AEZ v2: Authenticated Encryption by Enciphering

Viet Tung Hoang
University of Maryland
Georgetown University
tvhoang@umd.edu

Ted Krovetz
Sacramento State
ted@krovetz.net

Phillip Rogaway
UC Davis
ETH Zürich
rogaway@cs.ucdavis.edu

August 21, 2014

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 length expansion. AEZ is parallelizable and its computational cost is close to that of AES-CTR. On a recent Intel processor (Haswell), AEZ runs at about 0.75 cpb on adequately long messages.

The latest version of this document, and any other related material, can be found on the AEZ homepage: http://www.cs.ucdavis.edu/~rogaway/aez

Contents

0 Introduction **1**

1 Specification **3**

1.1 Notation 3

1.2 Parameters 4

1.3 Pseudocode 5

1.4 Usage cap 9

2 Security Goals **9**

3 Security Analysis **13**

4 Features **15**

5 Design Rationale **17**

6 Intellectual Property **18**

7 Consent **19**

8 Revision History **19**

References **20**

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 T to a ciphertext $Y = \text{Encipher}(K, T, X)$ having the same length as X. Going backwards, one can recover $X = \text{Decipher}(K, T, Y)$. 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 A, and an authenticator length ABYTES to a ciphertext $C = \text{Encrypt}(K, N, A, M)$ that is ABYTES bytes longer than M. Calling $\text{Decrypt}(K, N, A, C)$ returns either a string M or an indication of invalidity. The security property we seek is that of a *robust* authenticated-encryption scheme, a new and very strong notion that implies protection of the privacy and authenticity of M and the authenticity of N and A, and must do so to the maximal extent possible even if nonces get reused ("misuse resistance" [35]), the authenticator length (ABYTES) 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 of the form described 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 T that encodes N, A, and all parameters. Decryption works by deciphering the presented string (again using the tweak determined by N and A) 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 number of authenticator bytes 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, A, M) tuples, and 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 the overhead we pay is modest: AEZ isn't much slower than OCB.

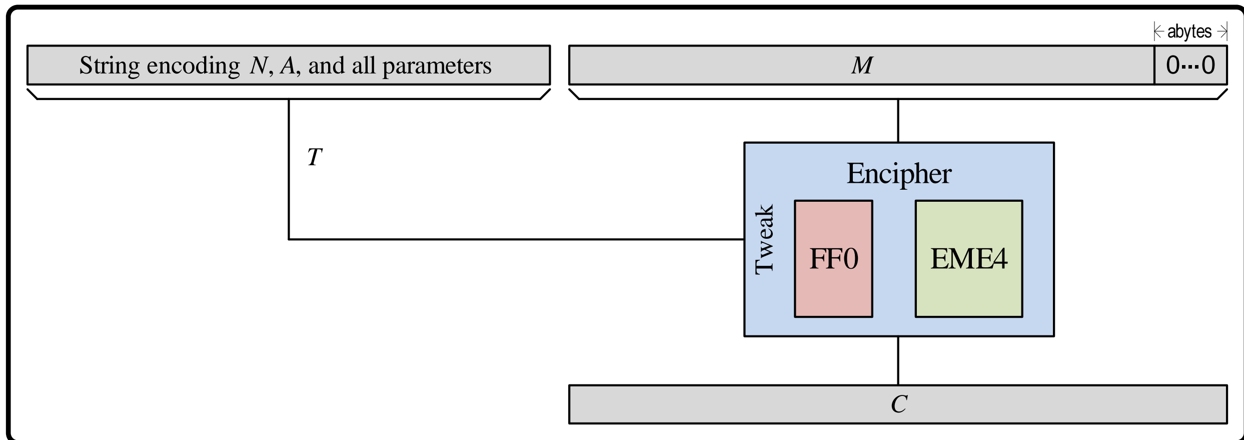

Figure 1: **High-level structure of AEZ**. After appending to the message a block of zeros we encipher it using a tweak that encodes the nonce, associated data, and parameters. The enciphering method depends on the length of what's being enciphered. Short messages are enciphered by FF0; longer ones, by EME4. The boxes for Encipher, FF0, and EME4 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 an algorithm we call FF0, which builds on FFX [4, 12]. When it's 32 bytes or more we use an algorithm we call EME4, which builds on EME [14, 15] and OTR [23].

FF0 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.

EME4 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 mode.

The design of EME4 employs a paradigm we call *accelerated provable-security*. One begins by designing a cryptographic scheme based on a well-known tool, say a tweakable blockcipher (TBC) [21]. One proves security assuming that the tool meets some standard assumption—here, that the TBC is secure as a tweakable PRP. As a heuristic final step, the TBC is *selectively* instantiated by a *scaled-down* primitive: most often, we use a reduced-round version of AES, instead of AES itself, to build the TBC. 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 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 help identify individuals who can't distinguish an S from a Z.

1 Specification

1.1 Notation

Numbers and strings. A *number* means a nonnegative integer, $\mathbb{N} = \{0, 1, 2, \ldots\}$ For numbers $i \le j$, let $[i..j]$ be the numbers $\{i, i+1, \ldots, j\}$. 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 $A \parallel B$. By 0^n we mean the string of n zero bits. If $\mathcal{X}$ is a set of strings then $\mathcal{X}^*$ is all strings, including ε, formed by concatenating elements of $\mathcal{X}$. The bitwise xor of equal-length strings A and B is denoted $A \oplus B$. For the xor of unequal-length strings, first drop the necessary number of rightmost bits from the longer ($10 \oplus 0100 = 11$). For X a string, let $X10^* = X \parallel 10^*$ be $X10^p$ with p the smallest number such that 128 divides $|X| + p + 1$. 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), $\mathrm{msb}(X) = X(1)$, $\mathrm{lsb}(X) = X(n)$, and $X(i..j) = X(i) \cdots X(j)$.

A byte is eight bits. The set of all bytes is denoted $\textsc{Byte}$. A byte string is an element of $\textsc{Byte}^*$. If X is a byte string then X^n is the n-byte string that repeats X a total of n times. The byte length of $X \in \textsc{Byte}^*$ is $\|X\| = |X|/8$. When $X \in \textsc{Byte}^*$ and $1 \le i \le j \le \|X\|$ then $X[i]$ is its ith byte (indexing from the left starting at 1) and $X[i..j]$ is the substring of X that runs from its ith to jth byte (inclusive). Let $[n]$ be the byte representing $n \bmod 256$ (so $[0]^{16} = 0^{128}$ and $[1]^{16} = (00000001)^{16}$) and let $[n]_t$ be the t-byte string representing $n \bmod 2^{8t}$ in t bytes.

If A is a string we write $(A_0, \ldots, A_m) \leftarrow A$ to indicate that m and $A_0, \ldots, A_m$ are the unique values such that $A_0 \cdots A_m = A$ and $|A_0| = \cdots = |A_{m-1}| = 128$ and $|A_m| \le 128$, with $A_m = \varepsilon$ only when $A = \varepsilon$. If $|M| \ge 256$ write $(M_0, M_0', \ldots, M_m, M_m', M_*, M_{**}) \leftarrow M$ to indicate that m and $M_0, M_0', \ldots, M_m, M_m', M_*, M_{**}$ are the unique values such that $M_0 M_0' \cdots M_m M_m' M_* M_{**} = M$ and $|M_0| = |M_0'| = \cdots = |M_m| = |M_m'| = 128$ and either $|M_*| = 128$ and $|M_{**}| < 128$, or $|M_*| < 128$ and $|M_{**}| = 0$. So if M consists of an even number of blocks then $M_* = M_{**} = \varepsilon$, and if M consists of an odd number of blocks then $M_{**} = \varepsilon$.

A block is 128 bits and a block string is a sequence of blocks. Let $\mathbf{0} = 0^{128}$. If $X = a_1 \cdots a_{128}$ is a block ($a_i \in \{0,1\}$) then $X \ll 1 = a_2 \cdots a_{128}\, 0$ and $X \gg 1 = 0\, a_1 \cdots a_{127}$. If $X = X[1] \cdots X[16]$ is a block then $\mathrm{rev}(X) = X[16] \cdots X[1]$ is its byte-reversal. For $n \in \mathbb{N}$ and $X \in \{0,1\}^{128}$ define $\mathrm{mul}(n, X)$ and nX by saying that $\mathrm{mul}(0, X) = \mathbf{0}$, $\mathrm{mul}(1, X) = X$, $\mathrm{mul}(2, X) = (X \ll 1) \oplus [135 \cdot \mathrm{msb}(X)]_{16}$, $\mathrm{mul}(2n, X) = \mathrm{mul}(2, \mathrm{mul}(n, X))$, and $\mathrm{mul}(2n+1, X) = \mathrm{mul}(2n, X) \oplus X$, and $nX = \mathrm{rev}(\mathrm{mul}(n, \mathrm{rev}(X)))$.

AES and AES4. We assume familiarity with AES. We write $\mathrm{AES}_K(X) = \mathrm{AES}(K, X)$ for AES encipherment of the 128-bit plaintext X using the 128-bit key K. 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. Let $\texttt{aesenclast}(X, K)$ be the same except omit `MixColumns`. 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 $\mathrm{AES}_{\boldsymbol{K}}(X) = \mathrm{AES}(\boldsymbol{K}, X)$ like we defined AES4 but compose nine rounds of `aesenc` then a round of `aesenclast`. Thus $\mathrm{AES}_K(X) = \mathrm{AES}_{\boldsymbol{K}}(X)$ where $\boldsymbol{K} = \texttt{expand}(K)$ is the vector of 11 subkeys produced from K by the AES key schedule.

symbol	comments
M	Plaintext. $M \in \text{BYTE}^*$
C	Ciphertext. $C \in \text{BYTE}^*$
K	Key. $K \in \text{BYTE}^*$. An entropy-extraction algorithm maps arbitrary keys to 32 bytes
N	Nonce (aka: public sequence number). $N \in \text{BYTE}^*$. $\|N\| \le 12$ recommended
A	Associated data. $A \in \text{BYTE}^*$. Users should use the empty string if they don't need the AD
ABYTES	Authenticator length. ABYTES $\in [0..16]$. Default is 16. C will be ABYTES longer than M
EXTNS	Extensions directive. EXTNS $\in \text{BYTE}^3$. Default is $[0]^3$. Other values direct pre/post processing

Figure 2: **Arguments and parameters**. The first five values are arguments to Encrypt( ) or Decrypt( ). The next two values are parameters. The current document only specifies the behavior of AEZ when EXTNS = $[0]^3$; other values will direct the invocation of integrated extensions.

1.2 Parameters

We'll take *parameter* to mean "a value on which AEZ encryption depends that we are expecting, independent of any particular API, to be held constant throughout some long-lived context." Thus we will not regard KEYBYTES as an AEZ parameter (we permit keys of any length), nor NPUBBYTES (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. With this understanding, we will regard AEZ as having two parameters (and even these could be considered as arguments instead of parameters). See Figure 2.

- The *authenticator length*, ABYTES, quantifies the authenticity provided. It also determines how much longer a ciphertext is than its plaintext. The possible values of ABYTES are integers between 0 and 16 (inclusive). While we call ABYTES a parameter, we do not insist that it be held constant throughout a session; a user is free to change it with each encryption. Still, we expect most applications to fix ABYTES.
- The *extensions directive*, EXTNS, will, in the future, unlock capabilities that have traditionally been seen as outside the scope of an encryption scheme's functionality. These include secret nonces (secret message numbers), plaintext-length obfuscation (via a specified padding regime), and encoding ciphertexts into a prescribed alphabet. These extensions will be realized by a wrapper that keylessly transforms a plaintext, AEZ encrypts it, then keylessly transforms the result. Pre- and post-processing is effectively absent (that is, the identity) when EXTNS = $[0]^3$. A document defining AEZ extensions will be released later.

AEZ parameters have defaults: ABYTES = 16 and EXTNS = $[0]^3$. The only named parameter set, `aez`, uses these. A conforming AEZ implementation is free to select defaults different from the ones given, and to let the user select ABYTES and/or EXTNS through the argument list of procedures and to let these values vary across calls. In any context where the key length or nonce length are *required* to be fixed, we select byte lengths for these of KEYBYTES = 16 and NPUBBYTES = 12.

We emphasize that AEZ itself does not support secret nonces. Readers who find fault with calling EXTNS an AEZ parameter but failing to specify pre- and post-processing behavior when it takes on a non-default value should regard EXTNS as the constant $[0]^3$.

1.3 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 A, N, and the parameters. The values are encoded into the tweak in a manner that enhances the efficiency of their processing—in particular, the AD always starts at the second block and ends on a block boundary, while the nonce is packed into the first block as long as this is possible. 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 FF0, while those of 32 bytes or more are enciphered using EME4.

Roughly following FFX [4, 12], algorithm EncipherFF0 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 AES or AES4 based on the first component of the tweak, employing the full AES only for tweaks beginning with a -1. The EncipherFF0 routine is illustrated at the bottom-left of Figure 5 for the setting where messages have 16 or more bytes.

A novel feature of EncipherFF0 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 [30]. Compared to swapping two points, our approach has the benefit that the natural implementation is constant-time.

EncipherEME4 melds EME [14, 15], OTR [23], and a variety of other ideas. Refer to the illustration at the top left of Figure 5. Consider the simplest case, where the message $M = M_0 M_0' M_1 M_1' \cdots M_m M_m'$ has an even number of blocks. Each rectangle with a pair of numbers is a tweakable PRP, the label being the tweak and the key K left implicit. Each successive pair of blocks $M_i M_i'$ (for $i \geq 1$) is initially subjected to a two-round Feistel network. This both begins the scrambling of $M_i M_i'$ and yields a value X that is a computational almost-xor-universal hash of $M_1 M_1' \cdots M_m M_m'$. The first pair of blocks are now processed, but where X initially offsets one of them. This both begins the scrambling of $M_0 M_0'$ and yields the value S that is a computational almost-universal hash of all of M. Note the same S can be computed when deciphering. The TBC calls of the middle row now inject into the Feistel network a random-looking position and S-dependent value. It should not be surprising that two additional Feistel rounds suffice to make the construction a strong PRP—provably so, under the assumptions we have stated. We call this construction $\widetilde{\text{EME4}}[\text{E}]$. It is the generalization of EME4 that employs an arbitrary TBC.

Messages with an odd number of blocks are processed as illustrated on the top-left plus and top-right figures. Messages with an even number of blocks and the final block fragmentary are processed as illustrated on the top-left plus bottom-right figures.

100 **algorithm** Encrypt(K, N, A, M) // AEZ authenticated encryption

101 $X \leftarrow M \,\|\, [0]^{\text{ABYTES}}$

102 $T \leftarrow \text{Format}(N, A)$

103 **if** $M = \varepsilon$ **then return** $\text{AMac}(K, T)[1..\text{ABYTES}]$

104 $C \leftarrow \text{Encipher}(K, T, X)$

105 **return** C

110 **algorithm** Decrypt(K, N, A, C) // AEZ authenticated decryption

111 $T \leftarrow \text{Format}(N, A)$

112 **if** $\|C\| \le \text{ABYTES}$ **then if** $(C = \text{AMac}(K, T)[1..\text{ABYTES}])$ **then return** ε **else return** $\perp$

113 $X \leftarrow \text{Decipher}(K, T, C)$

114 $M \,\|\, Z \leftarrow X$ where $\|Z\| = \text{ABYTES}$

115 **if** $(Z = [0]^{\text{ABYTES}})$ **then return** M **else return** $\perp$

120 **algorithm** Format(N, A) // Encode inputs and parameters

121 **if** $\|N\| \le 11$ **then return** $00 \,\|\, (\text{ABYTES})_6 \,\|\, \text{EXTNS} \,\|\, N \,\|\, 10^* \,\|\, A$

122 **if** $\|N\| = 12$ **then return** $01 \,\|\, (\text{ABYTES})_6 \,\|\, \text{EXTNS} \,\|\, N \,\|\, A$

123 **if** $\|N\| \ge 13$ **then return** $10 \,\|\, (\text{ABYTES})_6 \,\|\, \text{EXTNS} \,\|\, N[1..12] \,\|\, A \,\|\, 10^* \,\|\, N[13..\|N\|] \,\|\, [\|N\|]_8$

200 **algorithm** Encipher(K, T, X) // AEZ enciphering

201 **if** $\|X\| < 32$ **then return** $\text{EncipherFF0}(K, T, X)$

202 **if** $\|X\| \ge 32$ **then return** $\text{EncipherEME4}(K, T, X)$

210 **algorithm** EncipherFF0(K, T, M) // FF0 enciphering

211 $m \leftarrow |M|;\;\; n \leftarrow m/2;\;\; \Delta \leftarrow \text{AHash}(K, T)$

212 **if** $m = 8$ **then** $k \leftarrow 24$ **else if** $m = 16$ **then** $k \leftarrow 16$ **else if** $m < 128$ **then** $k \leftarrow 10$ **else** $k \leftarrow 8$

213 $L \leftarrow M(1..n);\;\; R \leftarrow M(n+1..m);$ **if** $m \ge 128$ **then** $j \leftarrow 5$ **else** $j \leftarrow 6$

214 **for** $i \leftarrow 0$ **to** $k-1$ **do** $R' \leftarrow L \oplus ((\text{E}_K^{0,j}(\Delta \oplus R10^* \oplus [i]_{16}))(1..n));\;\; L \leftarrow R;\;\; R \leftarrow R'$ **od**; $C \leftarrow R \,\|\, L$

215 **if** $m < 128$ **then** $C \leftarrow C \oplus (\text{E}_K^{0,7}(\Delta \oplus (C \vee 10^*)) \wedge 10^*)$

216 **return** C

220 **algorithm** EncipherEME4(K, T, M) // EME4 enciphering

221 $\Delta \leftarrow \text{AHash}(K, T);\;\; (M_0, M_0', \ldots, M_m, M_m', M_*, M_{**}) \leftarrow M;\;\; d \leftarrow |M| \bmod 256$

222 **for** $i \leftarrow 1$ **to** m **do** $X_i' \leftarrow M_i \oplus \text{E}_K^{1,i}(M_i');\;\; X_i \leftarrow M_i' \oplus \text{E}_K^{0,0}(X_i')$ **od**

223 **if** $d = 0$ **then** $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathbf{0}$ **else if** $d \le 127$ **then** $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \text{E}_K^{0,3}(M_*10^*)$

224 **else** $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \text{E}_K^{0,3}(M_*) \oplus \text{E}_K^{0,4}(M_{**}10^*)$ **fi**

225 $R \leftarrow M_0 \oplus \text{E}_K^{0,1}(M_0' \oplus X) \oplus \Delta;\;\; R' \leftarrow M_0' \oplus \text{E}_K^{-1,1}(R) \oplus X;\;\; S \leftarrow R \oplus R'$

226 **for** $i \leftarrow 1$ **to** m **do** $Z \leftarrow \text{E}_K^{2,i}(S);\; Y_i \leftarrow X_i' \oplus Z;\; Y_i' \leftarrow X_i \oplus Z;\; C_i' \leftarrow Y_i \oplus \text{E}_K^{0,0}(Y_i');\; C_i \leftarrow Y_i' \oplus \text{E}_K^{1,i}(C_i')$ **od**

227 **if** $d = 0$ **then** $C_* \leftarrow C_{**} \leftarrow \varepsilon;\;\; Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathbf{0}$

228 **else if** $d \le 127$ **then** $C_* \leftarrow M_* \oplus \text{E}_K^{-1,3}(S);\;\; C_{**} \leftarrow \varepsilon;\;\; Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \text{E}_K^{0,3}(C_*10^*)$

229 **else** $C_* \leftarrow M_* \oplus \text{E}_K^{-1,3}(S);\;\; C_{**} \leftarrow M_{**} \oplus \text{E}_K^{-1,4}(S);\;\; Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \text{E}_K^{0,3}(C_*) \oplus \text{E}_K^{0,4}(C_{**}10^*)$ **fi**

230 $C_0'' \leftarrow R \oplus \text{E}_K^{-1,2}(R');\;\; C_0 \leftarrow R' \oplus \text{E}_K^{0,2}(C_0'') \oplus \Delta;\;\; C_0' \leftarrow C_0'' \oplus Y$

231 **return** $C_0 C_0' \cdots C_m C_m'\, C_* C_{**}$

Figure 3: **AEZ authenticated-encryption: main routines.** The tweakable blockcipher E and message authentication code AMac are defined in Figure 4. Algorithm Decipher(K, T, C), not shown, returns the unique M such that $\text{Encipher}(K, T, M) = C$. See the accompanying text for how this is computed.

300 **algorithm** $\text{AHash}(K, A)$ // AXU hash
301 $(A_0, \ldots, A_m) \leftarrow A$
302 **if** $|A_m| \bmod 128 = 0$ **then return** $\text{E}_K^{3,0}(A_0) \oplus \text{E}_K^{3,1}(A_1) \oplus \cdots \oplus \text{E}_K^{3,m}(A_m)$
303 **if** $|A_m| \bmod 128 \neq 0$ **then return** $\text{E}_K^{3,0}(A_0) \oplus \text{E}_K^{3,1}(A_1) \oplus \cdots \oplus \text{E}_K^{3,m-1}(A_{m-1}) \oplus \text{E}_K^{1,0}(A_m 10^*)$

310 **algorithm** $\text{AMac}(K, A)$ // PRF
311 **return** $\text{E}_K^{-1,5}(\text{AHash}_K(A))$

400 **algorithm** $\text{E}_K^{i,j}(X)$ // TBC on $\mathcal{T} = \{0\} \times [0..7] \cup \{1,2,3\} \times \mathbb{N}$
401 $(J, L, K_0, K_1, K_2, K_3) \leftarrow \text{Expand}(\text{Extract}(K))$
402 $\boldsymbol{k}_0 \leftarrow (K_0, K_1, K_2, K_3, \mathbf{0})$; $\boldsymbol{k}_2 \leftarrow (K_2, K_3, K_0, K_1, \mathbf{0})$
403 $\boldsymbol{k}_1 \leftarrow (K_1, K_2, K_3, K_0, \mathbf{0})$; $\boldsymbol{k}_3 \leftarrow (K_3, K_0, K_1, K_2, \mathbf{0})$
404 $\boldsymbol{K} \leftarrow (L, J, 2J, 4J, K_0, K_1, K_2, K_3, K_0, K_1, K_2)$
405 **if** $i = -1$ **then return** $\text{AES}_{\boldsymbol{K}}(X \oplus jJ)$
406 **if** $i = 0$ **or** $j = 0$ **then return** $\text{AES4}_{\boldsymbol{k}_i}(X \oplus jJ)$
407 **return** $\text{AES4}_{\boldsymbol{k}_i}(X \oplus (j \bmod 8)J \oplus 2^{\lfloor (j-1)/8 \rfloor} L)$

410 **algorithm** $\text{Extract}(K)$ // Convert key to 256 bits
411 $\boldsymbol{z} \leftarrow [0][1][2]\cdots[15]$; **for** $i \leftarrow 1$ **to** 7 **do** $C_i \leftarrow \text{AES4}_{(\boldsymbol{z},\boldsymbol{z},\boldsymbol{z},\boldsymbol{z},\boldsymbol{z})}([i]^{16})$
412 $\boldsymbol{a} \leftarrow (\mathbf{0}, C_1, C_2, C_3, \mathbf{0})$; $\boldsymbol{b} \leftarrow (\mathbf{0}, C_4, C_5, C_6, \mathbf{0})$; $C \leftarrow C_7$
413 $(I_0, \ldots, I_m) \leftarrow K$; **if** $\|I_m\| = 16$
414 **then** $J \leftarrow \text{AES4}_{\boldsymbol{a}}(I_0 \oplus C) \oplus \text{AES4}_{\boldsymbol{a}}(I_1 \oplus 2C) \oplus \text{AES4}_{\boldsymbol{a}}(I_2 \oplus 2^2 C) \oplus \cdots \oplus \text{AES4}_{\boldsymbol{a}}(I_m \oplus 2^m C)$
415 $L \leftarrow \text{AES4}_{\boldsymbol{b}}(I_0 \oplus C) \oplus \text{AES4}_{\boldsymbol{b}}(I_1 \oplus 2C) \oplus \text{AES4}_{\boldsymbol{b}}(I_2 \oplus 2^2 C) \oplus \cdots \oplus \text{AES4}_{\boldsymbol{b}}(I_m \oplus 2^m C)$
416 **else** $J \leftarrow \text{AES4}_{\boldsymbol{a}}(I_0 \oplus C) \oplus \text{AES4}_{\boldsymbol{a}}(I_1 \oplus 2C) \oplus \text{AES4}_{\boldsymbol{a}}(I_2 \oplus 2^2 C) \oplus \cdots \oplus \text{AES4}_{\boldsymbol{a}}(I_m 10^* \oplus 3C)$
417 $L \leftarrow \text{AES4}_{\boldsymbol{b}}(I_0 \oplus C) \oplus \text{AES4}_{\boldsymbol{b}}(I_1 \oplus 2C) \oplus \text{AES4}_{\boldsymbol{b}}(I_2 \oplus 2^2 C) \oplus \cdots \oplus \text{AES4}_{\boldsymbol{b}}(I_m 10^* \oplus 3C)$
418 **return** $J \| L$

420 **algorithm** $\text{Expand}(K)$ // Map 256-bit string to vector of 128-bit subkeys
421 $(J, L) \leftarrow K$; $\boldsymbol{k} \leftarrow (J, L, 2J, L, 4J)$
422 **for** $i \leftarrow 0$ **to** 3 **do** $K_i \leftarrow \text{AES4}_{\boldsymbol{k}}([i]^{16})$
423 **return** $(J, L, K_0, K_1, K_2, K_3)$

Figure 4: **The universal hash, MAC, and tweakable blockcipher used by AEZ.** The last carries out key processing that an implementation would normally do at session-setup time. An alternative "scaled-up" algorithm AEZ10 would redefine E more simply, setting $\text{E}_K^{i,j} = \text{AES}_K(iI \oplus jJ \oplus X)$ where $I = \text{AES}_K(\mathbf{0})$ and $J = \text{AES}_K(\mathbf{1})$ and restricting keys $\{0,1\}^{128}$.

At this point we could instantiate E using a standard TBC-construction based on AES: the XE method [21, 33] would do, yielding the scheme AEZ10 specified in the caption of Figure 4. At that point we would have a provably-secure enciphering scheme (for strings of 32 or more bytes) with an amortized five AES calls per pair of blocks, so 2.5 AES calls per block. The cost would be similar to that of EME2 [14], spending an extra 0.5 AES calls per block but avoiding the repeated doubling and the need for an AES-inverse.

But suppose we shatter our abstraction boundary and look at all that is really going on to encipher M in AEZ10. Then the design starts to seem like major overkill: in effect, each block M_i is processed with 30 rounds of AES (ten of them shared with a neighboring block)—not counting the 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 apparent overkill, EME4 selectively prunes some of the AES calls that AEZ10 would

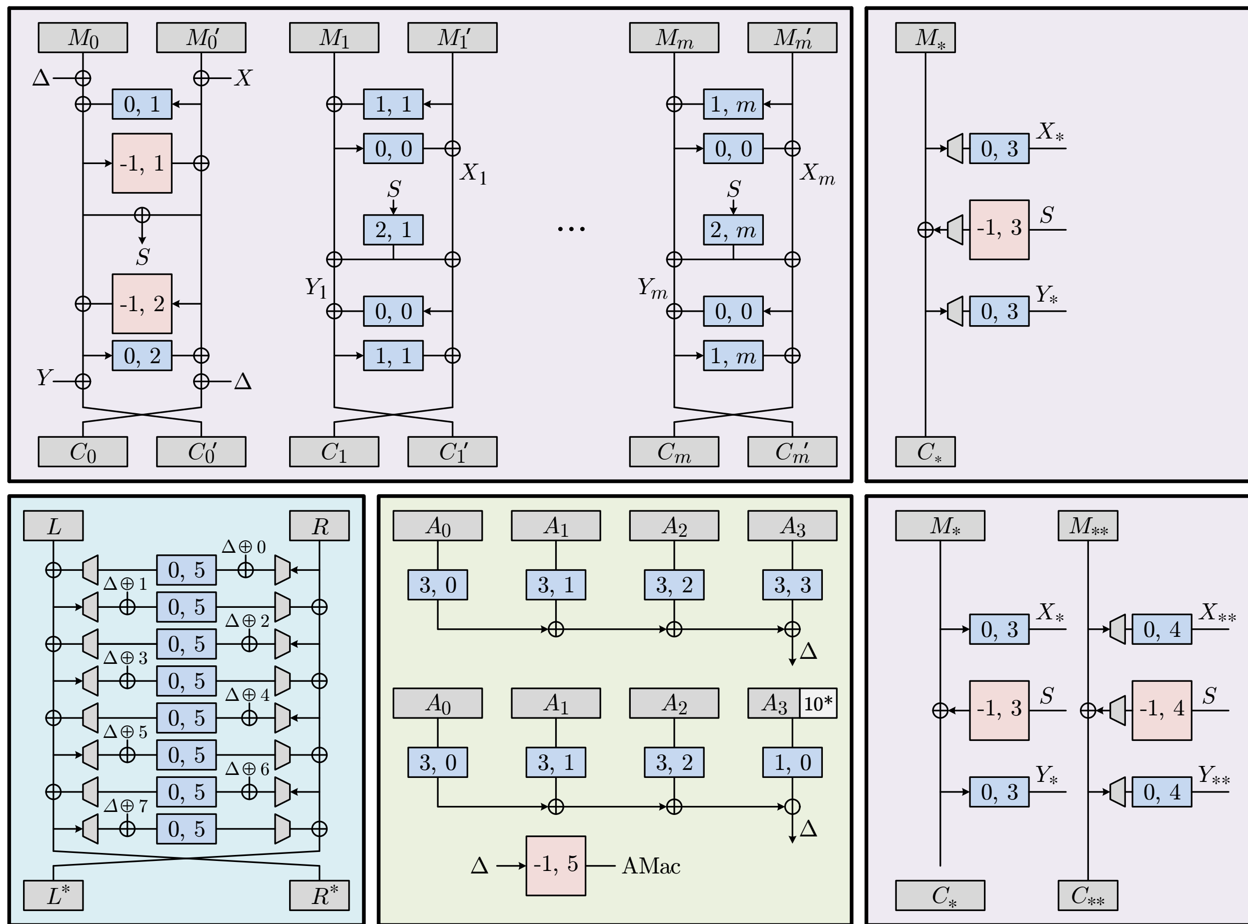

Figure 5: **Illustration of AEZ.** Rectangles with pairs of numbers are tweakable blockciphers, the pair being that tweak (the key, always K, is omitted). **Top left:** EME4 when the plaintext has an even number of blocks. **Top left + top right:** EME4 when the plaintext has an even number of full blocks and a fractional final block. **Top left + bottom right:** EME4 when the plaintext has an odd number of full blocks. **Bottom left:** FF0 when the plaintext has 16 or more bytes. **Bottom right:** AHash when the message has a full (top) or partial (bottom) final block; along with the turning of the hash into a MAC.

make, using AES4 instead of AES. In particular, we prune invocations where we are trying to achieve good xor-universal hashing. We leave enough AES rounds for confusion/diffusion processing of M so that each block M_i is effectively processed with 12 AES rounds, eight of these subsequent to injection of the highly-unpredictable S. The key steps in calculating S are not pruned, nor is the TBC used to mask any final fragment.

Deciphering. We define Decipher(K, T, Y) as the unique X such that Encipher$(K, T, X) = Y$. Logically, this is all we need say for the specification to be well-defined, so we omit writing out the implementing pseudocode. Still, that pseudocode is easy to describe. The reason this is so is that enciphering and deciphering are highly symmetric for both FF0 and EME4.

FF0 deciphering is identical to FF0 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 DecipherFF0(K, T, M) (the M now representing ciphertext) is identical to EncipherFF0(K, 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.

EME4 deciphering is identical to EME4 enciphering except that we must take the column-1 tweaks in reverse order. Specifically, a routine DecipherEME4(K, T, M) (the M now representing ciphertext) is identical to EncipherEME4(K, T, M) except we must swap tweaks $(0, 1)$ and $(0, 2)$, and we must swap tweaks $(-1, 1)$ and $(-1, 2)$. These four tweaks appear at lines 225 and 230.

Key processing. For the users' convenience, AEZ allows keys of any length. Using procedure Extract, the provided key is processed into 32 bytes using an almost-universal hash function with a fixed but "random" key. The approach is rooted in the leftover hash lemma [2, 10, 16]. The hash we select is similar to that used in PMAC [5]. Using procedure Expand, we stretch the 32-byte result from the entropy extraction to obtain the additional key material we will need. The method is basically counter mode, but still based on AES4. The extract-then-expand approach is traditional, and is that used by NIST recommendation SP 800-56C [7].

Hashing and MACing. We employ a MAC, the one we call AMac, only for the special case of a user enciphering an empty message. This is treated as a special case only for efficiency reasons, so as to make $F_K(X) = \text{Encrypt}(K, \varepsilon, X, \varepsilon)$ an attractive MAC. The chosen MAC is constructed in the Carter-Wegman tradition, employing a simple and parallelizable AES4-based universal hash function. That hash function, AHash, is exactly what is used to compute Δ, the distillation of the AD, nonce, and parameters needed by FF0 and EME4. The hash and MAC we use are depicted at the bottom-middle of Figure 5.

The tweakable blockcipher. The TBC used by AEZ employs a tweak (i, j) with $-1 \le i \le 3$ and $j \ge 0$. The first component selects between use of AES ($i = -1$) and AES4 ($i \ge 0$). Either way, the construction is based on XE [21, 33]. But we cap the number of different multiple of J that might be needed in order that a small and fixed amount of precomputation (to compute J, $2J$, and $4J$) will suffice. After each such phase (eight successive tweaks), we double L and add it in.

1.4 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, A, M) with a given key K, AEZ is acting on $\|N\| + \|A\| + \|M\|$ 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 [35]. In an MRAE scheme, repeating a nonce will violate privacy

only insofar as repetitions of (N, A, M) triples will be identified as such. It will not compromise authenticity at all. SIV [35] 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 [3].

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 113. Achieving robust AE implies that no harm would come of returning $(X, \perp)$ instead of $\perp$ at line 115.

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 any amount of plaintext 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 plaintext expansion* ($\tau = 8 \cdot \textsc{abytes}$). The statement is required to hold even in the face of decryption leaking some specified information. An upcoming academic paper defines and investigates this notion of *robust AE*. 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*, $|\mathcal{E}_K^{N,A}(M)| = |M| + \tau$, 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, A, M), returns $\pi_{N,A}(M)$ where, for each N, A, the function $\pi_{N,A}$ is a uniformly selected random injection from m-bit strings to $(m + \tau)$-bit ones. All of these functions are chosen independently. The decryption oracle, given (N, A, C), checks if there's an M such that $\pi_{N,A}(M) = C$. If so, it returns M. Otherwise it returns the distinguished value $\perp$.

The above notion is that of a *pseudorandom injection* (PRI). 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, A, C) is valid (that is, when there is an M such that $\pi_{N,A}(M) = C$), but when it's not, a simulator S gets to return what it wants. The return value may be based only on N, A, C, τ 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 a strengthening of a PRI 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 [35]. 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 with the former when τ is zero and the latter when τ is large.

Provable security. AEZ has been developed with provable security strongly in mind. The paradigm we have used is what we call the *accelerated* provable-security paradigm. 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 [21, 33], 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.

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.

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 $5s^2/2^{128} + t/2^{128}$ 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 5 and 3 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 $5s^2/2^{113} + t/2^{128}$ because of the higher maximal expected differential probability of AES4 [18] compared to an ideal hash or cipher.

Many authors prefer to think of security in terms of number-of-bits. We would summarize the $5s^2/2^{113} < s^2/2^{110}$ 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.

Security 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 certainly *are* easy distinguishing and forging attacks by the time the adversary queries AEZ with about 2^{64} blocks of message, AD,

or nonce. Similarly, we we 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 256-bit strings, rather than 128-bit strings, in our realization of extract-then-expand key processing.

3 Security Analysis

An academic paper with the relevant security proofs for AEZ is in preparation. In the meantime, 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 $\widetilde{\text{AEZ}}[\text{E}]$ be the generalization of AEZ where each E is a tweakable blockcipher (TBC) of the correct signature [21]. We can prove that $\widetilde{\text{AEZ}}[\text{E}]$ achieves RAE security as long as E is secure as a tweakable PRP. The claim assumes that $\|M\| + \text{ABYTES} \geq 16$ for each encryption query employing plaintext M, and $\|C\| \geq 16$ for each decryption query of a ciphertext C. These conditions hold automatically for the default choice of $\text{ABYTES} = 16$. With those provisos, RAE security can be proven along the following lines.

- EME4 provides a length-preserving, variable-input-length, strong PRP on $\text{Byte}^{\geq 32}$ (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 EncipherEME4 is incorporated by what can be regarded as the XEX construction [21, 33]. The underlying hash function, AHash, is almost-xor universal (AXU) when E is a PRP.
- The round functions of FF0 are derived from a tweakable blockcipher (TBC) with tweak space $\mathcal{T} = \{(i, 0) \mid i = 1, \ldots, 24\}$. We employ the XE construction [21, 33] to extend the tweak space to $\mathcal{T} \times \mathcal{N} \times \mathcal{A}$. One can then view that, for each (N, 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 [20, 28, 29], FF0 gives a length-preserving, strong tweakable-PRP on $\text{Byte}^{\geq 16} \cap \text{Byte}^{\leq 31}$, with birthday-bound distinguishing advantage.
- Once one has shown that the Encipher procedure of $\widetilde{\text{AEZ}}$ provides a length-preserving strong tweakable-PRP then $\widetilde{\text{AEZ}}$ 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 [18]. Similarly, the AXU security for AHash can be justified by viewing AHash 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 AHash is again AXU due to the fact that four-round AES with independent, uniformly random subkeys is an AXU hash [18].
- For EME4, when processing each pair of blocks M_0 and M_0', the first and last rounds only need to be AXU, due to the classic result of Naor and Reingold [25]. 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 FF0 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 $(52/2^{34})^4 \approx 2^{-113.088}$ [18]. 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 $A = A_0 A_1$ and $A' = A'_0 A'_1$. Then test if one again gets a collision with M and N but with AD values of either $A \,\|\, \mathbf{0}$ or $A' \,\|\, \mathbf{0}$. If so, one almost certainly has a "real" encryption oracle.

Security of AMac. If AHash is an AXU hash then AMac is a PRF, as AMac is constructed from the Carter-Wegman paradigm [6]. Alternatively, one can view AMac as an approximation of an AES-based PMAC [5] 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 [8, 9, 36, 37]. This gives another heuristic justification for the scaling down from full AES to AES4 in AHash.

Ciphertexts of less than one block. The claim that EncipherFF0 gives a tweakable, strong PRP over $\text{Byte}^{\leq 15}$ is 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 $2^{(k-4)n}$ plaintext/ciphertext pairs [27]. From our choice of the number of rounds, this attack needs at least 2^{72} 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 m-bit string vanish at about $2^{m/2}$ queries, and we are looking at settings with m as small as 8.

Key processing and AES4 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, we derive eleven subkeys $K_0, \ldots, K_{10}$ from the key K and steal the needed subkeys for AES4 from these. Associated to this choice, we elect to determine each subkey K_i using a more conservative (and also more parallelizable) key-scheduling algorithm than the traditional one used by AES, which gives rise to consecutive subkeys that are rather "close." (The cryptanalytic proximity of neighboring subkeys seems more likely to be problematic when the number of AES rounds is reduced from eleven to four.)

In defining AES4 subkeys, the final subkey is taken to be zero. This is provably without consequence when constructing an AXU hash function. It would seem to be fine in further AEZ contexts where

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]	$0.4m+2.4$ (3.2)	$0.4m+3$ (2.8)	$0.4m+2$ (2.8)	$0.4m+3$ (3.2)	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 12 or fewer bytes, ABYTES = 16. Other tasks: **Key setup**: $0.8m + 1.6$ (0.8). **Process AD**: $0.4m$ (0.4) (key already setup, nonce of 12 or fewer bytes). **MAC generation or verification**: 1 (1) (key already setup, AD already processed, nonce $N = \varepsilon$, its contribution precomputed).

each AES4 application except the last is followed by one that employs pre-whitening. Cascading a post-whitened permutation with a pre-whitened one seems redundant. Pleasantly, using zero as a final AES4 round key frees up the xor included in the `aesenc` instruction to do the other computational work needed for Feistel.

Our version of AES4 does not omit the final-round `MixColumns`, as AES itself was defined to do. In the context of repeated AES4 applications, omission of the final `MixColumns` likely *would* decrease security. See Dunkelman and Keller for some work in this direction [11]. And the motivation for removing the `MixColumns` step from the last round of AES is for us moot: the inverse AES cipher is never used.

The E construction is not provably-secure under the assumption that AES is a good blockcipher; the TBC construction is where the scaling-down has occurred. But one would get a provably good TBC if one dropped lines 401–403, regarded the AEZ key as a random $7 \cdot 16$ byte string $J \parallel L \parallel \boldsymbol{K} \parallel \boldsymbol{k}_0 \parallel \boldsymbol{k}_1 \parallel \boldsymbol{k}_2 \parallel \boldsymbol{k}_3$, and replaced the two AES4 calls by AES calls.

At present we view the entropy extraction procedure Extract as essentially heuristic, although some provable-security claims about it can be made from the leftover hash lemma [2, 10, 16]. The method follows the general plan of NIST recommendation SP 800-56C [7], employing entropy extraction followed by a CTR-mode expansion. For the former we produce produce 256 bits rather than 128 bits, as this seemed more appropriate to our choice of allowing such long keys.

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 m can be encrypted into strings of m + ABYTES bytes where $0 \leq$ ABYTES ≤ 16. One achieves the maximal privacy and authenticity consistent with ABYTES. The value ABYTES is authenticated and may change as often as a user likes.

Objective	**Robust-AE**, a goal that implies **MRAE** (nonce-reuse misuse resistance).
Type	**Blockcipher-based** scheme, based on **AES** and **AES4**.
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**. Subkeys are obtained using an extract-then-expand approach.
Nonce length	**Arbitrary**. May vary during a session.
Auth length	**0–16 bytes**. Expansion by 0 bytes gives a strong, tweakable, VIL blockcipher.
Nonce reuse	**Yes**. Secure against nonce-reuse in the strongest sense of the phrase [35].
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.
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.
Online	**No**. MRAE schemes can't be online (encryption or decryption).
Inverse free	**Yes**. The inverse direction of AES or AES4 is never used.
Context size	**144 bytes** (for $J, 2J, 4J, L, K_0$–K_3, Δ) or **112 bytes** (for J, L, K_0–K_3, Δ; two extra doublings/msg) or **48 bytes** (for J, L, Δ; 1.6 extra AES-equivalents/msg) or **32 bytes** (for J, L; can't exploit static AD).
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 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. ▸ Parameters are authenticated and 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 [31].

2) Computational cost is close to that of AES-CTR mode: roughly 1 AES-equivalent per block. And an implementation only needs to employ the forward direction of AES.
3) Nonces are optional (fix $N = \varepsilon$ if unused). If used, they can have any length. If unused, one gets the strongest possible security notion in their absence.
4) 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.)
5) AEZ functions well as a stand-alone MAC and as a stand-alone enciphering scheme. In the former context, it is parallelizable and uses about 0.4 AES operations per block.
6) Verification of plaintext redundancy enhances authenticity, as we have already explained.

7) 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.
8) Release of unverified plaintext does not cause any problems for AEZ. This is another part of the robust-AE notion.
9) 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.
10) An encryption implementation can make one left-to-right, constant-memory pass over the input, and then a second left-to-right, constant-memory pass over the input, this time outputting the ciphertext online. Decryption can be similarly realized. The cost increases about 40%, to an amortized 1.4 AES-equivalents per block.
11) It is possible to accelerate the rejection of invalid ciphertexts by having decryption compute the final ciphertext block M_m prior to computing the remainder of the plaintext. The cost is about 0.4 AES-equivalents per block.
12) AEZ is fully parallelizable in the processing of plaintext, ciphertext, and AD.
13) Static AD can be preprocessed so that one doesn't have to subsequently pay a per-message $|A|$-dependent cost. (Note: realizing this benefit requires an API that decouples provisioning of the AD and provisioning of other inputs.)
14) 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).
15) The context size has been kept quite small: that natural context size is 144 bytes, although an implementation can make due with as little as 32 or 48 bytes without incurring an excessive computational price.
16) No AEZ-related patents have been or will be requested.

On an Intel Haswell CPU, a preliminary implementation of AEZ enciphers 4 KByte messages at 0.75 cpb and has a peak processing rate (the marginal cost of processing an additional 256 bytes) of 0.69 cpb.

Advantages over GCM. AEZ has much stronger security properties than GCM. The later is not nonce-reuse secure, cannot safely generate short tags [13], and is not secure with respect to disclosure of unverified plaintext. GCM does not achieve the RAE security definition. AEZ avoids $\text{GF}(2^{128})$ 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 [35]. 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 [3]. 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 [4] was the obvious choice. It was already in a draft standard [12], 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).

For enciphering longer strings, there were a great many off-the-shelf alternatives we could turn to (see [34] for a list). The best-known was EME2 [14, 17]. 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.

Our EME4 solution builds on EME [14, 15] and OTR [23], but uses tweakable blockciphers [21] to arrive at an analyzable design. It makes strong use of what we have called accelerated provable-security. The scaled-down design, with a per-block amortized 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 Revision History

Below we record each public version of AEZ since its inception.

- **AEZ v1** (2014.03.15): Initial definition. Submitted to the CAESAR competition.
- **AEZ v1.1** (2014.04.29): A minor revision. Corrected typos in the document above.
- **AEZ v2** (2014.08.17): A major revision, we replaced the enciphering algorithm that had been used for adequately long strings, MEM, by a new algorithm, EME4. While no problems were ever found with MEM, the move facilitated two major gains: (a) the amortized cost was 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 was removed from AEZ. Also, EME4 was simpler, and we found ways to correspondingly simplify the entire spec.

Acknowledgments

Rogaway thanks Dustin Boswell and René Struik for their unwitting role motivating the creation of AEZ. In an April 2013 email to Mihir Bellare, Boswell wrote of wanting an easier-to-use encryption scheme, and he sketched how such a scheme might look to its user. Bellare kindly passed the note on. Struik later gave a presentation that emphasized the importance of minimizing length expansion in low-energy environments [38]. Rogaway also thanks Stefan Lucks for a Jan 2012 discussion in which Lucks advocated developing a solution to the problem of leaking unverified plaintexts.

Thanks to Terence Spies feedback during the AEZ design. Work with him on FPE and FFX helped motivate linking up FPE and AE. Thanks to Liden Mu, Chris Patton, Tom Ristenpart, and Yusi (James) Zhang for comments and corrections.

Hoang was supported by NSF grants CNS-0904380, CCF-0915675, CNS-1116800 and CNS-1228890;

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.

No animals were harmed in conducting our 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] B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. Pietrzak, F. Standaert, and Y. Yu. Leftover Hash Lemma, Revisited *CRYPTO 2011*, LNCS 6841, pp. 1–20, 2011.

[3] 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.

[4] 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.

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

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

[7] L. Chen. Recommendation for key derivation through extraction-then-expansion. NIST Special Publication 800-56C. Nov 2011.

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

[9] 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.

[10] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC Modes. *CRYPTO 2004*, LNCS 3152, Springer, pp. 494–510, 2004.

[11] 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.

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

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

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

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

[16] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. Construction of a pseudo-random generator from any one-way function. *SIAM Journal on Computing*, 28(4), pp. 1364-1396, 1999.

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

[18] 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.

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

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

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

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

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

[24] 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.

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

[26] M. Naor and O. Reingold. The NR mode of operation. Undated manuscript realizing the mechanism of [25].

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

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

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

[30] 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.

[31] 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.

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

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

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

[35] 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.

[36] 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.

[37] 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.

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