AEZ v4.2: Authenticated Encryption by Enciphering

Viet Tung Hoang
Florida State University
tvhoang@cs.fsu.edu

Ted Krovetz
Sacramento State
ted@krovetz.net

Phillip Rogaway
UC Davis
rogaway@cs.ucdavis.edu

September 15, 2016

The named authors are both designers and submitters.

Abstract

AEZ encrypts by appending to the plaintext a fixed authentication block and then enciphering the resulting string with an arbitrary-input-length blockcipher, this tweaked by the nonce and AD. The approach results in strong security and usability properties, including nonce-reuse misuse resistance, automatic exploitation of decryption-verified redundancy, and arbitrary, user-selectable ciphertext expansion. AEZ is parallelizable and its computational cost is close to that of AES-CTR. Our C implementation achieves peak speeds of 0.63 cpb on an Intel Skylake processor and 1.3 cpb on Apple's A9 ARM processor.

The latest version of this document, and all related materials, can always be found on the AEZ homepage: `http://www.cs.ucdavis.edu/~rogaway/aez`

Contents

0 Introduction

This document describes AEZ, which we view as both an enciphering scheme and an authenticated-encryption scheme. Before specifying it we provide a brief overview.

Authenticated encryption by enciphering. When we speak of an *enciphering scheme* we mean an object that is like a conventional blockcipher except that the plaintext's length is arbitrary and variable, and, additionally, there's a tweak. Regarding AEZ in this way, enciphering maps a key K, plaintext X, and tweak $\boldsymbol{T}$ to a ciphertext $Y = \text{Encipher}(K, \boldsymbol{T}, X)$ having the same length as X. Going backwards, one can recover $X = \text{Decipher}(K, \boldsymbol{T}, Y)$. The security property we seek is that of a tweakable, strong-PRP (pseudorandom permutation): for a random key K it should be hard to distinguish oracles $(\text{Encipher}(K, \cdot, \cdot), \text{Decipher}(K, \cdot, \cdot))$ from oracles $(\pi(\cdot, \cdot), \pi^{-1}(\cdot, \cdot))$ that realize a family of independent, uniformly random permutations and their inverse.

When we instead regard AEZ as an *authenticated-encryption* (AE) *scheme*, encryption maps key K, plaintext M, nonce N (also called a "public nonce" or "public message number"), associated data $\boldsymbol{A}$, and an authenticator length ABYTES to a ciphertext $C = \text{Encrypt}(K, N, \boldsymbol{A}, \tau, M)$ that is $\tau = 8 \cdot \text{ABYTES}$ bits longer than M. Calling $\text{Decrypt}(K, N, \boldsymbol{A}, \tau, C)$ returns either a string M or an indication of invalidity. The security property we seek is that of a *robust* authenticated-encryption (RAE) scheme [21], a new and very strong notion that implies protection of the privacy and authenticity of M and the authenticity of N and $\boldsymbol{A}$, and must do so to the maximal extent possible even if nonces get reused ("misuse resistance" [41]), the authenticator length is small (including zero), or if, on decryption, invalid plaintexts get prematurely released.

Why speak of enciphering when CAESAR is a competition for AE schemes? Because an enciphering scheme determines an AE scheme by a simple and generic transformation—the *encode-then-encipher* method—and the AE scheme one gets in this way has attractive security and usability properties.

Encode-then-encipher encrypts the string M by enciphering a string X that encodes both M and a block of ABYTES zero bytes, doing so using a tweak $\boldsymbol{T}$ that encodes N, $\boldsymbol{A}$, and ABYTES. Decryption works by deciphering the presented string (again using the tweak determined from N, $\boldsymbol{A}$, ABYTES) and verifying the presence of the anticipated zero bytes. See Figure 1.

What are these "attractive security and usability properties" to which we allude? (1) If plaintexts are known *a priori* not to repeat, no nonce is needed to ensure semantic security. (2) If there's arbitrary redundancy in plaintexts whose presence is verified on decryption, this augments authenticity. (3) Any authenticator length can be selected, achieving best-possible authenticity for this amount of expansion. (4) Because of the last two properties, one can minimize length-expansion for low-energy or bandwidth-constrained applications. (5) If what's supposed to be a nonce should accidentally get repeated, the privacy loss is limited to revealing repetitions in $(N, \boldsymbol{A}, M)$ tuples, while authenticity is not damaged at all. (6) If a decrypting party leaks some or all of a putative plaintext that was supposed to be squelched because of an authenticity-check failure, this won't compromise privacy or authenticity.

The authors believe that the properties just enumerated would sometimes be worth a considerable computational price. Yet, for software on capable platforms, the overhead we pay is modest: AEZ is about as fast as OCB.

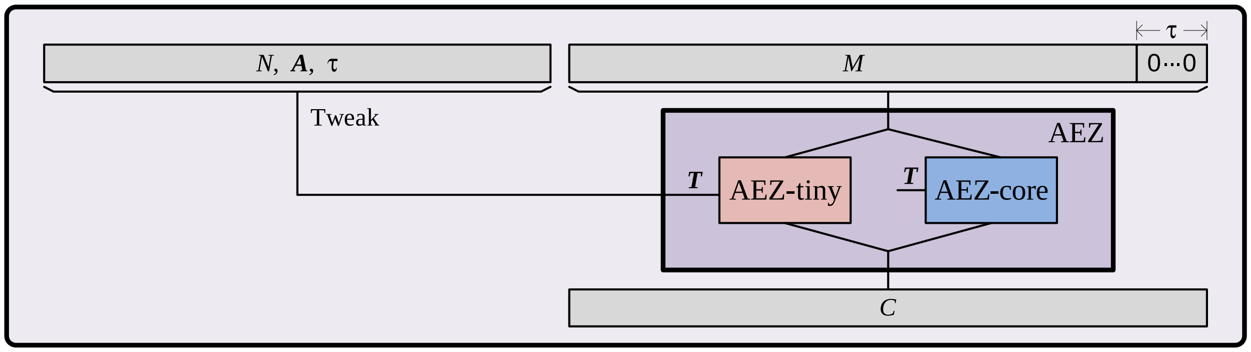

Figure 1: **High-level structure of AEZ**. After appending to the message a block of zeros we encipher it using a tweak comprising the nonce, associated data, and the ciphertext expansion $\tau = 8 \cdot \text{ABYTES}$. How this happens depends on the length of what's being enciphered: usually we use AEZ-core, but strings shorter than 32 bytes are enciphered by AEZ-tiny. Both depend on the underlying key K, which is not shown.

Realizing the enciphering. The way AEZ enciphers depends on the length of the plaintext. If it's fewer than 32 bytes we use AEZ-tiny, which builds on FFX [7, 15]. When it's 32 bytes or more we use AEZ-core, which builds on EME [19, 20] and OTR [29]. Again see Figure 1.

AEZ-tiny is a balanced-Feistel scheme. Its round function is based on AES4, a four-round version of AES. Guided by known attacks, more rounds are used for short strings than long ones.

AEZ-core resembles EME mode ("encipher-mask-encipher") but, for each of the two enciphering layers, consecutive pairs of blocks are processed together using a two-round Feistel network. The round function for this is based on AES4. The mask that is injected as the middle layer is determined, for each pair of blocks, using another AES4 call. The result is an enciphering scheme that employs five AES4 operations to encipher each consecutive pair of blocks, so 10 AES rounds per block. Thus our performance approaches that of AES-CTR.

The design of AEZ employs a paradigm we call *prove-then-prune*. One begins by designing a cryptographic scheme based on a well-known tool: for AEZ, a tweakable blockcipher (TBC) [27]. One proves security assuming that the tool meets some standard assumption; here, that the TBC is secure as a tweakable PRP. Finally, the tool—our TBC—is selectively instantiated by a *scaled-down* primitive: we will mostly use AES4, a reduced-round version of AES. The *thesis* underlying this approach is that it can be instrumental in finding complex but highly efficient schemes; and that if the instantiation is done judiciously, then the scaled-down scheme retains at least some assurance benefit flowing from the approach.

The name. The name "AEZ" is not exactly an acronym. The AE prefix is meant to suggest *authenticated encryption* and the overlapping EZ suffix is meant to suggest *easy*, in the sense of ease of correct use. The AES-like name is also a nod to the fact that AEZ is based on AES and can likewise be considered a species of blockcipher. Finally, the name can be used to identify individuals who can't distinguish an S from a Z.

Easy to use, not to implement. The easiness claim for AEZ is with respect to ease and versatility of use, not implementation. Writing software for AEZ is *not* easy, while doing a hardware design for AEZ is far worse. From the hardware designer's perspective, AEZ's name might seem ironic, the name better suggesting *anti-easy*, the *antithesis of easy*, or *anything-but easy*!

The difficulty with implementing AEZ in hardware stems from two sources. One is the sheer complexity of the mode, which glues together two unrelated methods, neither of which is pleasant in hardware. The more fundamental issue is that the RAE goal that AEZ targets simply can't be realized, for arbitrary-length messages, using a single-pass and a constant amount of memory. Hardware designs routinely expect this. An AEZ implementation must either cap the length of messages or employ an API and design that makes two passes. Additionally, a sensible hardware design would probably limit capabilities to AEZ-core, string-valued AD, and the default key length.

1 Specification

1.1 Notation

Numbers and strings. A *number* means a nonnegative integer, $\mathbb{N} = \{0, 1, 2, \ldots\}$. A bit is 0 or 1 and a string is a finite sequence of bits. The length of a string X is written $|X|$. The empty string ε is the string of length zero. Concatenation of strings A and B is written $A\,B$ or $A \parallel B$. When X is a string or a bit, X^n means X repeated n times; for example $0^3 = 000$ and $(01)^2 = 0101$. By $\mathcal{X}^*$ we denote the set of all strings over the alphabet $\mathcal{X}$, including ε. By $(\mathcal{X}^*)^*$ we denote the set of all vectors over $\mathcal{X}^*$, including the empty vector. The bitwise-and, bitwise-or, and bitwise-xor of strings A and B are denoted $A \wedge B$, $A \vee B$, and $A \oplus B$ respectively. For operations on unequal-length strings, first drop the necessary number of rightmost bits from the longer ($10 \oplus 0100 = 11$). For X a string, let $X0^* = X0^p$ with p the smallest number such that 128 divides $|X| + p$. If $|X| = n$ and $1 \le i \le j \le n$ then $X[i]$ is the ith bit of X (indexing from the left starting at 1), $\mathrm{msb}(X) = X[1]$, and $X[i..j] = X[i] \cdots X[j]$. Let $[n]_t$ be the t-bit string representing $n \bmod 2^t$ and let $[n]$ be shorthand for $[n]_8$; for example $[0]^{16} = ([0]_8)^{16} = 0^{128}$ and $[1]^{16} = (00000001)^{16}$. A byte is a string of eight bits. The set of all bytes is denoted $\textsc{Byte}$. A byte string is an element of $\textsc{Byte}^*$.

A block is 128 bits. Let $\mathbf{0} = 0^{128}$. If $X = a_1 \cdots a_{128}$ is a block ($a_i \in \{0,1\}$) then we define $X \ll 1 = a_2 \cdots a_{128}\, 0$. For $n \in \mathbb{N}$ and $X \in \{0,1\}^{128}$ define $n \cdot X$ by asserting that $0 \cdot X = \mathbf{0}$ and $1 \cdot X = X$ and $2 \cdot X = (X \ll 1) \oplus [135 \cdot \mathrm{msb}(X)]_{128}$ and $2n \cdot X = 2 \cdot (n \cdot X)$ and $(2n+1) \cdot X = (2n \cdot X) \oplus X$.

AES4 and AES10. We assume familiarity with AES. For $K, X \in \{0,1\}^{128}$ we write $\texttt{aesenc}(X, K)$ for a single round of AES: permute X by performing `SubBytes` then `ShiftRows` then `MixColumns`, then do an `AddRoundKey` with K. For $\boldsymbol{K} = (K_0, K_1, K_2, K_3, K_4)$ a list of five blocks let $\mathrm{AES4}_{\boldsymbol{K}}(X) = \mathrm{AES4}(\boldsymbol{K}, X)$ be

$$\texttt{aesenc}(\texttt{aesenc}(\texttt{aesenc}(\texttt{aesenc}(X \oplus K_0, K_1), K_2), K_3), K_4)\,.$$

For $\boldsymbol{K} = (K_0, K_1, \ldots, K_{10})$ a list of 11 blocks define $\mathrm{AES10}_{\boldsymbol{K}}(X) = \mathrm{AES10}(\boldsymbol{K}, X)$ like we defined AES4 but composed of ten rounds of `aesenc`. Note that we do *not* omit the final-round `MixColumns`, as does AES itself, for either AES4 or AES10.

BLAKE2b. AEZ requires a 48 byte key. If the user provides such a key, it is used directly, otherwise the provided key is transformed into 48 bytes using the cryptographic hash function

BLAKE2b [3]. All references to BLAKE2b in this document are specifically references to the unkeyed hash function named `id-blake2b384` in the BLAKE2 Internet Draft [43], which produces a 48-byte digest. For the sake of completeness, the BLAKE2b function is given in Appendix A.

1.2 Arguments and Parameters

By *parameter* we mean "a value on which AEZ encryption depends that, independent of any particular API, would usually to be held constant throughout some long-lived context." Under this interpretation of the word, AEZ has five arguments and one parameter. See Figure 2. In particular, we do not regard KEYBYTES = $|K|/8$ as a parameter (we permit keys of any length), nor NPUBBYTES = $|N|/8$ (we permit nonces to have varying lengths, even within a session). While these two values are omitted from the CAESAR-specified API, they could be specified in a different API.

The *authenticator length*, ABYTES, determines how much longer a ciphertext is than its plaintext. Arbitrary values are allowed, but values exceeding 16 are not expected to provide additional security. We do not insist that ABYTES be held constant throughout a session; a user is free to change it with each encryption. Still, we expect most applications to fix ABYTES, and choose to regard it as a parameter. We will use $\tau = 8 \cdot \text{ABYTES}$ if we want to measure ciphertext expansion in bits.

An unusual aspect of AEZ encryption is that it permits vector-valued AD: $\boldsymbol{A} \in (\text{Byte}^*)^*$. To recover the usual setting (a string-valued AD) the user selects an AD $\boldsymbol{A}$ with a single component.

We provide a default value for the ABYTES parameter: ABYTES = 16. The only named parameter set, denoted `aez`, uses this value. A conforming AEZ implementation is free to select a different default. In any context where the key length or nonce length are *required* to be fixed, we select byte lengths for these of KEYBYTES = 48 and NPUBBYTES = 12. Note that a default key length of 48-bytes does not mean that the designers are targeting 384-bits of security. We are not.

1.3 Targeted use cases

AEZ, for any choice of parameter values, primarily targets **use case 3: defense in depth.** This entails, for us: authenticity despite nonce misuse; limited privacy damage from nonce misuse; authenticity despite release of unverified plaintexts; limited privacy damage from release of unverified plaintexts. We do *not* aim for robustness in the case that AEZ is used on huge amounts of data (more data than what the specification document permits).

To a significant but lesser extent, AEZ also targets use case 2: high-performance applications. This entails, for us: high efficiency on machines with AES-NI capabilities; constant time execution when the message length is constant; messages sizes assumed to usually be fairly long.

1.4 AEZ Extensions

Early versions of AEZ included a parameter EXTNS, a string-valued *extensions directive.* The intent was that this would, in the future, unlock capabilities traditionally seen as outside the scope of an encryption scheme's functionality, including secret nonces (secret message numbers), plaintext-length obfuscation (via a specified padding regime), and encoding ciphertexts into a prescribed

symbol	comments
M	Plaintext. $M \in \text{BYTE}^*$
C	Ciphertext. $C \in \text{BYTE}^*$
K	Key. $K \in \text{BYTE}^*$. Recommend $\lvert K\rvert \geq 128$. Default is $\lvert K\rvert = 384$.
N	Nonce (aka: public sequence number). $N \in \text{BYTE}^*$. $\lvert N\rvert \leq 128$ recommended
$\boldsymbol{A}$	Associated data. $\boldsymbol{A} \in (\text{BYTE}^*)^*$. String-valued AD is regarded as a one-element vector
ABYTES	Authenticator length. ABYTES $\in \mathbb{N}$. Default is 16. ABYTES ≤ 16 recommended.

Figure 2: **Arguments and parameters to AEZ**. One might consider ABYTES an argument *or* a parameter: its value is allowed to change during the uses of a key, but conventionally this would not be done.

alphabet. These extensions are to be realized by a wrapper that keylessly transforms a plaintext, AEZ encrypts it, then keylessly transforms the result. We dropped the EXTNS parameter when we made the AD vector-valued, as the same effect can now be achieved using that feature. A document defining AEZ extensions will be released later.

1.5 Pseudocode

The definition of AEZ is provided in Figures 3 and 4. Let us explain some aspects of the pseudocode.

Encryption and decryption. To encrypt a string M we augment it with an *authenticator*—a block of ABYTES zero bytes—and encipher the resulting string, tweaking this with a tweak formed from $\boldsymbol{A}$, N, and ABYTES. Next we encipher the augmented message. To decrypt a ciphertext C we reverse the process, verifying the presence of the all-zero authenticator.

Enciphering. Messages are enciphered by either of two methods. Strings of 1–31 bytes are enciphered using AEZ-tiny, while those of 32 bytes or more are enciphered using AEZ-core.

Roughly following FFX [7, 15], AEZ-tiny uses a balanced Feistel network. The number of rounds depends on the length of the plaintext: as few as eight, or as many as 24. The round function is based on AES4. This is embodied in the pseudocode by the fact that our tweakable PRP decides to use AES4 or AES10 based on the first component of the tweak, employing the AES10 only for tweaks beginning with a -1. The Encipher-AEZ-tiny routine is illustrated at the bottom-right if Figure 5 for the setting where messages have 16 or more bytes.

A novel feature of AEZ-tiny is the possible xoring of a bit into the ciphertext just before the algorithm's conclusion. This is done to avoid simple random-permutation distinguishing attacks, for very short strings, based on the fact that Feistel networks only generate even permutations. A similar trick, conditionally swapping two fixed points, has been used before [36]. Our approach has the benefit that the natural implementation is constant-time.

The heart of AEZ is AEZ-core. It melds EME [19, 20], OTR [29], and a variety of other ideas. Consider the case where we want to encipher a string $M = M_1 M_1' \cdots M_m M_m' \, M_\mathsf{x} M_\mathsf{y}$ having an even number of blocks, all of them full. We call the first $2m$ blocks of M the i-blocks. Refer to the top-left

```
100 algorithm Encrypt($K, N, \boldsymbol{A}, \tau, M$)                // AEZ authenticated encryption
101     $X \leftarrow M \parallel 0^\tau$; $(A_1, \ldots, A_a) \leftarrow \boldsymbol{A}$
102     $\boldsymbol{T} \leftarrow ([\tau]_{128}, N, A_1, \ldots, A_a)$
103     if $M = \varepsilon$ then return AEZ-prf$(K, \boldsymbol{T}, \tau)$
104     return Encipher$(K, \boldsymbol{T}, X)$

110 algorithm Decrypt($K, N, \boldsymbol{A}, \tau, C$)                // AEZ authenticated decryption
111     $(A_1, \ldots, A_a) \leftarrow \boldsymbol{A}$;  $\boldsymbol{T} \leftarrow ([\tau]_{128}, N, A_1, \ldots, A_a)$
112     if $|C| < \tau$ then return $\bot$
113     if $|C| = \tau$ then if $C =$ AEZ-prf$(K, \boldsymbol{T}, \tau)$ then return $\varepsilon$ else return $\bot$ fi fi
114     $X \leftarrow$ Decipher$(K, \boldsymbol{T}, C)$
115     $M \parallel Z \leftarrow X$ where $|Z| = \tau$
116     if $(Z = 0^\tau)$ then return $M$ else return $\bot$

200 algorithm Encipher($K, \boldsymbol{T}, X$)                // AEZ enciphering
201     if $|X| < 256$ then return Encipher-AEZ-tiny$(K, \boldsymbol{T}, X)$
202     if $|X| \geq 256$ then return Encipher-AEZ-core$(K, \boldsymbol{T}, X)$

210 algorithm Encipher-AEZ-tiny($K, \boldsymbol{T}, M$)                // AEZ-tiny enciphering
211     $\mu \leftarrow |M|$;  $n \leftarrow \mu/2$;  $\Delta \leftarrow$ AEZ-hash$(K, \boldsymbol{T})$
212     if $\mu = 8$ then $k \leftarrow 24$ else if $\mu = 16$ then $k \leftarrow 16$ else if $\mu < 128$ then $k \leftarrow 10$ else $k \leftarrow 8$ fi
213     $L \leftarrow M[1..n]$;  $R \leftarrow M[n+1..\mu]$;  if $\mu \geq 128$ then $i \leftarrow 6$ else $i \leftarrow 7$ fi
214     for $j \leftarrow 0$ to $k-1$ do $R' \leftarrow L \oplus ((\mathrm{E}_K^{0,i}(\Delta \oplus R10^* \oplus [j]_{128}))[1..n])$;  $L \leftarrow R$;  $R \leftarrow R'$  od
215     $C \leftarrow R \parallel L$;  if $\mu < 128$ then $C \leftarrow C \oplus (\mathrm{E}_K^{0,3}(\Delta \oplus (C0^* \vee 10^*)) \wedge 10^*)$ fi
216     return $C$

220 algorithm Encipher-AEZ-core($K, \boldsymbol{T}, M$)                // AEZ-core enciphering
221     $\Delta \leftarrow$ AEZ-hash$(K, \boldsymbol{T})$
222     $M_1 M'_1 \cdots M_m M'_m\ M_{\mathsf{uv}}\ M_{\mathsf{x}} M_{\mathsf{y}} \leftarrow M$ where $|M_1| = \cdots = |M'_m| = |M_{\mathsf{x}}| = |M_{\mathsf{y}}| = 128$ and $|M_{\mathsf{uv}}| < 256$
223     $d \leftarrow |M_{\mathsf{uv}}|$;  if $d \leq 127$ then $M_{\mathsf{u}} \leftarrow M_{\mathsf{uv}}$; $M_{\mathsf{v}} \leftarrow \varepsilon$ else $M_{\mathsf{u}} \leftarrow M_{\mathsf{uv}}[1..128]$; $M_{\mathsf{v}} \leftarrow M_{\mathsf{uv}}[129..|M_{\mathsf{uv}}|]$ fi
224     for $i \leftarrow 1$ to $m$ do $W_i \leftarrow M_i \oplus \mathrm{E}_K^{1,i}(M'_i)$;  $X_i \leftarrow M'_i \oplus \mathrm{E}_K^{0,0}(W_i)$ od
225     if $d = 0$ then $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathbf{0}$ else if $d \leq 127$ then $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathrm{E}_K^{0,4}(M_{\mathsf{u}}10^*)$
226     else $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathrm{E}_K^{0,4}(M_{\mathsf{u}}) \oplus \mathrm{E}_K^{0,5}(M_{\mathsf{v}}10^*)$ fi
227     $S_{\mathsf{x}} \leftarrow M_{\mathsf{x}} \oplus \Delta \oplus X \oplus \mathrm{E}_K^{0,1}(M_{\mathsf{y}})$;  $S_{\mathsf{y}} \leftarrow M_{\mathsf{y}} \oplus \mathrm{E}_K^{-1,1}(S_{\mathsf{x}})$;  $S \leftarrow S_{\mathsf{x}} \oplus S_{\mathsf{y}}$
228     for $i \leftarrow 1$ to $m$ do $S' \leftarrow \mathrm{E}_K^{2,i}(S)$; $Y_i \leftarrow W_i \oplus S'$; $Z_i \leftarrow X_i \oplus S'$; $C'_i \leftarrow Y_i \oplus \mathrm{E}_K^{0,0}(Z_i)$; $C_i \leftarrow Z_i \oplus \mathrm{E}_K^{1,i}(C'_i)$ od
229     if $d = 0$ then $C_{\mathsf{u}} \leftarrow C_{\mathsf{v}} \leftarrow \varepsilon$;  $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathbf{0}$
230     else if $d \leq 127$ then $C_{\mathsf{u}} \leftarrow M_{\mathsf{u}} \oplus \mathrm{E}_K^{-1,4}(S)$;  $C_{\mathsf{v}} \leftarrow \varepsilon$;  $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathrm{E}_K^{0,4}(C_{\mathsf{u}}10^*)$
231     else $C_{\mathsf{u}} \leftarrow M_{\mathsf{u}} \oplus \mathrm{E}_K^{-1,4}(S)$;  $C_{\mathsf{v}} \leftarrow M_{\mathsf{v}} \oplus \mathrm{E}_K^{-1,5}(S)$;  $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathrm{E}_K^{0,4}(C_{\mathsf{u}}) \oplus \mathrm{E}_K^{0,5}(C_{\mathsf{v}}10^*)$ fi
232     $C_{\mathsf{y}} \leftarrow S_{\mathsf{x}} \oplus \mathrm{E}_K^{-1,2}(S_{\mathsf{y}})$;  $C_{\mathsf{x}} \leftarrow S_{\mathsf{y}} \oplus \Delta \oplus Y \oplus \mathrm{E}_K^{0,2}(C_{\mathsf{y}})$
233     return $C_1 C'_1 \cdots C_m C'_m\ C_{\mathsf{u}} C_{\mathsf{v}}\ C_{\mathsf{x}} C_{\mathsf{y}}$
```

Figure 3: **AEZ authenticated-encryption: main routines.** The tweakable blockcipher E, the hash AEZ-hash, and the PRF AEZ-prf are all defined in Figure 4. Requested ciphertext expansion is $\tau = 8 \cdot \textsc{abytes}$ bits. Algorithm Decipher$(K, \boldsymbol{T}, C)$, not shown, returns the unique M such that Encipher$(K, \boldsymbol{T}, M) = C$. See the accompanying text for how this is computed.

```
**algorithm** AEZ-hash($K, \boldsymbol{T}$)                         // AXU hash. $T$ is a vector of strings
$(T_1, \ldots, T_t) \leftarrow \boldsymbol{T}$
**for** $i \leftarrow 1$ **to** $t$ **do**
    $\ell \leftarrow \max(1, \lceil |T_i|/128 \rceil)$;  $j \leftarrow i + 2$;  $Z_1 \cdots Z_\ell \leftarrow T_i$ where $|Z_1| = \cdots = |Z_{\ell-1}| = 128$
    **if** $|Z_\ell| = 128$ **then** $\Delta_i \leftarrow \mathrm{E}_K^{j,1}(Z_1) \oplus \cdots \oplus \mathrm{E}_K^{j,\ell}(Z_\ell)$ **fi**
    **if** $|Z_\ell| < 128$ **then** $\Delta_i \leftarrow \mathrm{E}_K^{j,1}(Z_1) \oplus \cdots \oplus \mathrm{E}_K^{j,\ell-1}(Z_{\ell-1}) \oplus \mathrm{E}_K^{j,0}(Z_\ell 10^*)$ **fi**
**return** $\Delta_1 \oplus \cdots \oplus \Delta_t \oplus \mathbf{0}$
```

```
**algorithm** AEZ-prf($K, \boldsymbol{T}, \tau$)                        // PRF used when $M = \varepsilon$
$\Delta \leftarrow$ AEZ-hash($K, \boldsymbol{T}$)
**return** $(\mathrm{E}_K^{-1,3}(\Delta) \parallel \mathrm{E}_K^{-1,3}(\Delta \oplus [1]_{128}) \parallel \mathrm{E}_K^{-1,3}(\Delta \oplus [2]_{128}) \parallel \cdots)[1..\tau]$
```

```
**algorithm** $\mathrm{E}_K^{j,i}(X)$                                    // Scaled-down TBC
$I \parallel J \parallel L \leftarrow \mathrm{Extract}(K)$ where $|I| = |J| = |L| = 128$
$\boldsymbol{K} \leftarrow (\mathbf{0}, I, J, L, I, J, L, I, J, L, I)$
**if** $j = -1$ **then** $\Delta \leftarrow iJ$;  **return** $\mathrm{AES10}_{\boldsymbol{K}}(X \oplus \Delta)$ **fi**
$\boldsymbol{k} \leftarrow \boldsymbol{k}_1 \leftarrow (\mathbf{0}, J, I, L, \mathbf{0})$;  $\boldsymbol{k}_2 \leftarrow (\mathbf{0}, L, I, J, L)$
**if** $j = 0$ **then** $\Delta \leftarrow iI$;  **return** $\mathrm{AES4}_{\boldsymbol{k}}(X \oplus \Delta)$ **fi**
**if** $1 \le j \le 2$ **then** $\Delta \leftarrow (2^{3+\lfloor (i-1)/8 \rfloor} + ((i-1) \bmod 8))I$;  **return** $\mathrm{AES4}_{\boldsymbol{k}_j}(X \oplus \Delta)$ **fi**
**if** $j \ge 3$ **and** $i = 0$ **then** $\Delta \leftarrow 2^{j-3} \cdot L$;  **return** $\mathrm{AES4}_{\boldsymbol{k}}(X \oplus \Delta) \oplus \Delta$ **fi**
**if** $j \ge 3$ **and** $i \ge 1$ **then** $\Delta \leftarrow 2^{j-3} \cdot L \oplus (2^{3+\lfloor (i-1)/8 \rfloor} + (i-1 \bmod 8))J$;  **return** $\mathrm{AES4}_{\boldsymbol{k}}(X \oplus \Delta) \oplus \Delta$ **fi**
```

```
**algorithm** Extract($K$)                                          // Map key to subkeys
**if** $|K| = 384$ **then return** $K$
**else return** BLAKE2b($K$)
```

Figure 4: **AEZ's hash, PRF, TBC, key-derivation algorithms.** The last carries out key processing that an implementation would normally do at session-setup. Procedure Extract calls BLAKE2b, which returns a 48-byte digest. An alternative "scaled-up" algorithm, AEZ10, would redefine E by setting $\mathrm{E}_K^{j,i}(X) = \mathrm{AES}_K(X \oplus iI \oplus jJ)$ where $I = \mathrm{AES}_K(\mathbf{0})$ and $J = \mathrm{AES}_K(\mathbf{1})$, now restricting keys to $\{0,1\}^{128}$.

and top-right of Figure 5. Regard each rectangle with a pair of numbers as a TBC, the label being the tweak and the key K left implicit. Each pair of i-blocks $M_i M_i'$ is subjected to a two-round Feistel network. This both begins the scrambling of $M_i M_i'$ and yields a value $X = X_1 \oplus \cdots \oplus X_m$ that is a computational almost-xor-universal hash of $M_1 M_1' \cdots M_m M_m'$. The final pair of blocks $M_{\mathsf{x}} M_{\mathsf{y}}$ are now processed, but where X initially offsets one of them. This both begins the scrambling of $M_{\mathsf{x}} M_{\mathsf{y}}$ and yields the value S that is a computational almost-universal hash of all of M. The TBC calls of the middle row of the i-blocks now inject an (i, S)-dependent value. Two more Feistel rounds to each i-block gives $C_1 C_1' \cdots C_m C_m'$. To compute $C_{\mathsf{x}} C_{\mathsf{y}}$ we likewise employ two more Feistel rounds, the C_{x} value offset by a value $Y = Y_1 \oplus \cdots \oplus Y_m$ analogous to X.

The top-middle panel shows how to deal with messages having an even number of blocks, the last of these a fragment. Now the message is partitioned into $M_1 M_1' \cdots M_m M_m' \, M_{\mathsf{u}} M_{\mathsf{v}} \, M_{\mathsf{x}} M_{\mathsf{y}}$ with all blocks full except M_{v}, which will have 1–15 bytes. The Figure 5 trapezoids used to process M_{v} denote 10^* padding (top and bottom) and truncation (middle). The value $X = X_1 \oplus \cdots \oplus X_m \oplus X_{\mathsf{v}} \oplus X_{\mathsf{u}}$ now includes contributions from X_{u} and X_{v}, and similarly for $Y = Y_1 \oplus \cdots \oplus Y_m \oplus Y_{\mathsf{u}} \oplus Y_{\mathsf{v}}$.

Messages with an odd number of blocks are handled similarly, with the v-column omitted and the padding and truncation, if needed, moved to the u-column. We say "if needed" because no padding or truncation is used if the u block is full (ie, an odd number of blocks, all of them full).

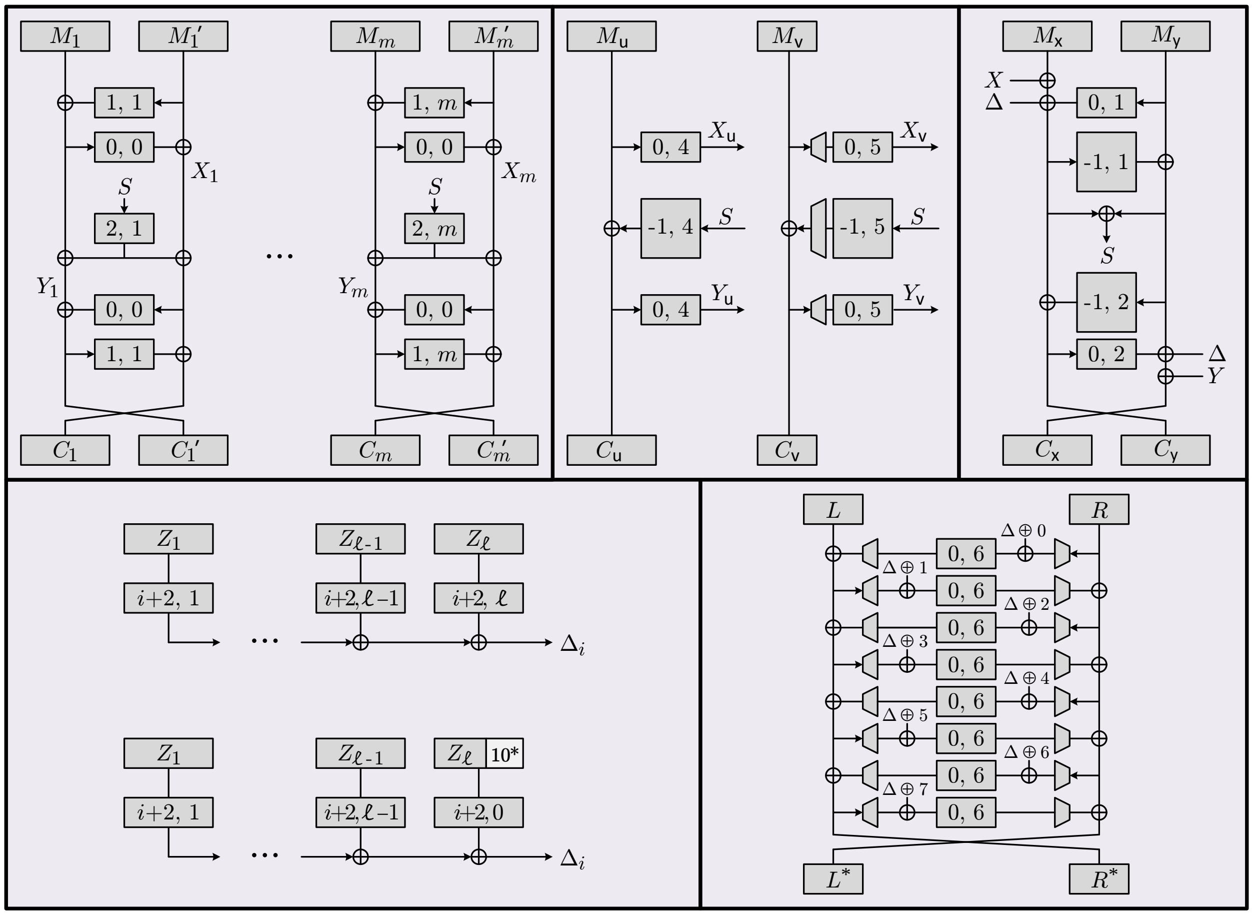

Figure 5: **Illustration of AEZ enciphering.** Rectangles with pairs of numbers are tweakable blockciphers, the pair being that tweak (the key, always K, is not shown). **Top row:** enciphering a message M of (32 or more bytes) with AEZ-core. The i-block (top left) is used for the bulk of the message, but the xy-block (top right) comprises the last 32 bytes, while the uv-block (top middle) comprises the prior 0–31 bytes. (The picture shows a uv-block of 17–31 bytes.) The string X is computed via $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus X_u \oplus X_v$; if X_u or X_v is undefined then this term is omitted in computing X. The string Y is computed analogously. **Bottom left:** AEZ-hash computes $\Delta = \bigoplus \Delta_i$ from a vector-valued tweak encoding ABYTES, N, and $\boldsymbol{A}$. Its i-th component $Z_1 \cdots Z_\ell$ is hashed as shown. **Bottom right:** AEZ-tiny, when operating on a string $M = L \parallel R$ of 16–31 bytes. More rounds are used if M has 1–15 bytes.

Let us call this construction just given AEZ-core[E]. It is the generalization of AEZ-core that employs an arbitrary tweakable blockcipher E. It should not be surprising that the construction is a strong-PRP under the assumption that the TBC used is secure as a tweakable-PRP. We prove this in the academic paper corresponding to this submission [21].

At this point we could instantiate E using a standard TBC based on AES: the XE method [27, 39] would do, yielding the scheme AEZ10 specified in the caption of Figure 4. We would then have a provably-secure enciphering scheme (for strings of 32 or more bytes) costing about five AES calls per pair of blocks, so 2.5 AES calls per block. The cost would be similar to EME [19, 20]: 0.5 more AES calls per block, but avoiding the repeated doubling and the use of AES-inverse.

But suppose we shatter our abstraction boundary and look at all that is going on to encipher M in AEZ10. Then the design starts to seem like major overkill: each block M_i is subjected to 30 rounds of AES (ten shared with a neighboring block), plus additional AES rounds to produce the unpredictable, M-dependent value S that gets injected into the process while 20 rounds yet remain.

In light of such overkill, AEZ-core selectively prunes some of the AES calls that AEZ10 would perform, using AES4 in their place. In particular, we prune invocations where we are trying to achieve computational xor-universal hashing. We leave enough AES rounds so that each block M_i is effectively processed with 12 AES rounds, eight of these subsequent to injection of the highly-unpredictable S and four of them shared with a neighboring block. The key steps in calculating S are not pruned, nor the TBC used to mask u- and v-blocks.

Tweak processing. So far we have not mentioned the processing of the tweak $\boldsymbol{T}$ built from N and $\boldsymbol{A}$. This is shown in the bottom-left of Figure 5. First we compute a hash AEZ-hash on $\boldsymbol{T}$ to create a value Δ; and then Δ is injected into AEZ-tiny and AEZ-core processing as shown.

Deciphering. We define Decipher$(K, \boldsymbol{T}, Y)$ as the unique X such that Encipher$(K, \boldsymbol{T}, X) = Y$. Logically, this is all we need say for the specification to be well-defined, so we omit writing out the implementing pseudocode in the body of this specification document. Still, we write it out in Appendix B, and explain here the needed changes in pseudocode. The reason the change is small is that enciphering and deciphering are highly symmetric for both AEZ-tiny and AEZ-core.

AEZ-tiny deciphering is identical to AEZ-tiny enciphering except that we must count backwards instead of forwards, and we must do the only-even-cycles correction (line 215) at the beginning instead of the end. Specifically, a routine Decipher-AEZ-tiny$(K, \boldsymbol{T}, M)$ (the M now representing ciphertext) is identical to Encipher-AEZ-tiny$(K, \boldsymbol{T}, M)$ except that line 214 is changed to count from $k-1$ down to 0, while for line 215 has each C replaced by M before moving the line up to just after line 212.

AEZ-core deciphering is identical to AEZ-core enciphering except that we must take the xy-tweaks in reverse order. Specifically, a routine Decipher-AEZ-core$(K, \boldsymbol{T}, M)$ (the M now representing ciphertext) is identical to Encipher-AEZ-core$(K, \boldsymbol{T}, M)$ except we swap tweaks $(0,1)$ and $(0,2)$, and we swap tweaks $(-1,1)$ and $(-1,2)$. These four tweaks appear at lines 227 and 232.

PRF. Using the Carter-Wegman approach, we also build a PRF AEZ-prf: counter-mode is employed to extend the output length if ABYTES > 16. The PRF is only used for the special case of enciphering the empty message. This is done for efficiency reasons—to make AEZ-prf$_K(X)$ = Encrypt$(K, \varepsilon, X, \tau, \varepsilon)$ an attractive PRF.

Key processing. For the users' convenience, AEZ allows keys of any length. Using procedure Extract, the provided key is processed into 48 bytes, if it is not already of that length, using a cryptographic hash function.

Tweakable blockcipher. The TBC $\mathrm{E}_K^{j,i}(X)$ takes a tweak $(j,i) \in (\{-1,0\} \times [0..7] \cup (\mathbb{N} \backslash \{0\}) \times \mathbb{N})$. The first component selects between AES10 (when $j = -1$) and AES4 (when $j \geq 0$). Either way, the construction is based on XE [27, 39]. A small amount of precomputation (to compute J, $2J$, $4J$, I, $2I$, and $4I$) will suffice.

1.6 Usage cap

We impose a limit that AEZ be used for at most 2^{48} bytes of data (about 280 TB); by that time, the user should rekey. For the purpose of this requirement, we say that, when encrypting $(N, \boldsymbol{A}, M)$ with a given key K, AEZ is acting on $\lceil |N|/8 \rceil + \lceil |\boldsymbol{A}|/8 \rceil + \lceil |M|/8 \rceil$ bytes. The above requirement stems from the existence of birthday attacks on AEZ, as well as the use of AES4 to create a universal hash function.

2 Security Goals

Nonce-reuse security. AEZ achieves *nonce-reuse misuse-resistance* (MRAE), as previously defined by Rogaway and Shrimpton [41]. In an MRAE scheme, repeating a nonce will violate privacy only insofar as repetitions of $(N, \boldsymbol{A}, M)$ triples will be identified as such. It will not compromise authenticity at all. SIV [41] is the best-known MRAE scheme.

Some researchers call AE schemes nonce-reuse misuse-resistant more broadly, encompassing schemes that achieve much weaker notions, like those that leak the longest common block-aligned prefix (for some fixed and typically small blocksize). Such notions were invented to approximate best-possible security for online schemes, which they do rather inexactly. MRAE schemes can't be online.

Exploitation of embedded novelty. MRAE security implies automatic exploitation of randomness or sequence numbers present in messages: in any context where messages are known to be distinct (eg, a sequence number is embedded somewhere within) or are extremely unlikely to collide (eg, a freshly-generated session key is embedded somewhere within), use of a nonce unnecessary. In such settings, omission of a nonce does *not* represent misuse; it is a sound way to encrypt.

Exploitation of domain-specific redundancy. In many contexts, plaintexts have a certain expected structure. This might arise because the message was produced by or for a particular protocol. We intend that if the user checks for the anticipated structure and regards messages as inauthentic if they don't comply, then this check augments authenticity and correspondingly lessens the need for the nominal redundancy that is inserted by AEZ before enciphering (that is, the extra ABYTES zero bytes). The concept of automatically exploiting redundancy present in plaintexts to achieve authenticity is well known in cryptographic folklore, where it has often been wrongly assumed, and demonstrably achieved for AE based on a strong-PRP [6].

Releasing unverified plaintext. When decrypting, an *unverified plaintext* is a string that will be released if the ciphertext is deemed authentic, but is supposed to be quashed otherwise. While not definitionally mandated, AE schemes routinely compute such a thing. One form of encryption-scheme misuse is to release some or all of the unverified plaintext despite the ciphertext's invalidity. This might happen because of an incremental decryption API or a more traditional side-channel.

Contemporaneous work by Andreeva *et. al* gives definitions to formalize an AE scheme's security against release of unverified plaintexts [1]. Our own definitional approach is different; we formalize *robust* AE, which incorporates the unverified-plaintext concern among its aspects. In claiming robust-AE security for AEZ the unverified plaintext is the value X computed at line 114. Achieving robust AE implies that no harm would come of returning $(X, \perp)$ instead of $\perp$ at line 116.

Per-message nonce-length and parameter authentication. No security problems result from employing nonces of varying lengths during a session, nor from changing the authenticator length ABYTES during a session. Of course accessing such capabilities would require an appropriate API.

Good security for low ciphertext expansion. Traditionally, AE security definitions "give up" when the adversary forges. This means that, at least definitionally, it's OK for a scheme to fail catastrophically when it first fails. A consequence is that authentication tags need to be so long that forgeries almost never occur. Yet there are applications where an occasional forgery is fine. For example, in some settings it ought to be fine to use a one-byte authenticator: while the adversary will have a 2^{-8} chance of forging a given message, we could still expect that, say, a reasonable adversary won't have much more than a 2^{-80} chance to forge ten consecutive messages.

AEZ permits short authentication tags, getting security as strong as possible given the selected authenticator length. This implies that we must use a new definition for AE, one that does *not* give up when a forgery occurs. It is described next.

Robust AE. Our new security definition for AE formalizes that one is doing *as good a job as possible for a given value τ of ciphertext expansion* ($\tau = 8 \cdot \text{ABYTES}$). The statement is required to hold even in the face of decryption leaking some specified information. An academic paper corresponding to this submission [21] defines and investigates this notion of *robust AE* (RAE). Here we sketch the idea.

We restrict attention to AE schemes $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ that operate on strings of any length and that are *τ-expanding*, $|\mathcal{E}_K^{N,\boldsymbol{A}}(M)| = |M| + \tau$, for a user-selectable $\tau \in [0..\tau_{\max}]$. We first consider an adversary that has access to one of two pairs of oracles. In the *real* setting the *encryption oracle* encrypts according to $\mathcal{E}$ and the *decryption oracle* decrypts according to $\mathcal{D}$. In the *ideal* setting the encryption oracle, asked $(N, \boldsymbol{A}, \tau, M)$, returns $\pi_{N,\boldsymbol{A},\tau}(M)$ where, for each $N, \boldsymbol{A}, \tau$, the function $\pi_{N,\boldsymbol{A},\tau}$ is a uniformly selected random injection from μ-bit strings to $(\mu + \tau)$-bit ones. All of these functions are chosen independently. The decryption oracle, given $(N, \boldsymbol{A}, \tau, C)$, checks if there's an M such that $\pi_{N,\boldsymbol{A},\tau}(M) = C$. If so, it returns M. Otherwise it returns the distinguished value $\perp$.

The above notion coincides with that of a *pseudorandom injection* (PRI) that has been updated to regard τ as in input. To arrive at the more general notion of an RAE scheme, we modify how decryption works in the ideal setting. This is unchanged when $(N, \boldsymbol{A}, \tau, C)$ is valid (that is, when there is an M such that $\pi_{N,\boldsymbol{A},\tau}(M) = C$), but when it's not, a simulator S gets to return what it wants. The return value may be based only on $N, \boldsymbol{A}, C, \tau$ and any saved state of S. The real decryption algorithm $\mathcal{D}$ can now be augmented to capture any desired leakage when the ciphertext is invalid: have algorithm $\mathcal{D}$ return what it wants, as long as it is recognizably invalid (eg, we can require that the length of the unverified plaintext not be $|C| - \tau$ bits). The notion is stronger than before insofar as not only must the scheme approximate a PRI with respect to valid ciphertexts, but, when they're invalid, the simulator must still be able to approximate that which $\mathcal{D}$ returns.

While the simulator S and invalid-message-returning $\mathcal{D}$ strengthen the RAE notion relative to the PRI notion, the key aspect, we think, is simply our insistence that encryption looks like a PRI even in the case that the ciphertext expansion is zero or small. In fact, when the ciphertext expansion is large, the PRI notion and the MRAE notion effectively coincide [41]. On the other hand, when ciphertext expansion is zero, the RAE (and PRI) notion coincides with that of a strong-PRP. RAE security can be thought of as a way to bridge strong-PRP security and MRAE security, coinciding

Security goal	**Query complexity**	**Time complexity**	**Approx formula**
Confidentiality of plaintext	55	128	$s^2/2^{110} + t/2^{128}$
Authenticity of plaintext	55	128	$s^2/2^{110} + t/2^{128}$
Authenticity of AD	55	128	$s^2/2^{110} + t/2^{128}$
Authenticity of the nonce	55	128	$s^2/2^{110} + t/2^{128}$
Robust AE	55	128	$s^2/2^{110} + t/2^{128}$

Figure 6: **Security goals for AEZ with default parameters (`aez`).** Query complexity is log base-2 of blocks queried: one needs about 2^{55} blocks before having a good chance to violate the goal. Time complexity is log base-2 of cycles: one needs about 2^{128} time to break the goal if one has only small amount of plaintext/ciphertext. The formula bounds adversarial advantage as a function of queried blocks (s) and time (t) by a known, modest-size adversary. The final row, RAE security, not only implies the other rows but also nonce-reuse misuse-resistance: AEZ provides maximum-possible robustness against nonce reuse.

with the former when τ is zero and the latter when τ is large.

Provable security. AEZ has been developed using the tools of provable security. The paradigm used is what we call *prove-then-prune*. First, a scheme is designed and proven secure when its underlying cryptographic tool—a tweakable blockcipher (TBC), in the case of AEZ—meets some well-established security definition. At that point one could instantiate the primitive with a conventional tool—eg, using AES and the XE construction [27, 39], as we described for AEZ10. One would then have a scheme with a customary provable-security claim. Instead, to make our scheme faster, we choose to selectively instantiate *some* of the TBC calls with a construction based on AES4, a four-round version of AES. Insofar as AES4 is *not* secure as a PRP (and, additionally, our method of tweaking it is not always XE), this step is effectively heuristic.

We call the instantiation of a scheme using a mixture of full and downgraded primitives the *scaled-down* design. In contrast, using a conventional construction for the primitive would yield the usual, *scaled-up* design. AEZ is a scaled-down realization of $\widetilde{\text{AEZ}}$. It is a *thesis* underlying our design methodology that the approach is useful both to discover good schemes and to have some measure of assurance for them.

Quantitative security statements. For the scaled-up version of AEZ with default parameters, we expect that an adversary cannot be exhibited that violates RAE security with advantage exceeding $4s^2/2^{128} + t/2^{128}$ where s is the total number of 16-byte blocks of messages encrypted or authenticated (plus 3 blocks per message, by convention) and t is the time (including the description size) in which the adversary runs. The second addend is a stand-in for an advantage term associated to breaking the PRP security for the underlying blockcipher. Constants 3 and 4 are the result of ongoing analysis. The number of encryption and decryption queries does not appear in the formula above because we have folded them into s.

For `aez` itself, the formula should be replaced by $4s^2/2^{113} + t/2^{128}$ because of the higher maximal expected differential probability of AES4 [24] compared to an ideal hash or cipher.

Many authors prefer to think of security in terms of number-of-bits. We would summarize the $4s^2/2^{113} < s^2/2^{110}$ term of the last formula by saying that `aez` is expected to have 55 bits of security. We warn that when an author makes a claim like "GCM has 128 bits of security" the focus is *time* complexity, imagining a fixed and small amount of ciphertext. When saying that we have at least 55 bits of security we are speaking exclusively of query complexity: that an adversary must gather roughly 2^{55} blocks (2^{59} bytes) worth of ciphertext before it has a good chance to break RAE security (assuming an explicitly given attack of reasonable description size and time complexity). Recall our usage cap, that AEZ should be used for at most 2^{48} bytes. One might summarize targeted security goals for `aez` as shown in Figure 6.

Non-goals. We have not tried to achieve security beyond the birthday bound; like traditional modes of operation based on a 128-bit blockcipher, there *are* easy distinguishing and forging attacks by the time the adversary queries AEZ with about 2^{64} blocks of message, AD, or nonce. There are even key-recovery birthday attacks: Chaigneau and Gilbert [11], following Fuhr, Leurent, and Suder [18], provide one that uses about $2^{66.5}$ chosen plaintexts. Similarly, we are not targeting time-complexity security in excess of what is inherent in employing a 128-bit key. (That said, we avoid the obvious 2^{128}-time brute-force attack for keys in excess of 128 bits by processing arbitrary-length keys to 48-byte subkey material in our key extraction processing.) We did not target related-key-attack security, although our use of a cryptographic hash function for keys not of 48 bytes suggests that we will achieve that end for strings of all other lengths.

Warnings. Robust AE should not be understood as blanket permission to omit a nonce or allocate too few bits for ciphertext expansion. Let us elaborate.

In the context of AE, misuse resistance (MR) has, of late, been brandished far too liberally, with some authors going so far as to call their *online* AE schemes *misuse resistant*, or even *nonce free*. We disapprove. Online AE schemes can never be misuse-resistant in the sense originally defined [41], and, what is worse, it is unclear that they imply *any* useful guarantee when nonces repeat. The definitions here [17] are deceptive, sounding stronger than they are [22].

We worry that the expansion of the term "misuse resistance" to online schemes may wrongly signal that there are online AE schemes where nonces are effectively optional. We wish to emphasize that, even with RAE, nonces *still* should not be construed as optional in the absence of supporting analysis. Specifically, a nonce must be used unless one has certitude that, even in the presence of the adversary, all encrypted $(\boldsymbol{A}_i, M_i)$ pairs will be distinct; or else one has some other domain-specific reason to believe that, for the given context, leaking plaintext-equality is not problematic.

In a similar vein, AEZ allows little or no ciphertext expansion. But the adversary's per-message forging probability increases with decreasing redundancy. Applications should not reduce ABYTES to zero or some other small value without ensuring that, combining the ABYTES zero bytes with any decryption-verified redundancy, there remain enough total bits r of redundancy that forging each message with probability 2^{-r} is alright.

The robust AE notion does not guarantee security against *arbitrary* leakage. For AEZ, one can release the entire string $X \leftarrow \text{Decipher}(K, \boldsymbol{T}, C)$ at line 114, but we have not indicated that anything beyond that may be released. In particular, Barwell, Page, and Stam [4, Appendix E] point out that if one is following the early-abort strategy to reject invalid ciphertext then it is crucial that one not release the partially decrypted ciphertext: with early-abort processing, all that may be released is, as usual, the indication of invalidity.

3 Security Analysis

An academic paper corresponding to this submission [21] gives the relevant security proofs for AEZ. Here we summarize some of our results. All are in the provable-security tradition (as opposed to our making cryptanalytic claims).

Ciphertexts of at least one block. Let $\widetilde{\mathrm{AEZ}}[\mathrm{E}]$ be the generalization of AEZ where each E is a tweakable blockcipher (TBC) of the correct signature [27]. We can prove that $\widetilde{\mathrm{AEZ}}[\mathrm{E}]$ achieves RAE security as long as E is secure as a tweakable PRP. The claim assumes that $|M| + \tau \geq 128$ for each encryption query employing plaintext M, and $|C| \geq 128$ for each decryption query of a ciphertext C. These conditions hold automatically for the default choice of ABYTES = 16. With those provisos, RAE security can be proven along the following lines.

- AEZ-core provides a length-preserving, variable-input-length, strong PRP on $\mathrm{Byte}^{\geq 32}$ (strings of 32 or more bytes) with birthday-bound distinguishing advantage. This statement requires only chosen-plaintext-attack PRP security for the underlying TBC.
- The tweak provided to Encipher-AEZ-core is incorporated by what can be regarded as the XEX construction [27, 39]. The underlying hash function, AEZ-hash, is almost-xor universal (AXU) when E is a PRP.
- The round functions of AEZ-tiny are derived from a tweakable blockcipher (TBC) with tweak space $\mathcal{T} = \{(i, 0) \mid i = 1, \ldots, 24\}$. We employ the XE construction [27, 39] to extend the tweak space to $\mathcal{T} \times \mathcal{N} \times \mathcal{A}$. One can then view that, for each $(N, \boldsymbol{A})$, we use independent round functions. Since a 6-round Feistel network on $\{0,1\}^{2n}$ already yields a strong PRP with birthday-bound distinguishing advantage [26, 34, 35], AEZ-tiny gives a length-preserving, strong tweakable-PRP on $\mathrm{Byte}^{\geq 16} \cap \mathrm{Byte}^{\leq 31}$, with birthday-bound distinguishing advantage.
- Once one has shown that the Encipher procedure of $\widetilde{\mathrm{AEZ}}$ provides a length-preserving strong tweakable-PRP then $\widetilde{\mathrm{AEZ}}$ itself is a robust-AE scheme. This follows from a generic result that asserts that encode-then-encipher conversion gives RAE security.

The choice of our TBC is heuristically justified as follows.

- The processing of the tweaks to compute the XE offsets only requires a universal hash, and four-round AES with independent, uniformly random subkeys is already known to be a good AXU hash [24]. Similarly, the AXU security for AEZ-hash can be justified by viewing AEZ-hash as an approximation of a variant in which the subkeys are chosen uniformly and independently from $\{0,1\}^{128 \cdot 4} \times \{0^{128}\}$. That variant of AEZ-hash is again AXU due to the fact that four-round AES with independent, uniformly random subkeys is an AXU hash [24].
- For AEZ-core, when processing each pair of blocks M_{x} and M_{y}, the first and last rounds only need to be AXU, due to the classic result of Naor and Reingold [31]. Then, for the four-round Feistel networks that process M_i and M_i' with $i \geq 1$, we heuristically use AES4 for the round function, since, even then, each ciphertext block C_i is processed with 12 AES rounds (four of which are shared with a single neighboring block), eight of which are subsequent to full mixing, and all of which are subsequent to the position-dependent masking.
- For AEZ-tiny we are effectively using a minimum of $32 = 8 \cdot 4$ rounds of AES. While AES4

is not itself a good PRF, it would seem to be a stronger round function than those used by most conventional Feistel-based designs.

Let ϵ be the maximum expected differential probability of (independently-keyed) AES4; this is known to be at most $(52/2^{34})^4 \approx 2^{-113.088}$ [24]. While $\widetilde{\text{AEZ}}$ achieves RAE security with birthday-bound security in the blocksize, AEZ only achieves RAE security with advantage about $\sigma^2 \cdot \epsilon$, where σ is the number of blocks that the adversary queries. There are corresponding attacks. As a simple example, let ABYTES = 16 and have an adversary repeatedly ask to encrypt a fixed message M with a fixed nonce N but using AD values that consist of two random blocks. A collision in ciphertexts will be found in about $1/\sqrt{\epsilon}$ expected queries. Say it arose from AD values of $\boldsymbol{A} = (A_0 A_1)$ and $\boldsymbol{A}' = (A'_0 A'_1)$. Then test if one again gets a collision with M and N but with AD values of either $\boldsymbol{A} \parallel \mathbf{0}$ or $\boldsymbol{A}' \parallel \mathbf{0}$. If so, one almost certainly has a "real" encryption oracle.

Security of AEZ-prf. If AEZ-hash is an AXU hash and AES10 is a good PRF then AEZ-prf is a PRF, as AEZ-prf is constructed from the Carter-Wegman paradigm [10], with output length expanded via counter mode of AES10 for ABYTES > 16. Alternatively, for the default ABYTES = 16, one can view AEZ-prf as an approximation of an AES-based PMAC [9] in which all but the final blockcipher call have had the number of AES rounds reduced from 10 to 4, a heuristic employed in ALRED, MARVIN, and PELICAN [12, 13, 42, 44]. This gives another heuristic justification for the scaling down from AES10 to AES4 in AEZ-hash.

Ciphertexts of less than one block. The claim that Encipher-AEZ-tiny gives a tweakable, strong PRP over $\text{BYTE}^{\le 15}$ is only heuristically justified. Consider a collection of independent, ideal, k-round Feistel networks on $\{0,1\}^{2n}$; the round functions are all uniformly random and independent. The best attack known that distinguishes them from a family of independent, truly random even permutations, requires at least $2^{(k-3)n}$ plaintext/ciphertext pairs [33]. (Patarin claims that his attack needs only $2^{(k-4)n}$ plaintext/ciphertext pairs, but Bellare, Hoang, and Tessaro [5] point out that this count is incorrect and the cost of his attack is at least $2^{(k-3)n}$ pairs.) From our choice of the number of rounds, this attack needs at least 2^{84} plaintext/ciphertext pairs, and thus doesn't violate our security goals.

There are of course many provable-security results on balanced Feistel as well, but proven bounds for a fixed-round Feistel network operating on an μ-bit string vanish at about $2^{\mu/2}$ queries, and we are looking at settings with μ as small as 8.

Key scheduling and AES4/AES10 details. For the analysis above we sometimes pretended that the subkeys for AES4 (excluding the XE offsets) are independent of other keys. In the implementation, to reduce context size, all subkeys are made from three blocks: I, J, and L. Associated to this choice, we elect to determine these from the underlying key K in a more conservative manner than with the AES key-scheduling algorithm, which we do not employ at all.

In defining AES4 and AES10 subkeys, the initial and final subkeys are sometimes taken to be zero. In most cases arguments can be given that the simplification is without adverse consequence. Cascading a post-whitened permutation with a pre-whitened one seems redundant. If prewhitening is used from the XE construction then little benefit is gained from an initial round key. If one is aiming to construct an AXU hash function, postwhitening is pointless. Pleasantly, using zero as a final AES4 round key frees up the xor included in the `aesenc` instruction to do the other

operation	$m \geq 2$ even $d = 128$	$m \geq 2$ even $d < 128$	$m \geq 3$ odd $d = 128$	$m \geq 3$ odd $d < 128$	$m = 1$ $d = 8$	$m = 1$ $d = 16$	$m = 1$ $d \geq 24$	$m = 2$ $d < 128$
encipher or decipher [a]	m+0.8 (3.6)	m+2.4 (3.6)	m+1.6 (3.6)	m+1.6 (3.6)	10 (10)	6.8 (6.8)	4.4 (4.4)	3.2 (3.2)
encrypt or decrypt [b]	m+3 (3.6)	m+3 (3.6)	m+2.2 (3.6)	m+3.8 (3.6)	3.6 (3.6)	3.6 (3.6)	3.6 (3.6)	5 (4)
reject invalid ciphertext [b]	0.4m+2.4 (2.8)	0.4m+2 (2.8)	0.4m+2.4 (2.8)	0.4m+2 (2.8)	0 (0)	0 (0)	0 (0)	3.6 (3.6)

Figure 7: **Efficiency of AEZ.** Worst-case computational work (and, parenthesized, latency) measured in AES-equivalents, defined as ten AES rounds. The nonempty string X being operated on has $m = \lceil |X|/128 \rceil$ blocks, the possibly-fragmentary last one having $1 \leq d \leq 128$ bits. Assumptions: (a) Key already setup, nonce and AD already processed. (b) Key already setup, AD already processed, nonce has 16 or fewer bytes, ABYTES = 16. Other tasks: **Key setup**: 1.2m (0.4), assuming all needed constants have been precomputed. **Process string-valued AD**: 0.4m (0.4) (key already setup, nonce of 16 or fewer bytes).

computational work needed for Feistel.

AES4 and AES10 do not omit the final-round `MixColumns`, as AES itself does. In the context of repeated AES4 applications, omission of the final `MixColumns` would likely decrease security. See Dunkelman and Keller for some work in this direction [14]. And the AES designers' motivation for removing the `MixColumns` step from the last round of AES is for us moot: the inverse AES cipher is never used.

We emphasize that the E construction is *not* secure as a tweakable-PRP; four AES rounds of AES is not sufficient for that purpose. This is where the scaling-down has occurred. One only gets a tweakable PRP by moving to the construction described for AEZ10.

For key scheduling, the cryptographic hash function BLAKE2b is used by Extract to create the three 128-bit subkeys I, J, L from the arbitrary-length key K provided. We dispense with calling BLAKE2b if the key K is already $3 \cdot 128$ bits. Applying a variable-output-length hash function is a conceptually simple way to accomplish the needed key scheduling. The specific choice of BLAKE2b is motivated by our intent to ultimately surround AEZ by a wrapper that supports slow and memory-intensive key processing using Argon2 [8], the winner of the password-hash competition. That algorithm is itself based on BLAKE2b, making it the most natural choice for AEZ. That said, we regard Extract as an essentially orthogonal aspect to the rest of the AEZ design.

4 Features

See Figure 7 for a table summarizing computational costs and Figure 8 for a table summarizing algorithmic features. Below we enumerate additional features and restate some key ones.

1) Strings of any byte length ℓ can be encrypted into strings of ℓ + ABYTES bytes for any (user-selectable) value ABYTES. One achieves the maximal privacy and authenticity consistent with ABYTES (assuming this value is not excessively large, whence its increase adds nothing). The value ABYTES is authenticated and may change as often as a user likes.
2) Computational cost is close to that of AES-CTR mode: roughly 1 AES-equivalent per block.

Objective	**Robust-AE**, a goal that implies **MRAE** (nonce-reuse misuse resistance) [41].
Type	**Blockcipher-based** scheme, based on **AES4** and **AES10**.
Intended for	**sw/hw/lw**. Intended to do well where AES does, in SW or HW, and on low-power devices where ciphertext length should be minimized.
Key length	**Arbitrary**. Keys of 48 bytes are most efficient. Fewer than 16 bytes is discouraged.
Nonce length	**Arbitrary**. May vary during a session.
Auth length	**Arbitrary**. Expansion beyond 16 bytes does not enhance security. Expansion by 0 bytes gives a strong, tweakable, VIL blockcipher.
Nonce reuse	**Yes**. Secure against nonce-reuse in the strongest sense of the phrase [41].
Online	**No**. MRAE and RAE schemes can't be online (neither encryption nor decryption).
Unverified plaintext	**Yes**. It is fine to release unverified plaintext (a recovered but inauthentic plaintext). This is one aspect of our notion of a robust AE (when $\mathcal{D}$ is appropriately defined).
Parallelizable	**Yes**. Two passes must be made to encrypt or decrypt, but both are parallelizable. Processing of the AD is also parallelizable.
Incremental	**No**. MRAE schemes can't be incremental. Use as a deterministic MAC is incremental with respect to block replacement or appending-on-the-right.
Inverse free	**Yes**. The inverse direction of AES4 or AES10 is never used.
Context size	**144 bytes** for $I, J, 2I, 2J, 4I, 4J, \Delta_1, \Delta_3$ or **128 bytes** for $I, J, 2I, 2J, 4I, 4J, L, \Delta_3$ (0.4 extra AES per AD) or **64 bytes** for I, J, L, Δ_3 (2 extra doublings per msg, 4 per AD) or **48 bytes** for I, J, L (can't preprocess the AD). We currently use 128 in our code.
Static AD	**Yes.** Static AD values can be preprocessed and used thereafter at near-zero cost.
Fast reject	**Yes.** Invalid ciphertexts can be rejected far more quickly than valid ones decrypted.
Performance	About the cost of OCB or AES-CTR, approaching 1.0 AES-equivalents per block
Proofs	Either: **Yes**, there are proofs, but then a heuristic optimization is applied to a provably-secure scheme to get a nice speedup; or **No**, there are no proofs for AEZ itself, although the authors employ provable-security to motivate and justify some design choices.
Further features	▸ Can exploit arbitrary redundancy in messages for authenticity ▸ Can be used as an efficient, parallelizable MAC (encrypt the empty string). ▸ Can be used to encipher short strings and to encrypt strings with low expansion. ▸ ABYTES is authenticated may vary during a session. ▸ Extensions (not AEZ itself) will support secret nonces, plaintext-length obfuscation, and radix64url output encoding. ▸ No patents.

Figure 8: **Table of properties for AEZ**. The choice of properties to list as rows evolved from slides prepared by Bart Preneel during a Dagstuhl workshop [37].

And an implementation only needs to employ the forward direction of AES.

3) Nonces can have any length. Their lengths can vary during a session.
4) The AD can be an arbitrary list of arbitrary strings. This obviates the need for users to encode multiple strings into a string-valued AD.
5) It is fine to omit nonces (select $N = \varepsilon$) if one is sure that $(\boldsymbol{A}, M)$ pairs will not repeat. If they do repeat, damage is limited to divulging equality among $(\boldsymbol{A}_i, M_i)$, $(\boldsymbol{A}_j, M_j)$ pairs.
6) Can be used as an arbitrary-output-length pseudorandom generator (PRG) or pseudorandom function (PRF): define $f_K(X, \ell)$ as the result encrypting $M = \varepsilon$ with $\boldsymbol{A} = (X)$ and ABYTES $= \ell$. Here ℓ is number of bytes requested.

7) Keys can have any length. A user may, for example, use a passphrase or DH ephemeral key. (Note: some features one might want for mapping a passphrase to a 128-bit key, like salting and an intentionally slow mapping to slow password guessing, are not natively provided.)
8) AEZ functions well as a stand-alone MAC and as a stand-alone enciphering scheme. The first use needs only 0.4 AES operations per block.
9) Verification of plaintext redundancy enhances authenticity, as we have already explained.
10) Short authenticators provide the security one would hope for. Our security notion doesn't "give up" when the adversary forges. This is part of the robust-AE notion.
11) Release of unverified plaintext is fine. This is another part of the robust-AE notion.
12) The security properties achieved by AEZ enable support for secret message numbers as a simple add-on. This will be accomplished as an AEZ extension. Further AEZ extensions will handle plaintext-length obfuscation, password salting, password guess-throttling, and encoding ciphertexts into a target alphabet.
13) An encryption implementation can make one left-to-right pass over the plaintext, writing an intermediate string as long as the plaintext; and then a second left-to-right, constant-memory pass over the intermediate string, this time outputting the ciphertext online. Decryption can be similarly realized. If one does not want to write out an intermediate string, which must not be released to the adversary, the cost increases from 1 AES-equivalent per block to 1.4 AES-equivalents per block.
14) Some AE schemes need that the AD be available before the message is processed. AEZ only requires AD processing be completed by the end of pass-1.
15) It is possible to accelerate the rejection of invalid ciphertexts by having decryption compute the final ciphertext block prior to computing the remainder of the plaintext. The cost for early-rejection is about 0.4 AES-equivalents per block.
16) AEZ is fully parallelizable in the processing of plaintext, ciphertext, and AD.
17) Static AD can be preprocessed so that one doesn't have to subsequently pay a per-message $|\boldsymbol{A}|$-dependent cost. Of course realizing this benefit requires an API that decouples provisioning of the AD and provisioning of other inputs.)
18) Word alignment of the message and AD are not disrupted (for example, one never prepends a byte to the message or AD, and then processes it.
19) The context size has been kept quite small: retaining "everything" one might want gives a context size is 144 bytes, and an implementation can make due with as little as 48 bytes without incurring an excessive computational price.
20) No AEZ-related patents have been or will be requested.

Performance is close to that of AES-CTR on modern processors. On an Intel Skylake CPU, where AES-CTR peaks at 0.625 cpb, our current implementation of AEZ, written in C with SSE intrinsics, encrypts or decrypts 16 KB strings at 0.64 cpb and 1500 byte strings at 0.72 cpb. On decryption, invalid messages are more quickly rejected; for example, an invalid 1500 byte string is rejected at a cost of 0.31 cpb. Associated data processing speed, which also determines MAC speed, is 0.29 cpb for 1500 bytes. On Apple's A9 ARMv8 implementation, where AES-CTR peaks at around 1.2 cpb, our implementation encrypts or decrypts 16 KB strings at 1.3 cpb and 1500 byte strings at 1.5 cpb. It takes about 0.6 cpb to either reject an invalid 1500 byte messages or process 1500 bytes of associated data. All of these figures assume 12-byte nonces and 16-byte tags.

Advantages over GCM. AEZ has much stronger security properties than GCM. The later is not nonce-reuse secure, cannot safely generate short tags [16], and is not secure with respect to disclosure of unverified plaintext. GCM does not achieve the RAE security definition. AEZ avoids GF(2^{128}) multiplies (apart from the finite-field "doubling" that it uses).

A closer match to AEZ in terms of high-level aims is SIV, which is at least nonce-reuse secure [41]. But SIV has to output 128-bits more than its input; it is not RAE secure; and it is not parallelizable (although the last issue could easily be fixed).

5 Design Rationale

Enciphering-based AE. An old result had already shown that enciphering with a strong PRP provides a versatile route to AE [6]. We recently came to understand just how attractive this route might be. On the one hand, we kept hearing requests for stronger AE security properties, like nonce-reuse misuse-resistance, authenticity without minimal ciphertext expansions, and security if unverified plaintexts are disclosed. Enciphering-based AE could deliver such aims. On the other hand, enciphering schemes that worked on either long or short strings were steadily becoming better-known objects. While they didn't have the efficiency of OCB, say, neither were they computationally exorbitant. And there was the hope of doing better.

Developing the enciphering scheme. With AES support increasingly embedded into devices, we wanted to base our enciphering scheme on the AES round function. A wide body of work had made abundantly clear that the best techniques for AES-based enciphering were going to depend on the length of the plaintext. When the plaintext was short, we would want a simple, `aesenc`-based design. For long strings we would want a more conventional mode. To cover all strings we'd have to glue the two together.

For enciphering short strings, some version of FFX [7] was the obvious choice. It was already in a draft standard [15], and the long history of Feistel networks made the choice seem safe (even if security bounds for balanced Feistel networks become disappointing when the input gets too short). To ensure security against known distinguishing attacks [33], AEZ uses extra rounds for enciphering very short strings. Our conservative round counts in AEZ-tiny are supported by recent results: for very short strings, FFX is somewhat vulnerable to a message-recovery attack described by Bellare, Hoang, and Tessaro [5]. For AEZ, their attack would need at least 2^{84} ciphertexts.

For enciphering longer strings, there were a great many off-the-shelf alternatives we could turn to (see [40] for a list). The best-known was EME2 [19, 23]. But its treatment of final fragments and long messages seemed complex, and it needed two AES calls per block and lots of doubling. Most alternatives traded a blockcipher calls for a potentially expensive finite-field operation, a direction we didn't want to go. We decided that no off-the-shelf solution would do.

AEZ-core builds on EME [19, 20] and OTR [29], but uses tweakable blockciphers [27] to arrive at an analyzable design. It makes strong use of what we have called prove-then-prune approach. The scaled-down design, with a per-block cost of just 1.0 times that of AES and no use of inverse-AES, was cheaper than we initially imagined to be possible. While it has long been understood that stream ciphers could be faster than blockciphers, it was not anticipated, at least by us, that a wide-blocksize blockcipher could be about as cheap as a conventional blockcipher.

No hidden weaknesses. The designers have not hidden any weaknesses in this cipher. The authors do not know any technical means by which one *could* intentionally weaken the design of a scheme like AEZ. The authors excoriate intelligence-agency efforts to subvert security standards and mass-market implementations.

6 Intellectual Property

The submitters have not applied for any patents in connection with this submission and have no intention to do so. As far as the inventors know, AEZ may be used in an application or context without IP-related restrictions. If any of this information changes, the submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

7 Consent

The submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

8 Changes

- **AEZ v1** (2014.03.15): Initial definition. Submitted to the CAESAR competition. **AEZ v1.1** (2014.04.29): A minor revision to correct various v1 typos.
- **AEZ v2** (2014.08.17): A major revision. The enciphering algorithm MEM was replaced by EME4. While no problems were ever found with MEM, the move facilitates two major gains: (a) the cost is reduced from from 1.8 times that of AES to 1.0 times that of AES, while (b) all use of the AES-inverse operation is removed from AEZ. Also, EME4 is simpler, and the entire spec was correspondingly simplified.

- **AEZ v3** (2014.09.22): A modest revision. To simplify implementations the $(M_{\mathsf{x}}, M_{\mathsf{y}})$ pair of blocks is now taken from the end of the string being enciphered/deciphered instead of the beginning. Round keys were simplified. To minimize latency and facilitate fast rejection of invalid ciphertexts, both X and Δ are added to M_{x}. Support is added for vector valued-AD, which entailed enriching AEZ-hash and removing Format(), EXTNS, and the upper limit on ABYTES. Functions (FF0, EME4, AHash, AMac) were renamed to (AEZ-tiny, AEZ-core, AEZ-hash, AEZ-prf).
- **AEZ v4** (2015.08.29): A minor revision. The Extract() procedure was rewritten to be conceptually simpler and ensure that knowledge of some subkeys (I, J, or L) won't imply knowledge of others. The default key length was changed to 48 bytes. The XEX construction, instead of XE, is now used in AEZ-hash. It was hoped that this would help frustrate a key-recovery birthday attack described by Fuhr, Leurent, and Suder [18], but it was later shown by Chaigneau and Gilbert that such birthday attacks remain [11]. Offset conventions in E were modified for aesthetic reasons. **AEZ v4.1** (2015.10.14): Housecleaning: fixed two typos in the pseudocode (lines 403, 408) and some minor textual edits. Update of performance claims to reflect newer CPUs. No algorithmic changes. **AEZ v4.2** (2016.09.15): Some notational tweaks in response to comments. Added mention of some recent attacks. No algorithmic changes.

Acknowledgments

For comments, ideas, feedback, and corrections, our thanks to **Guy Barwell**, **Mihir Bellare**, **Dan Bernstein**, **Dustin Boswell**, **Colin Chaigneau**, **Orr Dunkelman**, **Kris Gaj**, **Henri Gilbert**, **Danilo Gligoroski**, **David Gregory**, **Ekawat Homsirikamol** ("Ice"), **Philipp Jovanovic**, **Gaëtan Leurent**, **Stefan Lucks**, **Nick Mathewson**, **Liden Mu**, **Chris Patton**, **Tom Ristenpart**, **Terence Spies**, **René Struik**, and **Yusi Zhang** ("James").

Elaborating, Boswell, Lucks, Spies, and Struik all played roles in motivating the formalization of RAE and creation of AEZ. Lucks spoke with Rogaway at Jan 2012 Dagstuhl event where he advocated developing a solution to the problem of leaking unverified plaintexts. In an Apr 2013 email to Bellare, Boswell wrote of wanting an easier-to-use encryption scheme. Bellare passed the note on. Struik gave an Aug 2013 DIAC presentation that emphasized the importance of minimizing length expansion in low-energy environments [45]. Meanwhile, work with Spies on FPE helped motivate linking up enciphering and AE in the first place.

Patton and Mu both did AEZ implementations, which helped validate our own. Gaj and Homsirikamol kindly shared their thoughts on hardware-implementation aspects of AEZ. Unfortunately, most of their concerns were fundamental to the AEZ design, or to RAE/MRAE in general, so we weren't able to do much. Mathewson pointed out a couple of typos in the v4 spec and did an AEZ implementation in Python.

Krovetz was supported by NSF grant CNS-1314592 and Rogaway was supported by NSF grants CNS-1228828 and CNS-1314885. Many thanks to the NSF for their continuing support. Part of this work was done when Hoang was working at UC San Diego, University of Maryland, Georgetown University, and UC Santa Barbara, under the support of NSF grants CNS-0904380, CCF-0915675, CNS-1116800, CNS-1228890, CNS-1423566, CNS-1553758 (CAREER), and by the Glen and Susanne Culler Chair.

No animals were harmed in conducting this research.

References

[1] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda. How to securely release unverified plaintext in authenticated encryption. Cryptology ePrint Archive, Report 2014/144. Feb 25, 2014.

[2] C. Arnould. Towards developing ASIC and FPGA architectures of high-throughput CAESAR candidates. ETH Zürich Master's Project. March 2015.

[3] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein. BLAKE2: simpler, smaller, fast as MD5. *ACNS 2013*, LNCS vol. 7954, pp. 119–135, 2013.

[4] G. Barwell, D. Page, and M. Stam. Rogue decryption failures: Reconciling AE robustness notions. *IMA International Conference on Cryptography and Coding 2015*, pp. 94–111, 2015. Also Cryptology ePrint report 2015/895.

[5] M. Bellare, V. T. Hoang, and S. Tessaro. Message-recovery attacks on Feistel-based Format Preserving Encryption, to appear in *CCS 2016*.

[6] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. *ASIACRYPT 2000*, LNCS 1976, Springer, pp. 317–330, 2000.

[7] M. Bellare, P. Rogaway, and T. Spies. The FFX mode of operation for format-preserving encryption. Draft 1.1. Submission to NIST, available from their website. Feb 20, 2010.

[8] A. Biryukov, D. Dinu, and D. Khovratovich. Argon2. Password Hashing Competition (PHC) submission. July 8, 2015. https://password-hashing.net/submissions/specs/Argon-v3.pdf

[9] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. *EUROCRYPT 2002*, LNCS 2332, Springer, pp. 384–397, 2002.

[10] L. Carter and M. Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2), pp. 143–154, 1979.

[11] C. Chaigneau and H. Gilbert. Is AEZ v4.1 sufficiently resilient against key-recovery attacks? To appear in *FSE 2017*.

[12] J. Daemen and V. Rijmen. The Pelican MAC function. Cryptology ePrint Archive: Report 2005/088. 2005.

[13] J. Daemen and V. Rijmen. A new MAC construction ALRED and a specific instance ALPHA-MAC. *Fast Software Encryption*. LNCS 3557, pp. 1–17, 2005.

[14] O. Dunkelman and N. Keller. The effects of the omission of last round's MixColumns on AES. *Information Processing Letters*, 110, pp. 304–308, 2010.

[15] M. Dworkin. Recommendation for block cipher modes of operation: methods for format-preserving encryption. NIST Special Publication 800-38G: Draft. Jul 2013.

[16] N. Ferguson. Authentication weaknesses in GCM. Manuscript. May 2005.

[17] E. Fleischmann, C. Forler, S. Lucks, and J. Wenzel. McOE: A foolproof on-line authenticated encryption scheme. IACR Cryptology ePrint Archive 2011, 644 (2011).

[18] T. Fuhr, G. Leurent, and V. Suder. Collision attacks against CAESAR candidates—Forgery and key-recovery against AEZ and Marble. *ASIACRYPT 2015*, LNCS 9453, Springer, pp. 510-532, 2015.

[19] S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. INDOCRYPT 2004. pp. 315–327, 2004.

[20] S. Halevi and P. Rogaway. A parallelizable enciphering mode. *CT-RSA 2004*, LNCS 2964, Springer, pp. 292–304, 2004.

[21] V. T. Hoang, T. Krovetz, and P. Rogaway. Robust authenticated-encryption: AEZ and the problem that it solves. *EUROCRYPT 2015*, part I, LNCS 9056, Springer, pp. 15–44, 2015.

[22] V. T. Hoang, R. Reyhanitabar, P. Rogaway, and D. Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. *CRYPTO 2015*, part I, LNCS. 9215, Springer, 493–517, 2015.

[23] IEEE P1619.2. Draft standard architecture for wide-block encryption for shared storage media. 2008. Available from https://siswg.net.

[24] L. Keliher and J. Sui. Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. *IET Information Security*, 1(2), pp. 53–57, 2007.

[25] T. Krovetz and P. Rogaway. The software performance of authenticated-encryption modes. *FSE 2011*, LNCS 6733, Springer, pp. 306–327, 2011.

[26] R. Lampe and J. Patarin. Composition theorems for CCA cryptographic security. Cryptology ePrint report 2012/131. May 2012.

[27] M. Liskov, R. Rivest, and D. Wagner. Tweakable block ciphers. *CRYPTO 2002*, LNCS 2442, Springer, pp. 31–46, 2002

[28] D. McGrew. An interface and algorithms for authenticated encryption. RFC 5116. Jan 2008

[29] K. Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. *EUROCRYPT 2014*, LNCS 8441, Springer, pp. 275–292, 2014.

[30] K. Minematsu and Y. Tsunoo. Provably secure MACs from differentially-uniform permutations and AES-based implementations. FSE 2006, LNCS 4047, Springer, pp. 226–241, 2006.

[31] M. Naor and O. Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited. *Journal of Cryptology*, 12(1), pp. 29-66, 1999.

[32] M. Naor and O. Reingold. The NR mode of operation. Undated manuscript realizing the mechanism of [31].

[33] J. Patarin. Generic attacks on Feistel schemes. *ASIACRYPT 2001*, LNCS 2248, Springer, pp. 222–238, 2001. Also see Cryptology ePrint report 2008/036.

[34] J. Patarin. Security of balanced and unbalanced Feistel schemes with linear non equalities. Cryptology ePrint report 2010/293. May 2010.

[35] J. Patarin. Security of random Feistel schemes with 5 or more rounds. *CRYPTO 2004*, LNCS 3152, Springer, pp. 106–122, 2004.

[36] J. Patarin, B. Gittins, and J. Treger. Increasing block sizes using Feistel networks: the example of the AES. *Cryptography and Security: From Theory to Applications*, LNCS 6805, Springer, pp. 67–82, 2012.

[37] B. Preneel. Personal communications, via D. Bernstein. CAESAR competition: partial status of submissions (draft output of Dagstuhl discussion session 9 Jan 2014). Set of slides.

[38] P. Rogaway. Authenticated-encryption with associated-data. *ACM Conference on Computer and Communications Security*, ACM Press, pp. 98–107, 2002.

[39] P. Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. *ASIACRYPT 2004*, LNCS 3329, Springer, pp. 16–31, 2004.

[40] P. Rogaway. A synopsis of format-preserving encryption. Unpublished manuscript, available from the author's webpage. Mar 2010.

[41] P. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. *EUROCRYPT 2006*, LNCS 4004, Springer, pp. 373–390, 2006. Also Cryptology ePrint Report 2006/221, retitled, Deterministic authenticated-encryption: a provable-security treatment of the key-wrap problem. 2006.

[42] M. Simplício, P. Barbuda, P. Barreto, T. Carvalho, and C. Margi. The MARVIN message authentication code and the LETTERSOUP authenticated encryption scheme. *Security and Communication Networks*, 2(2), pp. 165–180, 2009.

[43] M-J. Saarinen and J-P. Aumasson. The BLAKE2 cryptographic hash and MAC. Internet Draft draft-saarinen-blake2-06. IETF. August 25, 2015. https://tools.ietf.org/html/draft-saarinen-blake2-06

[44] M. Simplício and P. Barreto. Revisiting the security of the ALRED Design and Two of Its Variants: Marvin and LetterSoup. *IEEE Transactions on Information Theory*, 58(9), pp. 6223–6238, 2012.

[45] R. Struik. AEAD ciphers for highly constrained networks. DIAC 2013 presentation. Chicago, Illinois, USA. Aug 13, 2013.

```
500 algorithm BLAKE2b($M$)                                          // BLAKE2b hashing
501 IV ← $\mathrm{IV}_0 \| \cdots \| \mathrm{IV}_7$;  $H_0 \gets \mathrm{IV} \oplus ([48] \| [0] \| [1] \| [1] \| [0]^{60})$
502 $M_0 \ldots M_{m-1} \gets M$ where $m \gets \lceil \|M\|/128 \rceil$ and $\|M_0\| = \cdots = \|M_{m-2}\| = 128$
503 for $i = 0$ to $m-1$ do $\ell_i \gets \|M_0 \cdots M_i\|$;  $H_{i+1} \gets \mathrm{Compress}(H_i, \mathrm{pad}(M_i), \ell_i)$ od
504 return $H_m[1 \ldots 384]$
```

Figure 9: **The code of BLAKE2b.** See Table 2 for the constants $\mathrm{IV}_0, \ldots, \mathrm{IV}_7$, and see the text for the description for the compression function Compress of BLAKE2b.

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

Table 1: **Permutations of** $\{0, 1, 2, \ldots, 15\}$ **used by BLAKE2b.**

IV_0 = `0x6a09e667f3bcc908`	IV_1 = `0xbb67ae8584caa73b`
IV_2 = `0x3c6ef372fe94f82b`	IV_3 = `0xa54ff53a5f1d36f1`
IV_4 = `0x510e527fade682d1`	IV_5 = `0x9b05688c2b3e6c1f`
IV_6 = `0x1f83d9abfb41bd6b`	IV_7 = `0x5be0cd19137e2179`

Table 2: **The 8-byte constants** $\mathrm{IV}_0, \ldots, \mathrm{IV}_7$ **used by BLAKE2b, written in hexadecimal.**

A Specification of BLAKE2b

For completeness, we give the full description of BLAKE2b, closely following the BLAKE2b documentation [3, 43]. For each byte string X, let $\|X\|$ denote the byte length of X. For each string X with $\|X\| \le 128$, let $\mathrm{pad}(X)$ denote the string Y with $\|Y\| = 128$ obtained by appending zero or more 0-bits to X. Let $X \ggg \ell$ denote the right circular shift by ℓ bits on the string X. For any 64-bit strings X and Y, let $X + Y$ denote the string obtained by regarding X and Y as integers, adding them modulo 2^{64}, then regarding the resulting integers as a string. BLAKE2b uses little-endian convention in parsing between integers and strings.

The specification of the BLAKE2b hash is given in Figure 9, where the compression function Compress is as follows. It takes as input a 64-byte string h, a 128-byte message M, and an integer $0 \le t < 2^{128}$. Let $f_0 = 0^{64}$ if t is a multiple of 128, and let $f_0 = 1^{64}$ otherwise. Let $f_1 = 0^{64}$. Parse t as $t_0 t_1$, where $|t_0| = |t_1| = 64$. Recall that BLAKE2b uses little-endian convention, so t_0 contains the least significant bits of t. Parse h as $h_0 \cdots h_7$, and M as $M_0 \cdots M_{15}$, where $|h_0| = \cdots = |h_7| = |M_0| = \cdots = |M_{15}| = 64$. Initialize $(v_0, \ldots, v_{15}) \gets (h_0, \ldots, h_7, \mathrm{IV}_0, \mathrm{IV}_1, \mathrm{IV}_2, \mathrm{IV}_3, t_0 \oplus \mathrm{IV}_4, t_1 \oplus \mathrm{IV}_5, f_0 \oplus \mathrm{IV}_6, f_1 \oplus \mathrm{IV}_7)$, where the constants $\mathrm{IV}_0, \ldots, \mathrm{IV}_7$ are defined in Table 2. One then processes the state $(v_0, \ldots, v_{15})$ in 12

```
**algorithm** Decipher($K, \boldsymbol{T}, C$)                                   // AEZ deciphering
**if** $|C| < 256$ **then return** Decipher-AEZ-tiny($K, \boldsymbol{T}, C$)
**if** $|C| \geq 256$ **then return** Decipher-AEZ-core($K, \boldsymbol{T}, C$)
```

```
**algorithm** Decipher-AEZ-tiny($K, \boldsymbol{T}, C$)                          // AEZ-tiny deciphering
$\mu \leftarrow |C|$;  $n \leftarrow \mu/2$;  $\Delta \leftarrow$ AEZ-hash($K, \boldsymbol{T}$)
**if** $\mu < 128$ **then** $C \leftarrow C \oplus (\mathrm{E}_K^{0,3}(\Delta \oplus (C0^* \vee 10^*)) \wedge 10^*)$ **fi**
**if** $\mu = 8$ **then** $k \leftarrow 24$ **else if** $\mu = 16$ **then** $k \leftarrow 16$ **else if** $\mu < 128$ **then** $k \leftarrow 10$ **else** $k \leftarrow 8$ **fi**
$L \leftarrow C[1..n]$;  $R \leftarrow C[n+1..\mu]$;  **if** $\mu \geq 128$ **then** $i \leftarrow 6$ **else** $i \leftarrow 7$ **fi**
**for** $j \leftarrow k-1$ **downto** 0 **do** $R' \leftarrow L \oplus ((\mathrm{E}_K^{0,i}(\Delta \oplus R10^* \oplus [j]_{128}))[1..n])$;  $L \leftarrow R$;  $R \leftarrow R'$ **od**
$M \leftarrow R \,\|\, L$;  **return** $M$
```

```
**algorithm** Decipher-AEZ-core($K, \boldsymbol{T}, C$)                          // AEZ-core deciphering
$\Delta \leftarrow$ AEZ-hash($K, \boldsymbol{T}$)
$C_1 C_1' \cdots C_m C_m' \; C_{\mathsf{uv}} \; C_{\mathsf{x}} C_{\mathsf{y}} \leftarrow C$ where $|C_1| = \cdots = |C_m'| = |C_{\mathsf{x}}| = |C_{\mathsf{y}}| = 128$ and $|C_{\mathsf{uv}}| < 256$
$d \leftarrow |C_{\mathsf{uv}}|$;  **if** $d \leq 127$ **then** $C_{\mathsf{u}} \leftarrow C_{\mathsf{uv}}$; $C_{\mathsf{v}} \leftarrow \varepsilon$ **else** $C_{\mathsf{u}} \leftarrow C_{\mathsf{uv}}[1..128]$; $C_{\mathsf{v}} \leftarrow C_{\mathsf{uv}}[129..|C_{\mathsf{uv}}|]$ **fi**
**for** $i \leftarrow 1$ **to** $m$ **do** $W_i \leftarrow C_i \oplus \mathrm{E}_K^{1,i}(C_i')$;  $Y_i \leftarrow C_i' \oplus \mathrm{E}_K^{0,0}(W_i)$ **od**
**if** $d = 0$ **then** $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathbf{0}$ **else if** $d \leq 127$ **then** $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathrm{E}_K^{0,4}(C_{\mathsf{u}}10^*)$
**else** $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathrm{E}_K^{0,4}(C_{\mathsf{u}}) \oplus \mathrm{E}_K^{0,5}(C_{\mathsf{v}}10^*)$ **fi**
$S_{\mathsf{x}} \leftarrow C_{\mathsf{x}} \oplus \Delta \oplus Y \oplus \mathrm{E}_K^{0,2}(C_{\mathsf{y}})$;  $S_{\mathsf{y}} \leftarrow C_{\mathsf{y}} \oplus \mathrm{E}_K^{-1,2}(S_{\mathsf{x}})$;  $S \leftarrow S_{\mathsf{x}} \oplus S_{\mathsf{y}}$
**for** $i \leftarrow 1$ **to** $m$ **do** $S' \leftarrow \mathrm{E}_K^{2,i}(S)$; $X_i \leftarrow W_i \oplus S'$; $Z_i \leftarrow Y_i \oplus S'$; $M_i' \leftarrow X_i \oplus \mathrm{E}_K^{0,0}(Z_i)$; $M_i \leftarrow Z_i \oplus \mathrm{E}_K^{1,i}(M_i')$ **od**
**if** $d = 0$ **then** $M_{\mathsf{u}} \leftarrow M_{\mathsf{v}} \leftarrow \varepsilon$;  $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathbf{0}$
**else if** $d \leq 127$ **then** $M_{\mathsf{u}} \leftarrow C_{\mathsf{u}} \oplus \mathrm{E}_K^{-1,4}(S)$;  $M_{\mathsf{v}} \leftarrow \varepsilon$;  $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathrm{E}_K^{0,4}(M_{\mathsf{u}}10^*)$
**else** $M_{\mathsf{u}} \leftarrow C_{\mathsf{u}} \oplus \mathrm{E}_K^{-1,4}(S)$;  $M_{\mathsf{v}} \leftarrow C_{\mathsf{v}} \oplus \mathrm{E}_K^{-1,5}(S)$;  $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathrm{E}_K^{0,4}(M_{\mathsf{u}}) \oplus \mathrm{E}_K^{0,5}(M_{\mathsf{v}}10^*)$ **fi**
$M_{\mathsf{y}} \leftarrow S_{\mathsf{x}} \oplus \mathrm{E}_K^{-1,1}(S_{\mathsf{y}})$;  $M_{\mathsf{x}} \leftarrow S_{\mathsf{y}} \oplus \Delta \oplus X \oplus \mathrm{E}_K^{0,1}(M_{\mathsf{y}})$
**return** $M_1 M_1' \cdots M_m M_m' \; M_{\mathsf{u}} M_{\mathsf{v}} \; M_{\mathsf{x}} M_{\mathsf{y}}$
```

Figure 10: **AEZ deciphering routines.**

rounds. In each round, we run the following sequence of operations:

$$\mathrm{G}_0(v_0, v_4, v_8, v_{12});\ \ \mathrm{G}_1(v_1, v_5, v_9, v_{13});\ \ \mathrm{G}_2(v_2, v_6, v_{10}, v_{14});\ \ \mathrm{G}_3(v_3, v_7, v_{11}, v_{15});$$
$$\mathrm{G}_4(v_0, v_5, v_{10}, v_{15});\ \ \mathrm{G}_5(v_1, v_6, v_{11}, v_{12});\ \ \mathrm{G}_6(v_2, v_7, v_8, v_{13});\ \ \mathrm{G}_7(v_3, v_4, v_9, v_{14})$$

In round r, the operation $G_i(a, b, c, d)$, with $i \in \{0, 1, \ldots, 7\}$, modifies the 64-bit variables a, b, c, d as follows:

$$j \leftarrow \sigma_{r \bmod 10}(2i);\ \ a \leftarrow a + b + M_j;\ \ d \leftarrow (d \oplus a) \ggg 32;\ \ c \leftarrow c + d;\ \ b \leftarrow (b \oplus c) \ggg 24;$$
$$j \leftarrow \sigma_{r \bmod 10}(2i+1);\ \ a \leftarrow a + b + M_j;\ \ d \leftarrow (d \oplus a) \ggg 16;\ \ c \leftarrow c + d;\ \ b \leftarrow (b \oplus c) \ggg 63$$

where the permutations $\sigma_0, \ldots, \sigma_9$ on $\{0, 1, \ldots, 15\}$ are specified in Table 1. At the end of the 12th round, output $(h_0 \oplus v_0 \oplus v_8) \,\|\, (h_1 \oplus v_1 \oplus v_9) \,\|\, \cdots \,\|\, (h_7 \oplus v_7 \oplus v_{15})$.

B Specification of AEZ deciphering algorithms

For completeness, in Figure 10 we give the code for the deciphering routines of AEZ.