32})$.

5.1 | Direct application

In AEZ, we applied Mukhopadhyay’s fault attack to the AES10 encryption function in the $M_r$ part or $M_p$ part of the AEZ-core function. The $M_r$ and $M_p$ parts are the only two parts in AEZ where a direct output of the block cipher is known to the user. Hence, these two parts are the only suitable places for fault attacks. In this article, we target only the $M_r$ part, but the same results can be obtained if $M_p$ part is targeted instead.

Compared to the standard AES, AES10 applies the MixColumns operation to the last round. However, this has not prevented the Mukhopadhyay’s attack since MixColumns is a linear transformation.

The approach to uniquely determine $(I, L, J)$ keys is performed as follows:

1. **Recovery of $I$**: Inject two faults (ie, one at a time) to a single byte at the input of the eighth round in AES10. After that, the correct and two faulty ciphertext messages are used in Mukhopadhyay’s attack to recover the last sub-key $(I)$ in AES10. The entire $I$ key can be uniquely determined with $99\%$ probability of success.

2. **Recovery of $L$**: Since $I$ is recovered, we can easily invert the tenth round in AES10. Inject two additional faults to a single byte at the input of the seventh round in AES10 and use Mukhopadhyay’s attack to recover $L$. Again, these two faults will uniquely recover the entire $L$.

3. **Recovery of $J$**: Now, both $I$ and $L$ are known, so we can calculate both the tenth and ninth rounds. Similarly, inject the last two faults at the input of the sixth round and recover the last sub-key $J$.

This approach requires in total one correct and six faulty ciphertext messages to uniquely reveal all three 128-bit keys, $(I, L, J)$. However, if we inject only one fault instead of two in step 3, the $J$ key space will be reduced to $2^{24}$. According to Mukhopadhyay’s attack, faults in Step 2 will also uniquely recover four bytes in $MC^{-1}(J)$. That is, the key space for $J$ is further reduced to $2^{24}$ instead of $2^{32}$. However, if only one fault is induced in both step 2 and 3, $I$ will be uniquely determined, key space for $L$ will be $2^{24}$, but key space for $J$ will be $(2^{24} \times 2^{24} = 2^{56})$.

5.2 | Improved application

In this section, we describe an alternative approach that exploits the knowledge of $I$ and the AEZ structure to reduce the faulty ciphertexts from six to four pairs, with all keys $(I, J, L)$ still uniquely determined with the same probability ($99\%)$. That is, our proposed approach improves the direct application of Mukhopadhyay’s attack on AEZ by at least 33.3\%.
5.2.1 Recovery of \( I \)

We start our attack by assuming that \( I \) is determined using two correct/faulty pairs, as in Section 5.1. However, we use this attack slightly differently so that we can reuse the injected faults and minimize the overall faulty ciphertexts. Differential fault attacks on AES, such as related works\(^7,8,17\) usually use two correct ciphertexts of different plaintext and their corresponding faulty ciphertexts in order to reveal the last sub-key. In our approach, we use only one correct ciphertext and two faulty ciphertexts all of the same plaintext. This application makes the difference due to the fault injections affect only small parts in AEZ, as represented in red in Figure 2. We experimentally verified that this approach still gives the same result (i.e., uniquely determines \( I \)).

We now proceed with the rest of the technique given that \( I \) has already been revealed using three ciphertexts, one correct \((C_u)\) and two faulty \((C'_u, C''_u)\) all of the same plaintext \((M_{u1})\). Note that all these three ciphertexts \((C_u, C'_u, C''_u)\) are also inputs to the second AES4 in \( M_u \) part.

5.2.2 Recovery of \( Y_u \)

In this part, we exploit the use of AES4 in the \( M_u \) part to reduce the number of faulty ciphertexts required to determine the remaining two keys: \( J \) and \( L \). AES4 is a four round function that uses all three keys as depicted in Figure 4. Note that \( Y_u \) in AES4 is not XOR-ed with any key (i.e., \( Y_u \) is XOR-ed with a string of zeros 0\(^{128}\)). As a primary step, we consider \( Y_{u1} \), corresponding to the correct ciphertext \((C_u)\), is a sub-key and we need to reveal its value.

Changes in \( Y_u \) due to injected faults to AES10 (or the second AES4) can be observed from the change in \( C_u \) value, as shown in red in Figure 2. To uniquely reveal \( Y_{u1} \), two single-byte faults, one at a time, are needed. Note that these two additional faults are different from the faults during the process of \( I \) recovery and should not happen simultaneously. The faults should occur at the input of the second round of the second AES4 in the \( M_u \) part. Let the outputs of AES4 corresponding to these correct and two faulty blocks be \( Y_{u1}, Y'_{u1}, Y''_{u1} \), respectively, and \( C_{u1}, C'_{u1}, C''_{u1} \) be their corresponding outputs at \( C_u \) block.

Considering \( Y_{u1} \) as the last sub-key implies that the output is a block of zero bytes 0\(^{128}\). The output corresponding to fault injections can be represented as \( Y'_{u1} = Y_{u1} \oplus \Delta_1 \) and \( Y''_{u1} = Y_{u1} \oplus \Delta_2 \), where \( \Delta_1 = C_{u1} \oplus C'_{u1} \) and \( \Delta_2 = C_{u1} \oplus C''_{u1} \). That is, three output values can be obtained: 0\(^{128}\) is the correct ciphertext and \((\Delta_1, \Delta_2)\) are the two faulty ciphertexts such that all outputs share the same secret value \( Y_{u1} \). Now, we can again use Mukhopadhyay's attack to uniquely determine \( Y_{u1} \).

Note that retrieving the value of \( Y_{u1} \) enables us to retrieve both \( Y'_{u1} \) and \( Y''_{u1} \), and both \( Y'_{u2} \) and \( Y''_{u2} \), corresponding to faults during the recovery process of \( I \).

5.2.3 Recovery of \( J \) and \( L \)

At this stage, \( I \) and \( Y_{u1} \) are determined using four fault injections. That is, the second AES4 in the \( M_u \) part shrinks to two rounds instead of four, as shown in Figure 5. The attacker now knows three inputs to AES4 \((C_{u1}, C'_{u1}, C''_{u1})\) and their corresponding outputs \((Y_{u1}, Y'_{u2}, Y''_{u2})\), and the function is reduced to only two rounds. This information is sufficient to uniquely reveal the values of \( J \) and \( L \).

Cryptanalysis of two rounds of AES can be performed in different ways. We outline one approach to analyze the reduced form of AES4 to reveal both \( J \) and \( L \), as they are unrelated. A sketch of the reduced AES4 is depicted in Figure 6. Note that each diagonal of \( J \) collides with all columns of \( L \), each in a byte. For example, the diagonal \([0, 5, 10, 15]\) collides with the first column \([0, 1, 2, 3]\) in byte \( b_0 \). Similarly, it collides with the second column \([4, 5, 6, 7]\) in \( b_1 \) and so on.

We proceed to find four bytes \([0, 5, 10, 15]\) and four bytes \([0, 1, 2, 3]\) by making hypotheses on their values, thus allowing to calculate the collision points (black bytes in Figure 6). The hypotheses that lead to collisions amongst the three message pairs are the most likely ones. However, this approach needs a total of \( 2^{32} \times 2^{32} = 2^{64} \) hypotheses and has to be performed four times to retrieve the entire \( J \) and \( L \). Hence, the time complexity for this approach is high.

To overcome this high complexity, one can compute the black bytes under only \( 2^8 \) hypotheses of one column of \( L \) for each of the three pairs and then store these values. This has to be performed three more times each with a different \( L \) column. After that, the attacker searches \( 2^{32} \) hypotheses.
of one diagonal of the sub-key J and detects which value leads to collisions amongst the three message pairs. If the number of pairs is l, the complexity of this step is $2^{32} + 4l^2$. Simulations verify that if $l \geq 3$, this approach can uniquely determine a diagonal of J.

The same analysis applies to the remaining three diagonals of J; however, there is no need to re-compute the collision bytes as this step has been done during the recovery process of the first diagonal. The overall complexity of this approach is then $2^{34} + 2^{10}$, which is practical on a standard desktop computer. When the entire J is recovered, L can be easily determined since $Z_u$, J, J, and $Y_u'$ are all known.

### 5.3 Further improvement

In Section 5.2, we outlined an approach that uniquely retrieves the three 128-bit keys ($I$, $J$, $L$) in AEZ using only four faults: two to retrieve I and two to retrieve $Y_u$. We now further reduce the number of fault injections to three: two to retrieve I and one to retrieve $Y_u$. That is, the number of value hypotheses for $Y_u$ is $2^{32}$ as Mukhopadhyay’s attack shows. However, we have seen in Section 5.2.3 that the total time complexity to retrieve J and L given a unique value for I and $Y_u$ is $(2^{34} + 2^{10})$. Hence, if $Y_u$ has $2^{32}$ possible values, the overall complexity rapidly increases to $(2^{34} + 2^{121})$, which is impractical given a standard computational equipment.

We proceed by first reducing the possible values for $Y_u$ before applying the recovery process for ($J$, $L$) as outlined in Section 5.2.3. Note that L is the penultimate key in both AES10 and AES4 in the $M_r$ part. After retrieving I, one can use its value to inversely compute the state matrix at the input of the 9th round MixColumns in AES10, where $MC^{-1}(L)$ is the current sub-key. Due to the fault propagation, Mukhopadhyay’s attack can be applied to uniquely determine four bytes $MC^{-1}(L)[0, 7, 10, 13]$ using the faults pattern at the input of the ninth round ($2f', f', f', 3f'$) (see Figure 3). That is, using two fault injections to AES10, we can retrieve four bytes of $MC^{-1}(L)$ in addition to the recovery of the entire I.

Given four bytes $MC^{-1}(L)[0, 7, 10, 13]$ and $2^{32}$ possible values for ($Y_{u1}, Y_{u2}$), we can inversely compute four bytes of ($Z_{u1} \oplus J)[0, 5, 10, 15]$ and ($Z_{u1} \oplus J)[0, 5, 10, 15]$. Since both $Z_{u1}$ and $Z_{u2}$ are known, we can XOR these bytes with each other to check which value satisfies the differential ($Z_{u1} \oplus Z_{u2})[0, 5, 10, 15]$. We discard each ($Y_{u1}, Y_{u2}$) value that does not satisfy this differential property. Simulations show that only one value for ($Y_{u1}, Y_{u2}$) gets returned as a result. From here, the recovery process for ($J, L$) is the same as in Section 5.2.

## 6 FAULT ATTACK ON AEZ V5

This section highlights the changes in the most updated version of AEZ v5. We consider the feasibility of the attacks described in Section 5 on this current AEZ version.

### 6.1 Changes from v4.2

As discussed in Section 2.2, AEZ v5 changes how offsets are calculated in order to avoid a forgery attack. Specifically, it simplifies the offsets used in the function AES4 as follows:

$$AES4_{w}(X) = aesr(aesr(aesr(X \oplus \Delta) \oplus J) \oplus I) \oplus L) \oplus 0^{128},$$

where $\Delta = j \cdot J \oplus 2^{[8]} \cdot I \oplus (i \mod 8) \cdot L.$
Nonetheless, the changes in v5 do not completely protect AEZ against fault analysis.

6.2 | Implications of changes

The main idea in Section 5.2 is to first retrieve the values of I and Y, which in turn reduces the number of rounds in AES4 from four to two. This approach enables the attacker to determine the remaining sub-key values, J and L, using a guess-and-exclude method as previously described in the section. However, this approach cannot directly apply to AES4 in AEZ v5. Determining I and Y will not shrink the number of rounds as the first sub-key (2l @ 4l) involves now an additional key L. Hence, the fault attacks we applied to v4.2 cannot be used directly against this updated version. Nonetheless, the changes in v5 do not completely protect AEZ against fault analysis.

6.3 | Attack approach using four faults

We proceed using the fault analysis concept in Section 5.2, but using different steps in the recovery process. One alternative for the recovery process on AEZ v5 is performed as follows:

1. **Recovery of I**: As discussed in Section 5, the recovery process starts by inducing two faults at the input of the 8th round of the AES10 function in the M_p part and then applying Mukhopadhyay’s attack to uniquely determine the entire I key. In addition to recovering I, these faults can be further exploited to uniquely retrieve four bytes MC−1(L)[0, 7, 10, 13] of the inverse-MixColumns value of the penultimate key L in AES10. These four bytes will be used in the next steps to reduce the key search for both L and Y. Note that this step produces three known inputs (C_{I1}, C_{I2}, C_{I3}) to AES4 such that their corresponding outputs (currently still unknown) are (Y_{I1}, Y_{I2}, Y_{I3}), respectively.

2. **Recovery of L**: Inject an additional fault at the input of the 7th round of AES10. Since I is uniquely determined, the encryption state in AES10 can be inversely computed until the end of the 9th round, and then, Mukhopadhyay’s attack can be applied to reduce the possible values for L to 2^{32} only. For each of the 2^{32} candidate values for L, compute the four bytes MC−1(L)[0, 7, 10, 13] and compare the calculated value to the right value retrieved in step 1. This step reduces the possible values for L from 2^{32} to 2^{24}. This step also produces an additional pair (C_{I1}′, C_{I2}′) for AES4.

3. **Recovery of Y**: Similarly as in Section 5.3, inject one extra fault at the beginning of the 8th round of the lower AES4 function in the M_p part. This step will also provide the attacker with only 2^{25} possible values for Y.

4. **Reduce search space of Y**: Since L is the penultimate key in both AES10 and AES4, the 2^{22} possible values for Y can be used with the retrieved value for MC−1(L)[0, 7, 10, 13] in step 1 to check which candidate value follows the fault pattern at the beginning of the third round of the lower AES4 function. This step will significantly reduce the possible values for Y to 2^{9}. That is, till this step four faults have been induced, the entire I has been retrieved, (2^{23}) values for Y, and (2^{24}) values for L.

5. **Reduce search space of L**: Using the values for Y and the unique value for MC−1(L)[0, 7, 10, 13], the attacker can use AES4 to calculate back four bytes of the internal state prior to the SubBytes operation of the second round, namely (Z_{u1} \oplus J)[0, 5, 10, 15]. This generates only \( \sim 2^{13} \) possible values for (Z_{u1} \oplus J)[0, 5, 10, 15]. After that, one can proceed by computing the Z_{u1} value (see Figure 4) for C_{u1}. Since there are 2^{24} possible values for L, Z_{u1} will also have 2^{24} possible values. The attacker can repeat such process for the remaining pairs (C_{I1}′, C_{I2}′, C_{I3}) to calculate (Z_{u1,1}′ \oplus Z_{u1,2}′, Z_{u1,3}′). Now, we can XOR these bytes with each other to check which value satisfies the differential (Z_{u1} \oplus Z_{u1}′)[0, 5, 10, 15], (Z_{u1} \oplus Z_{u1}′)[0, 5, 10, 15] and (Z_{u1} \oplus Z_{u1}′)[0, 5, 10, 15]. We discard each (L, Y) value that does not satisfy this differential property. Simulations show that only one value for (L, Y) gets returned as a result. The complexity of this step is \( \sim 2^{24} \) which is still very feasible.

6. **Recovery of J**: By knowing all C, Y, I and L variables, J can be easily determined.

The attack approach in this section is very similar to the previous analysis in Section 5.2 and 5.3, but this approach exploits the fact that L is the penultimate key in both AES10 and AES4 to reduce the key search for both Y and L. This enables the attacker to significantly eliminate the rounds in AES4 and easily determine J. This approach is also applicable to AEZ v4.2 and can be considered as an alternative mechanism. In fact, this approach is much easier than the approach in Section 5.2 for AEZ v4.2 since L is not involved at the beginning of the first round of AES4.

7 | EXPERIMENTAL RESULTS AND COMPARISON

We successfully verified the procedure discussed in Sections 5 and 6 by simulating random fault injections in a software implementation of AEZ v4.2 in C language with the GNU GCC compiler on a standard desktop computer. Note that we did not perform hardware fault injections, but these have been demonstrated on AES by the experiments of Selmane et al.\(^{18}\) so we consider this to be a feasible approach. The summary of data, time, memory costs is presented in Table 1. Note that the offline time complexity and the required memory are specific to our implementation. Hence, these figures
may vary for more optimized implementations. The remaining key exhaustive search is implementation independent and shows whether the key is uniquely retrieved or still additional search is required by the attacker.

Comparing the improved procedures of Sections 5.2 to 6.3 to direct application of the Mukhopadhyay’s attack, detailed in Section 5.1, shows that the improved procedures require fewer fault injections in order to retrieve the three main AEZ keys (I, J, L). The first three lines in Table 1 show the required faulty encryptions for direct application of Mukhopadhyay’s attack on AEZ. The middle two lines show the corresponding requirements for the improved attacks outlined in Sections 5.2 and 5.3. Observe that the number of fault injections needed has been reduced, and the key is uniquely determined. The last line shows that four faults are required to uniquely determine all keys in AEZ v5. The number of faults cannot be further reduced since L is used in the first sub-key of AES4.

### 8 | POSSIBLE SOLUTIONS

Faults are injected to both AES10 and AES4 in AEZ, and then, we exploit the lightweight function AES4 in order to minimize the number of faults required to recover all three 128-bit keys, I, J, and L. The direct application of fault analysis on AES10, as discussed in Section 5.1, cannot be avoided with the current structure of M4 part in AEZ. To thwart this attack, AEZ structure would need to be changed so that the direct outputs of AES10 in M4 and M1 parts are not known to the user.

This section outlines two suggestions to avoid exploiting AES4 in order to minimize the number of faults. One intuitive solution is replacing AES4 function with the more secure function AES10. However, this suggestion burdens the scheme with extra computation that would affect the overall performance. Separate experiments are needed to compute the actual rate of performance degradation that would result in AEZ.

An alternative solution is to change the sequence of round keys in AES4. One possible definition for AES4 in M4 and M1, parts is as follows:

\[
AES4(X) = a_{esr}(a_{esr}(a_{esr}(X \oplus \Delta_1) \oplus \Delta_2) \oplus I) \oplus J) \oplus 0,
\]

where \(\Delta_1 = j \cdot J \oplus 2^{[16]} \cdot I \oplus (i \text{ mod } 8) \cdot L \) and \(\Delta_2 = L \oplus J\). Both ultimate and penultimate keys in AES10 and AES4 are different. This modification will not allow the attacker to shrink AES4 from four to two rounds so the attacks described in Sections 5 and 6 cannot be applied.

### 9 | CONCLUSION

This article presented the results of applying well-known fault attacks on AES to the block cipher mode of operation AEZ v4.2 and v5. AEZ uses a relatively large key of size 384 bits organized as three 128-bit keys, I, J, and L. Unlike the standard AES, the three 128-bit keys in AEZ are unrelated to each other. Hence, the number of fault injections required to retrieve all three keys is larger than for AES.

AEZ claims to be a misuse-resistant scheme where the nonce can be omitted or repeated. We demonstrated that Mukhopadhyay’s attack can be applied to AEZ if the nonce is either omitted or repeated. Direct application of this attack requires at least six fault injections to uniquely determine the three 128-bit keys.

We shown how the structure of AEZ v4.2 can be exploited to allow an improved attack, which reduces the number of fault injections to four while still uniquely determining all three keys. After that, we further reduced the number of fault injections to three through additional search of an intermediate value \(Y_{ij}\) in a precomputed table. This makes the attack more practical and simple as each extra fault injection requires time and cost. This result compares favourably to the existing attack on AES, which requires two faults to retrieve the 128-bit AES key, whereas our improved attack on AEZ requires only three faults to retrieve the entire 384-bit key.

Currently, AEZ has been updated to v5. However, direct application of Mukhopadhyay’s attack is still possible. Furthermore, we again proposed an improved attack that reduces the number of faults required for successful key retrieval from six to four.

Direct application of Mukhopadhyay’s attack on AEZ cannot be avoided unless the AEZ structure is changed. However, we pointed out that exploiting the light function AES4 to reduce the number of faults can be prevented. One way is to replace the AES4 function by AES10. Another approach is to change the sub-keys in AES4.

We stress that these attacks, as with all differential fault attacks, do not apply to AEZ if a nonce is changed for every message.

### TABLE 1 | Comparison of different attacks on AEZ

| Version | Approach | No. of Faulty Encryptions | Memory | Offline Time Complexity | Remaining Key Exhaustive Search |
|---------|----------|---------------------------|--------|------------------------|-------------------------------|
| v4.2 & v5 | Direct application (Sect. 5.1) | 6 | 4 KB | \(\approx 2^{24}\) | 0 |
|         |          | 5 | 4 KB | \(\approx 2^{24}\) | \(2^{24}\) |
|         |          | 4 | 4 KB | \(\approx 2^{24}\) | \(2^{26}\) |
| v4.2    | Improved application (Sect. 5.2) | 4 | 4 KB | \(\approx 2^{24}\) | 0 |
|         | Improved application (Sect. 5.3) | 3 | 16 KB | \(\approx 2^{25}\) | 0 |
| v5      | Improved application (Sect. 6.3) | 4 | 18 KB | \(\approx 2^{24}\) | 0 |
ACKNOWLEDGMENT

Hassan Qahur Al Mahri is supported by a scholarship funded by the Government of the Sultanate of Oman.

ORCID

Hassan Qahur Al Mahri https://orcid.org/0000-0001-8443-417X

REFERENCES

1. Kocher P. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Lecture Notes in Computer Science. Vol. 1109. Basel, Switzerland: Springer; 1996:104-113.

2. Hoang VT, Krotetz T, Rogaway P. Robust authenticated-encryption AEZ and the problem that it solves. Lecture Notes in Computer Science. Vol. 9056. Basel, Switzerland: Springer; 2015:15-44.

3. Bernstein D. Cryptographic competitions: CAESAR. http://competitions.cr.yp.to/caesar-submissions.html. Accessed October 11, 2017.

4. Fuhr T, Leurent G, Suder V. Collision attacks against CAESAR candidates: forgery and key-recovery against AEZ and marble. Lecture Notes in Computer Science. Vol. 9453. Basel, Switzerland: Springer; 2015:510-532.

5. Chaigneau C, Gilbert H. Is AEZ v4.1 sufficiently resilient against key-recovery attacks? IACR Trans Symmetric Cryptol. 2016;2016(1):114-133.

6. Mennink B. Weak keys for AEZ, and the external key padding attack. Lecture Notes in Computer Science. Vol. 10159. Basel, Switzerland: Springer; 2017:223-237.

7. Mukhopadhyay D. An improved fault based attack of the advanced encryption standard. Lecture Notes in Computer Science. Vol. 5580. Basel, Switzerland: Springer; 2009:421-434.

8. Tunstall M, Mukhopadhyay D, Ali S. Differential fault analysis of the advanced encryption standard using a single fault. Lecture Notes in Computer Science. Vol. 6633. Basel, Switzerland: Springer; 2011:224-233.

9. Al Mahri HQ, Simpson L, Bartlett H, Dawson E, Wong KK-H. A fault-based attack on AEZ v4.2. Paper presented at: 2017 IEEE Trustcom/BigDataSE/ICESS; 2017; Sydney, Australia.

10. Hoang V, Krotetz T, Rogaway P. AEZ V4.2: Authenticated encryption by enciphering. CAESAR submission. http://competitions.cr.yp.to/caesar-submissions.html. Accessed October 11, 2017.

11. Rogaway P, Shrimpton T. A provable-security treatment of the key-wrap problem. Lecture Notes in Computer Science. Vol. 4004. Basel, Switzerland: Springer; 2006:373-390.

12. Bellare M, Boldyreva A, Knudsen L, Namprempre C. Online ciphers and the hash-CBC construction. Lecture Notes in Computer Science. Vol. 2139. Basel, Switzerland: Springer; 2001:292-309.

13. Aumasson J-P, Neves S, Wilcox-O’Hearn Z, Winnerlein C. BLAKE2: Simpler, smaller, faster than MD5. Lecture Notes in Computer Science. Vol. 7954. Basel, Switzerland: Springer; 2013:119-135.

14. Bonnetain X, Derbez P, Duval S, et al. An easy attack on AEZ. https://www.nuee.nagoya-u.ac.jp/labs/tiwata/fse2017/slides/Rump-02.pdf. Accessed October 11, 2017.

15. Giraud C. DFA on AES. Lecture Notes in Computer Science. Vol. 3373. Basel, Switzerland: Springer; 2004:27-41.

16. Dusart P, Letourneux G, Vivolo O. Differential fault analysis on A.E.S. Lecture Notes in Computer Science. Vol. 2846. Basel, Switzerland: Springer; 2003:293-306.

17. Piret G, Quisquater J. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. Lecture Notes in Computer Science. Vol. 2779. Basel, Switzerland: Springer; 2003:77-88.

18. Selmane N, Guilley S, Danger J-L. Practical setup time violation attacks on AES. Paper presented at: Seventh European Dependable Computing Conference; 2008; Kaunas, Lithuania.

How to cite this article: Qahur Al Mahri H, Simpson L, Bartlett H, Dawson E, Wong KK-H. Fault analysis of AEZ. Concurrency Computat Pract Exper. 2019;31:e4785. https://doi.org/10.1002/cpe.4785