Robust Authenticated-Encryption AEZ and the Problem That It Solves

Viet Tung Hoang[1,2(✉)], Ted Krovetz[3], and Phillip Rogaway[4]

[1] Department of Computer Science, University of Maryland, College Park, USA
tvhoang@umd.edu
[2] Department of Computer Science, Georgetown University, Washington DC, USA
[3] Department of Computer Science, California State University, Sacramento, USA
[4] Department of Computer Science, University of California, Davis, USA

Abstract. With a scheme for *robust* authenticated-encryption a user can select an arbitrary value $\lambda \geq 0$ and then encrypt a plaintext of any length into a ciphertext that's λ characters longer. The scheme must provide all the privacy and authenticity possible for the requested λ. We formalize and investigate this idea, and construct a well-optimized solution, AEZ, from the AES round function. Our scheme encrypts strings at almost the same rate as OCB-AES or CTR-AES (on Haswell, AEZ has a peak speed of about 0.7 cpb). To accomplish this we employ an approach we call *prove-then-prune*: prove security and then instantiate with a *scaled-down* primitive (e.g., reducing rounds for blockcipher calls).

Keywords: AEZ · Authenticated encryption · CAESAR competition · Misuse resistance · Modes of operation · Nonce reuse · Prove-then-prune · Robust AE

1 Introduction

We expose the low cost and high benefit of building authenticated-encryption (AE) schemes that achieve the unprecedentedly strong goal we call *robust* AE (henceforth RAE). We explain why RAE is desirable, define its syntax and security, and explore its guarantees. Then we construct an RAE scheme, AEZ, from AES4 and AES10 (four- and ten-round AES). AEZ's efficiency—nearly that of AES-based OCB [32] or CTR mode—flies in the face of a community's collective work [4,11–13,22–25,35,38–40,52–54,60] in which wide-block enciphering schemes—a special case of RAE—were always far more expensive than conventional blockciphers. Achieving this efficiency has entailed using a design paradigm, the *prove-then-prune* approach, with implications beyond AE.

Ciphertext expansion. One can motivate RAE from a syntactic point of view. Recall that in a nonce-based AE scheme, a plaintext M is mapped to a ciphertext $C = \mathcal{E}_K^{N,A}(M)$ under the control of a key K, nonce N, and associated data (AD) A. Typically the *ciphertext expansion* (or *stretch*) $\lambda = |C| - |M|$ is a constant or user-selectable parameter. For conventional AE, the stretch mustn't

E. Oswald and M. Fischlin (Eds.): EUROCRYPT 2015, Part I, LNCS 9056, pp. 15–44, 2015.
DOI: 10.1007/978-3-662-46800-5_2

be too small, as customary definitions would break: a trivial adversary can get large advantage. This is because AE definitions "give up" when the first forgery occurs. The issue isn't *only* definitional: no prior AE scheme provides a desirable security guarantee when the ciphertext expansion is small.

Still, we know that meaningful security is possible even for zero-stretch: a strong pseudorandom permutation buys significant security, even from an AE point of view [5]. What is more, it would seem to be *useful* to allow small stretch, as, for example, short tags can save significant energy in resource-constrained environments (as discussed, e.g., by Struik [58]).

RAE takes a liberal approach towards ciphertext expansion, accommodating whatever stretch a user requests. This leads to schemes that deliver more than conventional AE even when the stretch is not small. Indeed we could have motivated RAE without considering small-λ, describing a desire to achieve nonce-reuse misuse-resistance [51], to automatically exploit novelty or redundancy in plaintexts [5], or to accommodate the release of unverified plaintexts [1,21]. But our ideas are most easily understood by asking what it means, and what it takes, to do well for any stretch.

Defining RAE. So consider an AE scheme that expands a plaintext $M \in \{0,1\}^*$ by a user-selectable number of bits[1] $\tau \geq 0$. We ask: what's the best privacy and authenticity guarantee possible for some arbitrary, specified τ? Robust AE formalizes an answer.

Recall the definition of a *pseudorandom-injection* (PRI) [51]: for each nonce N and associated data A, for a fixed $\tau \geq 0$, the scheme's encryption algorithm should resemble a uniformly chosen injective function $\pi_{N,A,\tau}$ from binary strings to τ-bit longer ones. Decryption of an invalid ciphertext (one lacking a preimage under π) should return an indication of invalidity.

PRIs were introduced as an alternative characterization of nonce-reuse misuse-resistant AE (henceforth MRAE). But PRIs only approximate MRAE schemes with large stretch. We recast the PRI notion as prescriptive: the user selects $\tau \geq 0$ and then the scheme must look like a PRI for the chosen value. This is our basic definition for RAE.

RAE can be thought of as a bridge connecting blockciphers and AE. When $\tau = 0$ an RAE scheme is a kind of blockcipher—a tweakable blockcipher (TBC) [34] that operates on messages and tweaks of arbitrary length and is secure as strong pseudorandom permutation (PRP). The nonce and AD comprise the tweak. When $\tau \gtrsim 128$ an RAE scheme amounts to an MRAE scheme. An RAE scheme encompasses both objects, and everything in between.

In defining RAE we are actually a bit more generous than what was sketched above, allowing an RAE's decryption algorithm to return information about an invalid ciphertext beyond a single-valued indication of invalidity. The information just needs to be harmless. To formalize this the reference experiment uses

[1] We'll later permit arbitrary alphabets. To avoid confusion, we use λ to measure ciphertext expansion in characters (bits, bytes, etc.) and τ to measure it in bits.

a simulator S to provide responses to invalid decryption queries. It must do this without benefit of the family of random injections.

Enciphering-Based AE. We can achieve RAE with *enciphering-based AE*. The idea, rooted in folklore, was formalized by Bellare and Rogaway [5] and, in a different form, by Shrimpton and Terashima [56]. In its modern incarnation, enciphering-based AE works like this:

> Take the message you want to encrypt, augment it with τ bits of redundancy, and then encipher the resulting string by applying an arbitrary-input-length tweakable blockcipher. Tweak this using the nonce, AD, and an encoding of τ. On decryption, check for the presence of the anticipated redundancy and reject a ciphertext if it is not there.

We will prove that this method achieves RAE. In fact, we'll prove that this is so even if the decryption algorithm releases candidate plaintexts with incorrect redundancy.

AEZ. We construct a highly optimized RAE scheme, AEZ. We use the same name to refer to the arbitrary-input-length tweakable blockcipher from which it's built.[2] With the increasing ubiquity of hardware AES support, we choose to base AEZ on the AES round function.

How AEZ works depends on the length of the input; see Fig. 1. To encipher a plaintext of fewer than 32 bytes we use AEZ-tiny, a balanced-Feistel scheme with a round function based on AES4, a four-round version of AES. The construction builds on FFX [6,17]. The more interesting case, AEZ-core, is used to encipher strings of 32 bytes or more. It builds on EME [22,24] and OTR [36]. Look ahead to the top-left panel of Fig. 7. There are two enciphering layers, with consecutive pairs of blocks processed together using a two-round Feistel network. The round function for this is again based on AES4. The mask injected as the middle layer is determined, for each pair of consecutive blocks, using another AES4 call.

Performance. AEZ-core is remarkably fast; as the description above implies, we need about five AES4 calls to encipher each consecutive pair of blocks, so ten AES rounds per block. Thus our performance approaches that of CTR-AES. An implementation of AEZ on Haswell using AES-NI has a peak speed of 0.72 cpb—about the same as OCB [32]. Look ahead to Fig. 8. Additionally, invalid strings can be rejected, and AD processed, in about 0.4 AES-equivalents per block, or 0.29 cpb peak (again on Haswell). Only the forward direction of AES is used, saving chip area in hardware realizations. The context size, about 128 bytes, is small, and key setup, about 1.2 AES-equivalents for a 128-bit key, is fast.

For a two-pass mode achieving MRAE, the cluster of performance characteristics described is unexpected. Part of the explanation as to how this is possible lies in the use of a design approach that benefits from both classical and provable-security design. Let us explain.

[2] Since an RAE scheme trivially determines an arbitrary-input-length tweakable blockcipher (set $\tau = 0$) it makes sense to use a single name for both objects.

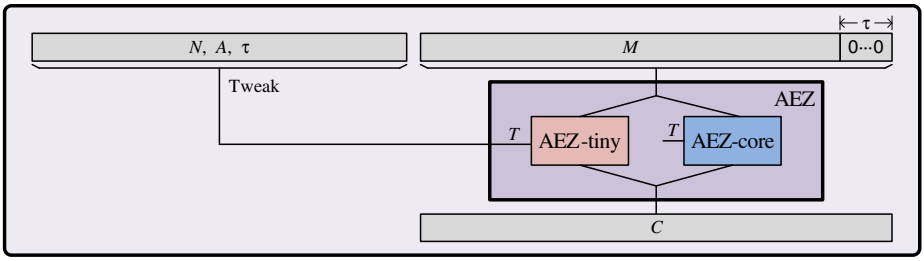


Fig. 1. High-level structure of AEZ. After appending to the message a block of τ zero bits we encipher it using a tweak T comprising the nonce N, associated data A, and stretch τ. Enciphering depends on the length of the plaintext: usually we use AEZ-core, but strings shorter than 32 bytes are enciphered by AEZ-tiny. Both depend on the underlying key K, which is not shown in the diagram above.

Prove-then-prune design. We designed AEZ using an approach we call *prove-then-prune*. It works like this:

> To achieve some complex cryptographic goal, design a scheme in the provable-security tradition, choosing an underlying primitive and demonstrably achieving the goal when it's instantiated by an object achieving some standard assumption. Then, to improve speed, selectively instantiate some of the applications of the primitive using a *scaled-down* (e.g., reduced-round) construction. Use heuristic or cryptanalytic reasons to support the expectation that, despite scaling down, the scheme remains secure.

Specifically, AEZ is designed in terms of a tweakable blockcipher (TBC). If this TBC had been instantiated in the "usual" way, say using AES and the XE construction [34,49], we would have a provably-sound design on message space $\{0,1\}^{\geq 128}$. The cost would be about 2.5 times the cost of AES. But to speed things up, we instantiate most TBC calls with an AES4-based construction. Heuristics reasons to suggest that security nonetheless remains. Our design was specifically chosen so as to make a scaled-down instantiation plausible.

The thesis underlying prove-then-prune approach is that it can be instrumental for devising highly efficient schemes for complex aims. We believe that *if* the instantiation is done judiciously, then the scaled-down scheme retains *some* assurance benefit. Still, it is important to emphasize the limitations of prove-then-prune. Naming an approach is not license to abuse it. The method is dangerous in the same sort of way that designing a confusion/diffusion primitive is: one has no guarantees for the object that will actually be used. Additionally, the set of people with provable-security competence is nearly disjoint from those with cryptanalytic competence. The authors think it essential that cryptanalysts study AEZ. This is all the more true because pruning was aggressive.

In some way, prove-then-prune is implicit in prior work: schemes like ALRED [15] typify a trend in which reduced-round AES in used in contexts where full AES would demonstrably do the job.

RAE BENEFITS. What do we hope to gain with RAE? Our definition and scheme are meant to achieve all of the following: (1) If (M, A) tuples are known *a priori* not to repeat, no nonce is needed to ensure semantic security. (2) If there's 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 stretch. (4) Because of the last two properties, one can minimize length-expansion in many 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, 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 enumerated would sometimes be worth a considerable computational price. Yet the overhead we pay is low: AEZ is almost as fast as OCB.

DISCUSSION. AEZ's name is meant to simultaneously suggest AE, AES, and EZ (easy), the last in the sense of ease of correct use. But the simplicity is for the user; we would not claim that the AEZ algorithm is simple.

Since McOE and COPA [2,20], some recent AE schemes have been advertised as nonce-reuse misuse-resistant despite being online.[3] But online schemes are never misuse-resistant in the sense originally defined [51].[4] They never support automatic exploitation of novelty or verified redundancy [5] and are always vulnerable to a simple message-recovery attack [47]. We disagree with the presumption that two-pass AE schemes are routinely problematic; in fact, our work suggests that, on capable platforms, there isn't even a performance penalty. Finally, short messages routinely dominate networking applications, and we know of no application setting where it's important to limit latency to just a few bytes, the implicit expectation for proposed online schemes.

This paper upends some well-entrenched assumptions. Before, AE-quality was always measured with respect to an aspirational goal; now we're suggesting to employ an achievable one. Before, substantial ciphertext expansion was seen as necessary for any good AE; now we're allowing an arbitrary, user-supplied input. Before, AE schemes and blockciphers were considered fundamentally different species of primitives; now we're saying that, once the definitions are strengthened, they're pretty much the same thing. Before, one could either give a provable-security design or one that follows a more heuristic tradition; now we're doing the one and yet still finding need for the other.

AEZ is one of 57 CAESAR submissions [7]. It's distinguished by being the notionally strongest submission. We expect it to help clarify the potential cost and benefit of two-pass AE.

[3] By *online* we mean that the encryption algorithm can be realized in $O(1)$ memory and a single pass over M.

[4] If the first bit of ciphertext doesn't depend on the last bit of plaintext an adversary easily wins the MRAE game.

2 Prior AE Definitions

Fix an alphabet Σ. Typically Σ is $\{0,1\}$ or $\{0,1\}^8$, but other values, like $\Sigma = \{0,1,\ldots,9\}$, are fine. For $x \in \Sigma^*$ let $|x|$ denote its length. We write ε for the empty string and $x \xleftarrow{} X$ for uniformly sampling from a distribution X. If X is a finite set, it has the uniform distribution.

Syntax. We formalize a nonce-based AE scheme as a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key space $\mathcal{K}$ is a set of strings with an associated distribution. The encryption algorithm $\mathcal{E}$ is deterministic and maps a four-tuple $(K, N, A, M) \in (\Sigma^*)^4$ to a value $C = \mathcal{E}_K^{N,A}(M)$ that is either a string in Σ^* or the distinguished symbol $\perp$. Later we will allow AD to be a vector of strings, $A \in (\Sigma^*)^*$. The distinction is insignificant insofar as we can always encode a vector of strings as a string. We require the existence of sets $\mathcal{N}$, $\mathcal{A}$ and $\mathcal{M}$ (the nonce space, AD space, and message space) such that $\mathcal{E}_K^{N,A}(M) \neq \perp$ iff $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$. The decryption algorithm $\mathcal{D}$ is deterministic and takes a four-tuple (K, N, A, C) to a value $\mathcal{D}_K^{N,A}(C) \in \Sigma^* \cup \{\perp\}$. The length of a string-valued $C = \mathcal{E}_K^{N,A}(M)$ is not allowed to depend on anything beyond $|N|$, $|A|$ and $|M|$. In fact, usually $\lambda = |C| - |M|$ is a constant, in which case we call the scheme λ-*expanding* and refer to λ as the *ciphertext expansion* or *stretch*. We require that if $C = \mathcal{E}_K^{N,A}(M)$ is a string then $\mathcal{D}_K^{N,A}(C) = M$. Algorithm $\mathcal{D}$ *rejects* ciphertext C if $\mathcal{D}_K^{N,A}(C) = \perp$ and *accepts* it otherwise.

AE and MRAE security. Both conventional-AE and MRAE security can be defined using a compact, all-in-one formulation [51]. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE-scheme. Consider an adversary $\mathcal{A}$ with access to an encryption oracle Enc and a decryption oracle Dec. We define the MRAE security of $\mathcal{A}$ as $\mathbf{Adv}^{\mathrm{mrae}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{Real}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{Ideal}_{\Pi}} \Rightarrow 1]$, the difference in the probability that $\mathcal{A}$ outputs 1 when run in the **Real** and **Ideal** games of Fig. 2. Both begin by selecting $K \xleftarrow{} \mathcal{K}$. Game **Real** answers encryption queries (N, A, M) with $\mathcal{E}_K^{N,A}(M)$ and decryption queries (N, A, C) with $\mathcal{D}_K^{N,A}(C)$. Game **Ideal** answers $\mathsf{Dec}(N, A, C)$ queries with $\perp$ and $\mathsf{Enc}(N, A, M)$ queries with $|C|$ uniformly chosen characters, where $C \leftarrow \mathcal{E}_K^{N,A}(M)$. For games **Real** and **Ideal**, adversaries may not repeat an Enc or Dec query, ask an Enc query $(N, A, M) \notin \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, ask a Dec query $(N, A, C) \notin \mathcal{N} \times \mathcal{A} \times \Sigma^*$, or ask a Dec query (N, A, C) after an Enc query of (N, A, M) returned C.

The above definition captures MRAE security because repeated nonces were allowed and were properly serviced. For the conventional AE notion, $\mathbf{Adv}^{\mathrm{ae}}_{\Pi}(\mathcal{A})$, modify **Real** and **Ideal** by having an $\mathsf{Enc}(N, A, M)$ query following an earlier $\mathsf{Enc}(N, A', M')$ query return $\perp$. This has the same effect as prohibiting repeated N-values to the Enc oracle.

PRI security. We define security in the sense of a *pseudorandom-injection* (PRI) [51]. Fix a λ-expanding AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$; for now, λ is a constant associated to a (well-behaved) AE scheme. Let $\mathbf{Adv}^{\mathrm{pri}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{Real}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{PRI}_{\Pi}} \Rightarrow 1]$ with the oracles again defined in Fig. 2. There $\mathrm{Inj}(\lambda)$ denotes

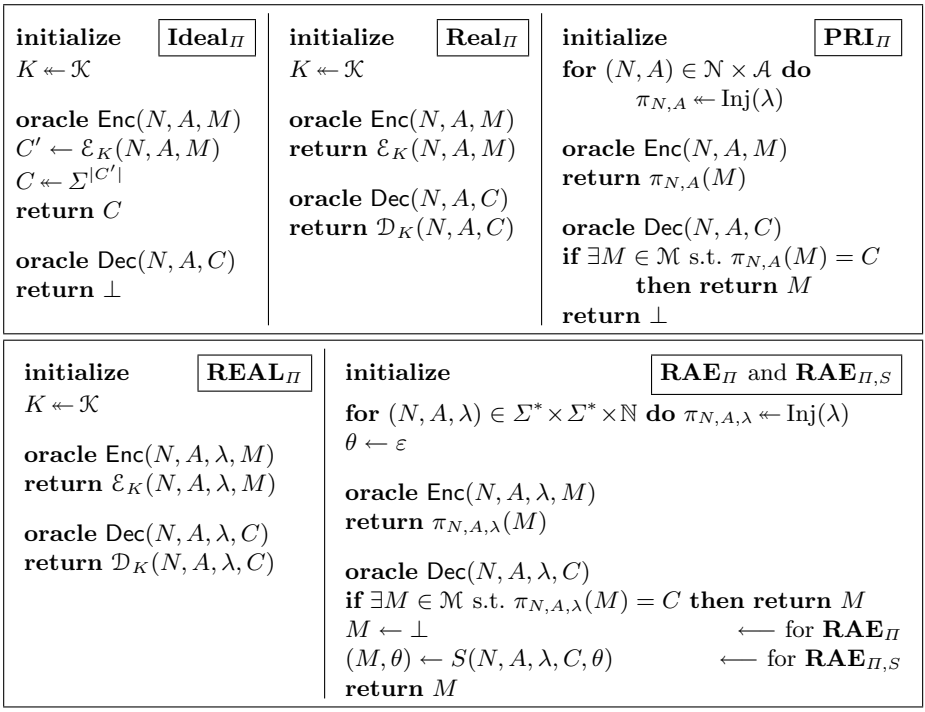


Fig. 2. Games for defining security. The top three games are the usual ones for defining the AE and MRAE notions. The bottom two games are used to define RAE.

the set of all one-to-one functions from Σ^* to Σ^* that increase the length of their inputs by λ characters. The same query restrictions apply as before.

Besides defining PRI security, Rogaway and Shrimpton showed that, for large ciphertext expansion λ, the notion essentially coincides with MRAE security [51]. Below we clarify the role of the ciphertext expansion by giving a sharper extended version of their result. To state our bound, define the *misuse count* as follows. Initially, set $r = 0$. Then, for each encryption query $\mathsf{Enc}(N, A, M)$, if there was a prior query (N, A, M') such that $|M'| = |M|$, increment r by 1. The final value of r is the misuse count. Below we show that good PRI security implies good MRAE security as long as q is small compared to $|\Sigma|^{\lambda}$ and r is small compared to $|\Sigma|^{(\lambda+m_{\min})/2}$ (with all variables defined below). The proof is in Appendix B.1.

Theorem 1. $|\mathbf{Adv}^{\mathrm{pri}}_{\Pi}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{mrae}}_{\Pi}(\mathcal{A})| \leq 2q/|\Sigma|^{\lambda} + (r^2 + r)/|\Sigma|^{\lambda+m_{\min}+1}$ for any λ-expanding AE scheme Π and adversary $\mathcal{A}$, where r is the misuse count of $\mathcal{A}$'s queries, q is the number of queries it asks, and $m_{\min}$ is the length of the shortest string in the message space.

In short, the PRI definition captures best-*possible* security of a λ-expanding AE scheme, while the MRAE formulation captures an unreachable ideal. The gap between the realizable and the ideal is formalized by Theorem 1. It is small if the ciphertext expansion is large, and it is large if the ciphertext expansion

is small. The latter is so because any *actual* encryption algorithm must map distinct (N, A, M) and (N, A, M') to distinct ciphertexts, whence real encryption *can't* return uniformly random characters. Similarly, for any infinite message space, *some* unqueried ciphertexts *must* be valid, whence a decryption oracle that *always* returns an indication of invalidity is hoping for too much. Building on the PRI notion, we will now look towards an even more precise way to capture best-possible AE security.

3 RAE Security

Syntax. The principle difference between a PRI and an RAE scheme is that, for the latter, the ciphertext expansion λ is no longer a property of a scheme: it's an arbitrary *input* from the user. All values $\lambda \in \mathbb{N}$ should be allowed.[5] Corresponding to this change, we'll write $\mathcal{E}_K^{N,A,\lambda}(M)$ and $\mathcal{D}_K^{N,A,\lambda}(C)$. The difference may look small, but its consequences are not.

Fix an alphabet Σ. Our formal definition again has an RAE scheme being a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, but with the signature of $\mathcal{E}$ and $\mathcal{D}$ updated. The encryption algorithm $\mathcal{E}$ is deterministic and maps a five-tuple $(K, N, A, \lambda, M) \in (\Sigma^*)^3 \times \mathbb{N} \times \Sigma^*$ to a string $C = \mathcal{E}_K^{N,A,\lambda}(M)$ of length $|M| + \lambda$. For maximal utility when realized, we are not permitting a return value of $\perp$: an RAE scheme must be able to encrypt any M using any N, A, and λ. The decryption algorithm $\mathcal{D}$ is deterministic and takes a five-tuple (K, N, A, λ, C) to a value $\mathcal{D}_K^{N,A,\lambda}(C) \in \Sigma^* \cup \{\perp\}$. We require that $\mathcal{D}_K^{N,A,\lambda}(\mathcal{E}_K^{N,A,\lambda}(M)) = M$ for all K, N, A, λ, M. If there's no M such that $C = \mathcal{E}_K^{N,A,\lambda}(M)$ then $\mathcal{D}_K^{N,A,\lambda}(C) = \perp$. Later in this section we will relax this requirement as a way to model the possibility of decryption algorithms that reveal information beyond an indication of invalidity.

RAE security. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an RAE scheme over alphabet Σ. Its security is defined using the games $\mathbf{REAL}_\Pi$ and $\mathbf{RAE}_\Pi$ at the bottom of Fig. 2. (For the moment, ignore $\mathbf{RAE}_{\Pi,S}$.) The adversary $\mathcal{A}$ has two oracles, an encryption oracle Enc and a decryption oracle Dec. For game **REAL**, these are realized by the actual encryption and decryption algorithms, which now take in the argument λ. For game $\mathbf{RAE}_\Pi$ we behave according to the family of random injections $\pi_{N,A,\lambda}$ chosen at the beginning of the game, responding to each encryption query (N, A, λ, M) with $C = \pi_{N,A,\lambda}(M)$ and responding to each decryption query (N, A, λ, C) with $\pi^{-1}_{N,A,\lambda}(C)$, if that inverse exists, and $\perp$ if it does not. We let $\mathbf{Adv}^{\mathrm{rae}}_\Pi(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{REAL}_\Pi} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{RAE}_\Pi} \Rightarrow 1]$. There are no restrictions on the kinds of queries the adversary may make.

To gain some appreciation for the RAE definition, consider an adversary that asks to encrypt a message M using a single byte of stretch. Such a scheme would not be considered secure in the MRAE setting, as forging with probability 1/256 is easy. But under the RAE viewpoint, that isn't a defect per se, as the user who

[5] It might be OK to set some reasonable upperbound $\lambda \leq \lambda_{\max}$, but there shouldn't be a nonzero lowerbound.

requests one-byte expansion would *expect* 1/256 of all ciphertexts to have some preimage. If a user should try to decrypt such a ciphertext C using the same K, N, A but $\lambda = 0$, a plaintext *will* emerge, never an indication of invalidity, but that plaintext should be unrelated to the originally encrypted one.

Decryption-call leakage. An AE scheme will fail to approximate the $\mathbf{RAE}_{\Pi}$ abstraction if its decryption algorithm, when presented an invalid ciphertext, routinely returns anything beyond an indication of invalidity. We now explain how to relax this expectation so that it's OK to return additional material as long as it is known to be useless.

We said earlier that, for an RAE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and any N, A, λ, C, if there's no M such that $C = \mathcal{E}_K^{N,A,\lambda}(M)$ then we expect $\mathcal{D}_K^{N,A,\lambda}(C)$ to return $\perp$. Let us relax this requirement so that $\mathcal{D}_K^{N,A,\lambda}(C)$ may instead return a string, as long as its length is not $|C| - \lambda$. Any such string is trivially recognized as invalid, so, in effect, we are having $\mathcal{D}$ return both $\perp$ and an arbitrary piece of side information Y. We are not suggesting that the "real" decryption algorithm should return anything other than $\perp$ when presented an invalid ciphertext; instead, we are effectively overloading $\mathcal{D}$ by folding into it a "leakage function" that captures that which a decryption algorithm's realization may leak about a presented ciphertext.

Using this generalized syntax, we define a game $\mathbf{RAE}_{\Pi,S}$ parameterized by a probabilistic algorithm S, the *simulator*. Again see Fig. 2. Simulator S is called upon to produce imitation ciphertexts when there's no preimage under $\pi_{N,A,\lambda}$. To accomplish this task S is provided nothing beyond the current oracle query and any saved state θ it wants to maintain. An RAE scheme is judged secure if there's a simulator S—preferably an efficient one—such that $(\mathcal{E}, \mathcal{D})$ is indistinguishable from the pair of oracles defined in $\mathbf{RAE}_{\Pi,S}$. We refine the RAE advantage by asserting that $\mathbf{Adv}^{\mathrm{rae}}_{\Pi,S}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{REAL}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{RAE}_{\Pi,S}} \Rightarrow 1]$. The "basic" RAE definition corresponds to the case where simulator S ignores its input and returns $(\perp, \varepsilon)$.

The RAE definition effectively captures that, while it may be "nice" for decryption to reveal nothing but $\perp$ on presentation of an invalid ciphertext, there are plenty of other things we could return without damaging privacy or authenticity. In fact, it really doesn't matter *what* is returned just so long as it's recognizably invalid and doesn't depend on the key.

Illustration. Fig. 3 illustrates two possibilities for how an RAE scheme might encrypt 2-bit strings with 2-bit ciphertext expansion ($\lambda = 2$). The key K, nonce N, and AD A are all fixed. For encryption, the four possible plaintexts are bijectively paired with four of the 16 possible ciphertexts. For decryption we show two possibilities. On the left is a *conventional* decryption algorithm: the 12 ciphertexts without a preimage decrypt to an indication of invalidity. One expects the simulator to always return $(\perp, \varepsilon)$. On the right is a *sloppy* decryption algorithm. The 12 ciphertexts with no preimage decrypt to 12 distinct strings, all recognizably invalid, all of the form $abcd \in \{0,1\}^4$ with $cd \neq 00$. Here the simulator S might sample without replacement from the named set of size 12.

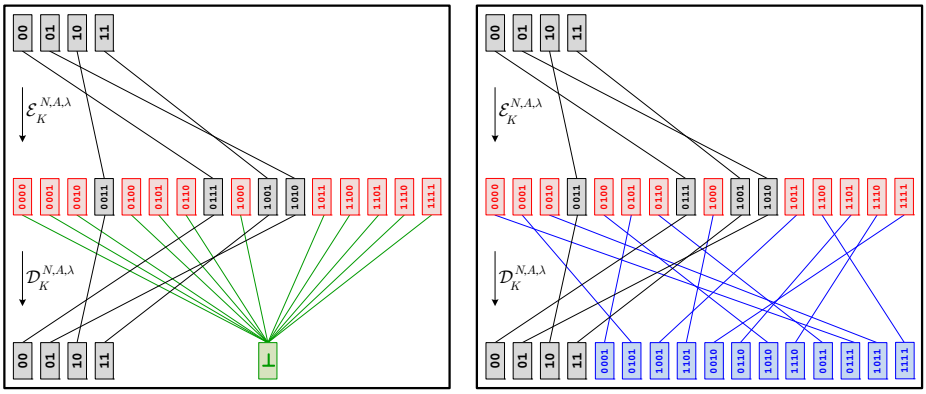


Fig. 3. Illustrating RAE. Two ways an RAE scheme might encrypt and decrypt a 2-bit string with 2-bit stretch.

Discussion. The reader may have noticed that there is no distinction in the RAE security definition between the nonce N and associated data (AD) A. For this reason, either could be dropped—say the nonce—leaving us a signature $\mathcal{E}_K^{A,\lambda}(M)$ and $\mathcal{D}_K^{A,\lambda}(C)$. There's an especially good argument for doing this when the AD A is vector-valued: the user is already free to use one of its components as a nonce. Still, for greater uniformity in treatment across AE notions, and to encourage users to provide a nonce, we have retained both N and A.

We gave our definition of RAE into two stages only for pedagogical purposes: this paper offers only one definition for RAE. The simulator S may be trivial or not; that is the only distinction.

Andreeva *et. al* [1] recently provided several security definitions also meant to capture the requirement that a decryption algorithm releases only harmless information when presented an invalid ciphertext and a repeated nonce. Our own work is radically different from theirs insofar as we provide a single definition, RAE, that rolls into it this, among many, considerations.

4 Verified Redundancy Enhances Authenticity

If a plaintext contains redundancy, one naively expects that verifying its presence upon decryption should enhance the authenticity guarantee provided. For the case of enciphering-based encryption, which provides no authenticity guarantee on its own, this has been formally supported [5,51]. But even in this case the existing results are with respect to conventional notions of AE, and such notions are too blunt to capture what one expects from verified redundancy. This is because the notions "give up" as soon as a single ciphertext forgery is made.

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be RAE scheme and let $v\colon \Sigma^* \to \{0,1\}$ be a function for indicating the "valid" strings: it determines $\mathcal{M}_v \subseteq \Sigma^*$ by $\mathcal{M}_v = \{M \in \Sigma^* : v(M) = 1\}$. Let $\Pi_v = (\mathcal{K}, \mathcal{E}, \tilde{\mathcal{D}})$ be the AE scheme built from Π that declares messages invalid if v says so: $\tilde{\mathcal{D}}_K^{N,A,\lambda}(C) = M$ if $|M| = |C| - \lambda$ and $v(M) = 1$, or

if $|M| \neq |C| - \lambda$, where $M = \mathcal{D}_K^{N,A,\lambda}(C)$, while $\tilde{\mathcal{D}}_K^{N,A,\lambda}(C) = \mathbf{0} \,\|\, M$ otherwise, with $\mathbf{0}$ a canonical point in Σ. Let $d_v = \max_{\ell \in \mathbb{N}} \left\{ (|\mathcal{M}_v \cap \Sigma^\ell|)/|\Sigma|^\ell \right\}$ be the *density* of $\mathcal{M}_v$.

Suppose, for example, that $\Sigma = \{0,1\}$ and $d_v = 1/256$: there's a byte worth of redundancy in the message space. We'd like to be able to make statements about the authenticity of Π_v such as: the chance that an adversary can forge 10 successive, distinct ciphertexts is negligibly more than 2^{-80}. Conventional AE definitions don't let one say such a thing; they stop at the bound $q/|\Sigma|^\lambda$ where q is the number of queries and λ is the ciphertext expansion (assumed here to be a constant). One would like to obtain a much sharper bound via d_v and λ—in our example, the forgery probability should be about about $q(d_v/|\Sigma|^\lambda)^{10}$. This way, even if, say, $\lambda = 0$ and $d_v = 1/2$, we are *still* able to make strong statements about the security of Π_v. Intuitively, for an RAE scheme Π, the scheme Π_v should have about $(\lambda_{\min} + \log(1/d_v))\log(|\Sigma|)$ bits of authenticity, where $\lambda_{\min}$ is the minimum ciphertext expansion of any query—even after multiple successful forgeries and even in the presence of decryption leakage, future forgeries still remain just as hard.

To capture the intuition above, in Theorem 2 we show that Π_v itself is RAE-secure. The proof is in Appendix B.2. Consequently, in game **RAE**, for any query (N, A, λ, C) with $|C| = \ell + \lambda$ to Dec, the chance that this query is a successful forgery is about $|\mathcal{M}_v \cap \Sigma^\ell|/|\Sigma|^{\ell+\lambda} \leq d_v/|\Sigma|^\lambda$, despite any decryption leakage and past successful forgeries.

Theorem 2. Let Π and Π_v be defined as above. There is an explicitly given reduction $\mathcal{R}$ with the following property. For any simulator S and any adversary $\mathcal{A}$, there is a simulator S' such that the adversary $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}^{\mathrm{rae}}_{\Pi,S}(\mathcal{B}) = \mathbf{Adv}^{\mathrm{rae}}_{\Pi_v,S'}(\mathcal{A})$. Adversary $\mathcal{B}$ makes the same queries as $\mathcal{A}$ and has essentially the same running time.

Note that for good RAE security, we want the simulator S to be efficient. This is important for privacy, but when the concern is authenticity, it's less of an issue: a computationally-unbounded simulator may give the adversary some information that it can't compute itself, but as long as the adversary can't forge, whatever the adversary learns from the simulator is irrelevant for authenticity. Still, in the proof of Theorem 2, for each query (N, A, λ, C), the simulator S' either runs S or samples from $\Sigma^\ell \cap \mathcal{M}_v$, where $\ell = |C| - \lambda$. For functions v that arise from real-world usage, sampling from $\Sigma^\ell \cap \mathcal{M}_v$ is likely to be simple and efficient, whence S' will be about as fast as S itself.

5 Robust AE from an Arbitrary-Input-Length TBC

We now show how to make an AE scheme that achieves RAE security. We begin with some basic definitions. Let $\mathcal{M} \subseteq \Sigma^*$ and $\mathcal{T}$ be sets. A *blockcipher* $\widetilde{\mathbb{E}} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ is a mapping such that $\widetilde{\mathbb{E}}_K^T(\cdot) = \widetilde{\mathbb{E}}(K, T, \cdot)$ is a length-preserving permutation on $\mathcal{M}$ for any K, T. Thus $|\widetilde{\mathbb{E}}_K^T(X)| = |X|$ and there's a unique $\widetilde{\mathbb{D}} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^* \to \mathcal{M} \cup \{\perp\}$ such that $\widetilde{\mathbb{E}}_K^T(M) = C$ implies

$\widetilde{\mathbb{D}}_K^T(C) = M$ and $\widetilde{\mathbb{D}}_K^T(C) = \perp$ when there's no M such that $\widetilde{\mathbb{E}}_K^T(M) = C$. We call $\mathcal{T}$ the *tweak space* of $\widetilde{\mathbb{E}}$. When $|\mathcal{T}| = 1$ we make the tweak implicit, writing $\mathbb{E}\colon \mathcal{K} \times \mathcal{M} \to \mathcal{M}$, now with inverse $\mathbb{D}$. We define $\mathrm{Perm}(\mathcal{M})$ as the set of all length-preserving permutations on $\mathcal{M}$, and $\mathrm{Perm}(\mathcal{T}, \mathcal{M})$ the set of all mappings $\widetilde{\pi}\colon \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ where $\widetilde{\pi}(T, \cdot) \in \mathrm{Perm}(\mathcal{M})$ for all $T \in \mathcal{T}$. We usually use *encipher* instead of *encrypt* when speaking of applying a blockcipher, and similarly for *decipher* and *decrypt*.

An *arbitrary-input-length* blockcipher is a blockcipher with message space $\mathcal{M} = \Sigma^*$. To be maximally useful, we will want a rich tweak space as well. These are versatile objects. A bit less general, a *wide-block* blockcipher has message space $\Sigma^{\geq n}$ for some fixed n. Again one prefers a rich tweak space. A *conventional* blockcipher has message space $\{0,1\}^n$ for some fixed n.

The strong, tweakable, PRP advantage of an adversary A attacking a blockcipher $\widetilde{\mathbb{E}}$ is defined as $\mathbf{Adv}_{\widetilde{\mathbb{E}}}^{\pm\widetilde{\mathrm{prp}}}(\mathcal{A}) = \Pr[K \twoheadleftarrow \mathcal{K} : \mathcal{A}^{\widetilde{\mathbb{E}}_K(\cdot,\cdot),\widetilde{\mathbb{D}}_K(\cdot,\cdot)} \Rightarrow 1] - \Pr[\widetilde{\pi} \twoheadleftarrow \mathrm{Perm}(\mathcal{T}, \mathcal{M}) : \mathcal{A}^{\widetilde{\pi}(\cdot,\cdot),\widetilde{\pi}^{-1}(\cdot,\cdot)} \Rightarrow 1]$. We'll write $\mathbf{Adv}_{\mathbb{E}}^{\pm\mathrm{prp}}(\mathcal{A}) = \Pr[K \twoheadleftarrow \mathcal{K} : \mathcal{A}^{\mathbb{E}_K(\cdot),\mathbb{D}_K(\cdot)} \Rightarrow 1] - \Pr[\pi \twoheadleftarrow \mathrm{Perm}(\mathcal{M})\colon \mathcal{A}^{\pi(\cdot),\pi^{-1}(\cdot)} \Rightarrow 1]$ if there's no tweak. If we prohibit the adversary $\mathcal{A}$ from querying the second oracle we drop the word "strong" and write $\mathbf{Adv}_{\widetilde{\mathbb{E}}}^{\widetilde{\mathrm{prp}}}(\mathcal{A})$ and $\mathbf{Adv}_{\mathbb{E}}^{\mathrm{prp}}(\mathcal{A})$ respectively.

ENCODE-THEN-ENCIPHER. Fix Σ. Let $\widetilde{\mathbb{E}}\colon \mathcal{K} \times \mathcal{T} \times \Sigma^* \to \Sigma^*$ be an arbitrary-input-length tweakable blockcipher with tweak space $\mathcal{T} = \Sigma^* \times \Sigma^* \times \mathbb{N}$. Let $\widetilde{\mathbb{D}}$ be its inverse. Let $\mathrm{Encode} : \Sigma^* \times \mathbb{N} \to \Sigma^*$ be an injective function satisfying $|\mathrm{Encode}(M, \lambda)| = |M| + \lambda$. We write the second argument to Encode as a subscript, $\mathrm{Encode}_\lambda(M) \in \Sigma^{|M|+\lambda}$. An example encoding function is $\mathrm{Encode}_\lambda(M) = M \,\|\, \mathbf{0}^\lambda$.

For any encoding function Encode there's a corresponding Decode: $\Sigma^* \times \mathbb{N} \to \Sigma^* \cup \{\perp\}$ such that $\mathrm{Decode}_\lambda(X) = M$ if there's an M satisfying $\mathrm{Encode}_\lambda(M) = X$, while $\mathrm{Decode}_\lambda(X) = \perp$ if there's no such M. We expect Encode and Decode to be trivially computable, as in the example.

From $\widetilde{\mathbb{E}}\colon \mathcal{K} \times \mathcal{T} \times \Sigma^* \to \Sigma^*$ and Encode we define the encode-then-encipher construction as the RAE scheme $\mathrm{EtE}[\mathrm{Encode}, \widetilde{\mathbb{E}}] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where

- $\mathcal{E}_K^{N,A,\lambda}(M) = \widetilde{\mathbb{E}}_K^{(N,A,\lambda)}(\mathrm{Encode}_\lambda(M))$,
- $\mathcal{D}_K^{N,A,\lambda}(C) = M$ if $\widetilde{\mathbb{D}}_K^{(N,A,\lambda)}(C) = X$ and M satisfies $\mathrm{Encode}_\lambda(M) = X$,
- $\mathcal{D}_K^{N,A,\lambda}(C) = X$ if $\widetilde{\mathbb{D}}_K^{(N,A,\lambda)}(C) = X$ and no M satisfies $\mathrm{Encode}_\lambda(M) = X$.

We stress that decryption does not simply return $\perp$ when called on an invalid (N, A, λ, C), as is conventionally done; instead, we define decryption to "leak" the entire improperly encoded string X. Nonetheless, Theorem 3 shows that $\mathrm{EtE}[\mathrm{Encode}, \widetilde{\mathbb{E}}]$ is RAE-secure when $\widetilde{\mathbb{E}}$ is secure as a strong, tweakable PRP. Its proof appears in the full version [28].

Theorem 3 (EtE is RAE-secure). Let Encode and $\widetilde{\mathbb{E}}\colon \mathcal{K} \times \mathcal{T} \times \Sigma^* \to \Sigma^*$ be defined as above. Then there's an explicitly given reduction $\mathcal{R}$ and an efficient simulator S with the following property. For any adversary $\mathcal{A}$, the adversary $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}^{\mathrm{rae}}_{\mathrm{EtE}[\mathrm{Encode},\widetilde{\mathbb{E}}],S}(\mathcal{A}) \le \mathbf{Adv}^{\pm\mathrm{prp}}_{\widetilde{\mathbb{E}}}(\mathcal{B})$. It makes at most q queries whose total length is at most that of $\mathcal{A}$'s queries plus $q\lambda_{\max}$, where q is the number of $\mathcal{A}$'s queries and $\lambda_{\max}$ is the largest stretch among them. The running time of $\mathcal{B}$ is about that of $\mathcal{A}$, plus the time associated to computations of Encode and Decode.

6 Wide-Block Enciphering: AEZ-core

Let $n \ge 1$ be an integer and let $\{0,1\}^{\ge 2n} = \{x \in \{0,1\}^* \colon |x| \ge 2n\}$. Define the *block length* of a string x as $\lceil |x|/n \rceil$. We show how to build a strong PRP on $\{0,1\}^{\ge 2n}$ from a TBC on $\{0,1\}^n$. We'll use about 2.5 TBC calls per n-bit block. Later we'll instantiate the TBC using mostly AES4, employing the prove-then-prune paradigm to selectively scale-down. This will reduce the amortized cost to about one AES call per block. Also see the full version [28] for how to tweak a wide-block blockcipher.

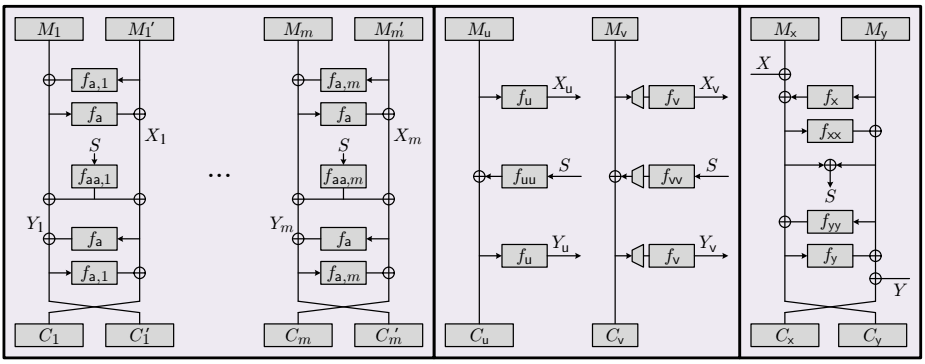
We begin by recalling the definition of a pseudorandom function (PRF) $f : \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$. For an adversary $\mathcal{A}$ attacking f, its PRF advantage is $\mathbf{Adv}^{\mathrm{prf}}_f(\mathcal{A}) = \Pr[K \xleftarrow{} \mathcal{K} \colon \mathcal{A}^{f_K(\cdot)} \Rightarrow 1] - \Pr[\rho \xleftarrow{} \mathrm{Func}(\mathcal{M}, n) \colon \mathcal{A}^{\rho(\cdot)} \Rightarrow 1]$ where $\mathrm{Func}(\mathcal{M}, n)$ is the set of all functions from $\mathcal{M}$ to $\{0,1\}^n$.

AEZ-core. Let $\mathcal{T} = \{\mathsf{a}, \mathsf{u}, \mathsf{uu}, \mathsf{v}, \mathsf{vv}, \mathsf{x}, \mathsf{xx}, \mathsf{y}, \mathsf{yy}\} \cup (\{\mathsf{a}, \mathsf{aa}\} \times \mathbb{N})$ be the tweak space. Suppose we have a PRF $f \colon \mathcal{K} \times (\mathcal{T} \times \{0,1\}^n) \to \{0,1\}^n$. One can instantiate this with a TBC $\widetilde{E}$ on $\{0,1\}^n$ by setting $f_K(K,(T,X)) = \widetilde{E}^T_K(X)$. Consider the wide-block blockcipher AEZ-core$[f]$ defined and illustrated in Fig. 6. It loosely follows EME/EME2 [22,24,29], but avoids all doubling operations and only uses the forward direction of the underlying TBC. AEZ-core$[f]$ operates on $\mathcal{M} = \{0,1\}^{\ge 2n}$ and itself takes in no tweak. Theorem 4 shows that it's a strong PRP. The proof is in the full version [28].

Theorem 4. Let $n \ge 1$ be an integer and let $\mathcal{T}$ and f be as above. There's an explicitly given reduction $\mathcal{R}$ with the following property. For any adversary $\mathcal{A}$, adversary $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}^{\pm\mathrm{prp}}_{\mathrm{AEZ\text{-}core}[f]}(\mathcal{A}) \le \mathbf{Adv}^{\mathrm{prf}}_f(\mathcal{B}) + 2\sigma^2/2^n$ where σ is the total block length of $\mathcal{A}$'s queries. Adversary $\mathcal{B}$ uses the same running time as $\mathcal{A}$, and makes at most 2.5σ queries.

Discussion. AEZ-core and its inverse are almost the same: the only change needed is to take the rightmost column of tweaks in reverse order. Given that one must have *some* asymmetry in an RAE scheme—an involution is certainly RAE-insecure—this is about as symmetric a design as one could hope for. A high degree of symmetry can help maximize efficiency of both hardware and software. Symmetry is the reason for the wire-crossing just before each $C_i\, C'_i$.

Among the efficiency characteristics of AEZ-core is that one can selectively decrypt a chosen block about 2.5 times more quickly than decrypting everything.




```
10  algorithm AEZ-core(K, M)                                   //AEZ-core
11  M_1 M_1' ··· M_m M_m' M_uv M_x M_y ← M
12      where |M_1| = ··· = |M_m'| = |M_x| = |M_y| = n and |M_uv| < 2n
13  d ← |M_uv|;  if d < n then M_u ← M_uv; M_v ← ε
14  else M_u ← M_uv[1..n]; M_v ← M_uv[n + 1..|M_uv|] fi
15  for i ← 1 to m do W_i ← M_i ⊕ f_{a,i}(M_i');  X_i ← M_i' ⊕ f_a(W_i) od
16  if d = 0 then X ← X_1 ⊕ ··· ⊕ X_m ⊕ 0
17  else if d < n then X ← X_1 ⊕ ··· ⊕ X_m ⊕ f_u(M_u10*)
18  else X ← X_1 ⊕ ··· ⊕ X_m ⊕ f_u(M_u) ⊕ f_v(M_v10*) fi
19  S_x ← M_x ⊕ X ⊕ f_x(M_y);  S_y ← M_y ⊕ f_xx(S_x);  S ← S_x ⊕ S_y
20  for i←1 to m do
21      S' ← f_{aa,i}(S); Y_i ← W_i ⊕ S'; Z_i ← X_i ⊕ S'
22      C_i' ← Y_i ⊕ f_a(Z_i); C_i ← Z_i ⊕ f_{a,i}(C_i') od
23  if d = 0 then C_u ← C_v ← ε;  Y ← Y_1 ⊕ ··· ⊕ Y_m ⊕ 0
24  else if d < n then C_u ← M_u ⊕ f_uu(S); C_v ← ε; Y ← Y_1 ⊕ ··· ⊕ Y_m ⊕ f_u(C_u10*)
25  else C_u ← M_u ⊕ f_u(S);  C_v ← M_v ⊕ f_vv(S)
26      Y ← Y_1 ⊕ ··· ⊕ Y_m ⊕ f_u(C_u) ⊕ f_v(C_v10*) fi
27  C_y ← S_x ⊕ f_yy(S_y);  C_x ← S_y ⊕ Y ⊕ f_y(C_y)
28  return C_1 C_1' ··· C_m C_m' C_u C_v C_x C_y
```

Fig. 4. The AEZ-core[f] construction. The method builds a strong-PRP on $\{0,1\}^{\geq 2n}$ from an n-bit-output PRF f that operates on its subscript and argument. It's key K is implicit. The PRF can be realized by a TBC.

When AEZ-core is turned into an RAE scheme by the EtE construction, this observation is put to good use in achieving fast rejection of ciphertexts whose final 0^τ bits is plaintext is not correct. That it is undamaging to release this timing information is guaranteed by results already show—in particular, that it is ok to release the *entire* speculative plaintext.

AEZ-core confines "specialized" processing to the final 2–4 blocks. This helps with efficiency and simplicity compared to having specialized processing at the beginning or at the beginning and end. In particular, the 0^τ authenticator used to make an RAE scheme will be put at the *end* of the message (adding a variable number of zero-bits at beginning could destroy word alignment) and, as long as $\tau \leq 2n$, it will be found in the final two blocks.

Numerous alternatives to AEZ-core were considered before arriving at our design. Correct alternatives we know are slower or more complex, while most simplifications are wrong. For example, consider trying to cheapen the design by using $c_i \cdot f_{\mathsf{aa},1}(S)$ instead of $f_{\mathsf{aa},i}(S)$ where each c_i is a public constant and the product is in $\mathrm{GF}(2^n)$. This fails for *any* choice of c_i. See Appendix C.

One variant of AEZ-core that *does* work is to eliminate the "left-hand" xor coming out of $f_{\mathsf{aa},i}$. (One then has to define X_i as the output of f_{a} instead of that output xor'ed with M'_1, and change Y_i similarly.) We have kept this xor because it's needed for symmetry.

7 Definition of AEZ

So far we have described two key elements of AEZ: the EtE construction and the AEZ-core[f] wide-block blockcipher. Now we give AEZ's complete description. First a bit of notation.

Notation. The bit length of a string X is written $|X|$. For the bitwise xor of unequal-length strings, 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 $|X| + p$. 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.

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 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 $(2n+1) \cdot X = (2n \cdot X) \oplus X$.

For $K, X \in \{0,1\}^{128}$ we write $\mathtt{aesenc}(X, K)$ for a single round of AES: `SubBytes`, `ShiftRows`, `MixColumns`, then 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 $\mathtt{aesenc}(\mathtt{aesenc}(\mathtt{aesenc}(\mathtt{aesenc}(X \oplus K_0, K_1), K_2), K_3), K_4)$. For $\boldsymbol{K}$ a list of 11 blocks, $\boldsymbol{K} = (K_0, K_1, \ldots, K_{10})$, define $\mathrm{AES10}_{\boldsymbol{K}}(X) = \mathrm{AES10}(\boldsymbol{K}, X)$ like we defined AES4 but with ten rounds of `aesenc`. We do not omit the final-round `MixColumns`.

AEZ definition. See Figs. 5 and 6 for the definition of AEZ, and Fig. 7 for an illustration. Most of it is self-explanatory. We briefly explain some of the algorithm's more unusual elements.

AEZ operates on arbitrary byte strings. Not only is the plaintext $M \in \mathrm{Byte}^*$ arbitrary, but so too the key $Key \in \mathrm{Byte}^*$ and nonce $N \in \mathrm{Byte}^*$. The AD is even more general: an arbitrary-length vector of byte strings, $A \in (\mathrm{Byte}^*)^*$. The requested ciphertext expansion of $\lambda \in \mathbb{N}$ bytes is measured in $\tau = 8\lambda$ bits.

100 **algorithm** ENCRYPT(K, N, A, τ, M) //AEZ authenticated encryption
101 $X \leftarrow M \,\|\, 0^\tau$; $(A_1, \ldots, A_m) \leftarrow A$
102 $T \leftarrow ([\tau]_{128}, N, A_1, \ldots, A_m)$
103 **if** $M = \varepsilon$ **then return** AEZ-prf(K, T, τ) **else return** Encipher(K, T, X)

110 **algorithm** DECRYPT(K, N, A, τ, C) //AEZ authenticated decryption
111 $(A_1, \ldots, A_m) \leftarrow A$; $T \leftarrow ([\tau]_{128}, N, A_1, \ldots, A_m)$
112 **if** $|C| < \tau$ **then return** $\perp$
113 **if** $|C| = \tau$ **then if** $C =$ AEZ-prf(K, T, τ) **then return** ε **else return** $\perp$ **fi fi**
114 $X \leftarrow$ Decipher(K, T, C); $M \,\|\, Z \leftarrow X$ where $|Z| = \tau$
115 **if** $(Z = 0^\tau)$ **then return** M **else return** $\perp$

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

210 **algorithm** Encipher-AEZ-tiny(K, T, M) //AEZ-tiny enciphering
211 $m \leftarrow |M|$; $n \leftarrow m/2$; $\Delta \leftarrow$ AEZ-hash(K, T)
212 **if** $m = 8$ **then** $k \leftarrow 24$ **else if** $m = 16$ **then** $k \leftarrow 16$
213 **else if** $m < 128$ **then** $k \leftarrow 10$ **else** $k \leftarrow 8$ **fi**
214 $L \leftarrow M(1\,..\,n)$; $R \leftarrow M(n+1\,..\,m)$; **if** $m \geq 128$ **then** $j \leftarrow 6$ **else** $j \leftarrow 7$ **fi**
215 **for** $i \leftarrow 0$ **to** $k-1$ **do**
216 $R' \leftarrow L \oplus ((\mathrm{E}_K^{0,j}(\Delta \oplus R10^* \oplus [i]_{128}))(1\,..\,n))$; $L \leftarrow R$; $R \leftarrow R'$ **od**
217 $C \leftarrow R \,\|\, L$; **if** $m < 128$ **then** $C \leftarrow C \oplus (\mathrm{E}_K^{0,3}(\Delta \oplus (C0^* \vee 10^*)) \wedge 10^*)$ **fi**
218 **return** C

220 **algorithm** Encipher-AEZ-core(K, T, M) //AEZ-core enciphering
221 $M_1 M_1' \cdots M_m M_m'\ M_{\mathsf{uv}}\ M_{\mathsf{x}} M_{\mathsf{y}} \leftarrow M$
222 where $|M_1| = \cdots = |M_m'| = |M_{\mathsf{x}}| = |M_{\mathsf{y}}| = 128$ and $|M_{\mathsf{uv}}| < 256$
223 $\Delta \leftarrow$ AEZ-hash(K, T); $d \leftarrow |M_{\mathsf{uv}}|$
224 **if** $d \leq 127$ **then** $M_{\mathsf{u}} \leftarrow M_{\mathsf{uv}}$; $M_{\mathsf{v}} \leftarrow \varepsilon$
225 **else** $M_{\mathsf{u}} \leftarrow M_{\mathsf{uv}}[1..128]$; $M_{\mathsf{v}} \leftarrow M_{\mathsf{uv}}[129..|M_{\mathsf{uv}}|]$ **fi**
226 **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**
227 **if** $d = 0$ **then** $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathbf{0}$
228 **else if** $d \leq 127$ **then** $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathrm{E}_K^{0,4}(M_{\mathsf{u}}10^*)$
229 **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**
230 $S_{\mathsf{x}} \leftarrow M_{\mathsf{x}} \oplus \Delta \oplus X \oplus \mathrm{E}_K^{0,1}(M_{\mathsf{y}})$; $S_{\mathsf{y}} \leftarrow M_{\mathsf{y}} \oplus \mathrm{E}_K^{-1,1}(S_{\mathsf{x}})$; $S \leftarrow S_{\mathsf{x}} \oplus S_{\mathsf{y}}$
231 **for** $i \leftarrow 1$ **to** m **do**
232 $S' \leftarrow \mathrm{E}_K^{2,i}(S)$; $Y_i \leftarrow W_i \oplus S'$; $Z_i \leftarrow X_i \oplus S'$
233 $C_i' \leftarrow Y_i \oplus \mathrm{E}_K^{0,0}(Z_i)$; $C_i \leftarrow Z_i \oplus \mathrm{E}_K^{1,i}(C_i')$ **od**
234 **if** $d = 0$ **then** $C_{\mathsf{u}} \leftarrow C_{\mathsf{v}} \leftarrow \varepsilon$; $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathbf{0}$
235 **else if** $d \leq 127$ **then**
236 $C_{\mathsf{u}} \leftarrow M_{\mathsf{u}} \oplus \mathrm{E}_K^{-1,4}(S)$; $C_{\mathsf{v}} \leftarrow \varepsilon$; $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathrm{E}_K^{0,4}(C_{\mathsf{u}}10^*)$
237 **else** $C_{\mathsf{u}} \leftarrow M_{\mathsf{u}} \oplus \mathrm{E}_K^{-1,4}(S)$; $C_{\mathsf{v}} \leftarrow M_{\mathsf{v}} \oplus \mathrm{E}_K^{-1,5}(S)$
238 $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathrm{E}_K^{0,4}(C_{\mathsf{u}}) \oplus \mathrm{E}_K^{0,5}(C_{\mathsf{v}}10^*)$ **fi**
239 $C_{\mathsf{y}} \leftarrow S_{\mathsf{x}} \oplus \mathrm{E}_K^{-1,2}(S_{\mathsf{y}})$; $C_{\mathsf{x}} \leftarrow S_{\mathsf{y}} \oplus \Delta \oplus Y \oplus \mathrm{E}_K^{0,2}(C_{\mathsf{y}})$
240 **return** $C_1 C_1' \cdots C_m C_m'\ C_{\mathsf{u}} C_{\mathsf{v}}\ C_{\mathsf{x}} C_{\mathsf{y}}$

Fig. 5. Main routines of AEZ. The tweakable blockcipher E, the hash AEZ-hash, and the PRF AEZ-prf are defined in Fig. 6. The ciphertext expansion is $\tau = 8\lambda$ bits.

300 **algorithm** AEZ-hash(K, T) //AXU hash. T is a vector of strings
301 $(T_1, \ldots, T_t) \leftarrow T$
302 **for** $i \leftarrow 1$ **to** t **do**
303 $m \leftarrow \max(1, \lceil |T_i|/128 \rceil)$; $X_1 \cdots X_m \leftarrow T_i$ $//|X_1| = \cdots = |X_{m-1}| = 128$
304 **if** $|X_m| = 128$ **then** $\Delta_i \leftarrow \mathrm{E}_K^{2+i,1}(X_1) \oplus \cdots \oplus \mathrm{E}_K^{2+i,m}(X_m)$
305 **if** $|X_m| < 128$ **then**
306 $\Delta_i \leftarrow \mathrm{E}_K^{2+i,1}(X_1) \oplus \cdots \oplus \mathrm{E}_K^{2+i,m-1}(X_{m-1}) \oplus \mathrm{E}_K^{2+i,0}(X_m 10^*)$
307 **return** $\Delta_1 \oplus \cdots \oplus \Delta_t \oplus \mathbf{0}$

310 **algorithm** AEZ-prf(K, T, τ) //PRF used when $M = \varepsilon$
311 $\Delta \leftarrow$ AEZ-hash(K, T)
312 **return** $(\mathrm{E}_K^{-1,3}(\Delta) \parallel \mathrm{E}_K^{-1,3}(\Delta \oplus [1]_{128}) \parallel \mathrm{E}_K^{-1,3}(\Delta \oplus [2]_{128}) \parallel \cdots)[1..\tau]$

400 **algorithm** $\mathrm{E}_K^{i,j}(X)$ //Scaled-down TBC
401 $I \parallel J \parallel L \leftarrow \mathrm{Extract}(K)$ where $|I| = |J| = |L| = 128$
402 $\boldsymbol{k}_0 \leftarrow (\mathbf{0}, I, J, L, \mathbf{0})$; $\boldsymbol{k}_1 \leftarrow (\mathbf{0}, J, L, I, \mathbf{0})$; $\boldsymbol{k}_2 \leftarrow (\mathbf{0}, L, I, J, I)$
403 $\boldsymbol{K} \leftarrow (\mathbf{0}, I, L, J, I, L, J, I, L, J, I)$
404 **if** $i = -1$ **and** $0 \le j \le 7$ **then return** $\mathrm{AES10}_{\boldsymbol{K}}(X \oplus jJ)$
405 **if** $i = 0$ **and** $0 \le j \le 7$ **then return** $\mathrm{AES4}_{\boldsymbol{k}_0}(X \oplus jJ)$
406 **if** $1 \le i \le 2$ **and** $j \ge 1$ **then return** $\mathrm{AES4}_{\boldsymbol{k}_i}(X \oplus (j \bmod 8)J \oplus 2^{\lfloor (j-1)/8 \rfloor} L)$
407 **if** $i \ge 3$ **and** $j \ge 1$ **then**
408 **return** $\mathrm{AES4}_{\boldsymbol{k}_0}(X \oplus (j \bmod 8)J \oplus 2^{\lfloor (j-1)/8 \rfloor} \cdot L \oplus (i-2)8J)$
409 **if** $i \ge 3$ **and** $j = 0$ **then return** $\mathrm{AES4}_{\boldsymbol{k}_0}(X \oplus (i-2)8J)$

Fig. 6. AEZ's hash, PRF, and TBC. The last is the locus of prove-then-prune scaling-down. The key K is turned into 384 bits by a key-derivation function Extract.

At line 217, Encipher-AEZ-tiny may xor a bit into the ciphertext just before the algorithm's conclusion. This is done to avoid a simple random-permutation distinguishing attacks, for very short strings, based on the fact that Feistel networks only generate *even* permutations [30]. A similar trick, conditionally swapping two fixed points, has been used before [45]. Our approach has the benefit that the natural implementation is constant-time.

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. Still, the additional pseudocode is easy to describe. AEZ-tiny deciphering is identical to AEZ-tiny enciphering except we must count backwards instead of forwards, and must do the even-cycles correction (line 217) at the beginning instead of the end. Specifically, Decipher-AEZ-tiny(K, T, M) is identical to Encipher-AEZ-tiny(K, T, M) except that line 215 is changed to count from $k - 1$ down to 0, while line 217 has each C replaced by M before moving the line up to just after line 213. And AEZ-core deciphering is identical to AEZ-core enciphering except that we must take the xy-tweaks in reverse order. Specifically, Decipher-AEZ-core(K, T, M) is identical to Encipher-AEZ-core(K, T, M) except we swap tweaks $(0, 1)$ and $(0, 2)$, and we swap tweaks $(-1, 1)$ and $(-1, 2)$. These appear at lines 230 and 239.

The TBC $\mathrm{E}_K^{i,j}(X)$ takes a tweak $(i, j) \in \{-1, 0\} \times [0..7] \cup \{1, 2, 3\} \times \mathbb{N}$. The first component selects between AES10 (when $i = -1$) and AES4 (when $i \ge 0$).

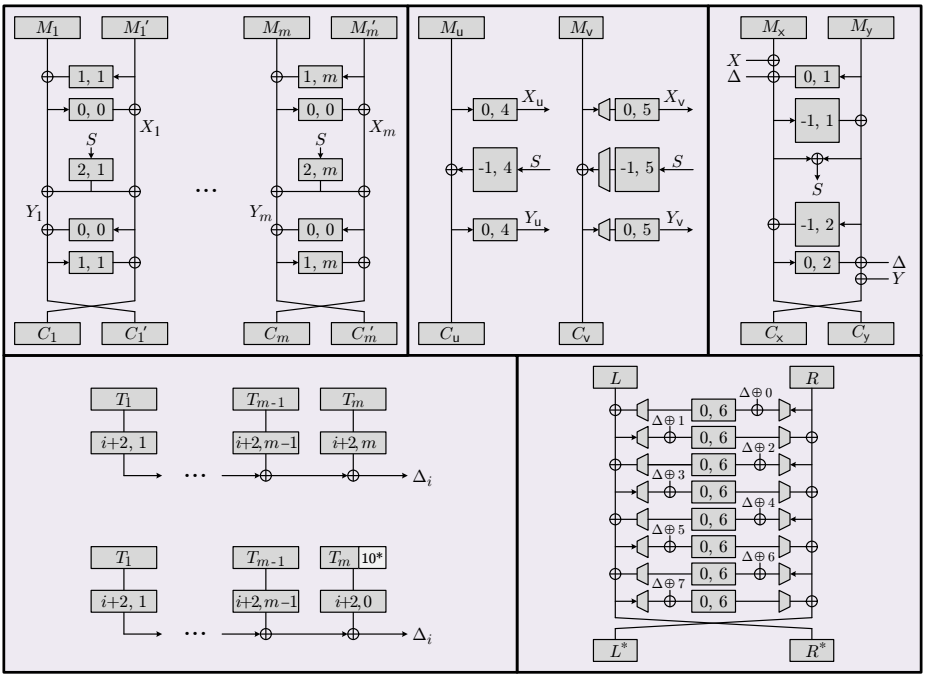


Fig. 7. Illustrating AEZ enciphering. Rectangles with pairs of numbers are TBCs, the pair being the tweak (the key, always K, is not shown). **Top row:** enciphering a message M of (32 or more bytes) with AEZ-core. The diagram shows processing a string that is (exclude the middle panel) or isn't (include the middle panel) a multiple of 16 bytes. **Bottom left:** AEZ-hash is an xor-universal hash built from AES4. It computes $\Delta = \bigoplus \Delta_i$ from a vector-valued tweak T comprising A, N, and τ. Its i-th component $T_1 \cdots T_m$ is hashed as shown. **Bottom right:** AEZ-tiny, when operating on a string $M = L \,\|\, R$ of 16–31 bytes. More rounds are used if M has 1–15 bytes.

Either way, the construction is based on XE [34,49]. We emphasize that E is *not* secure as a tweakable-PRP, since AES4 itself is completely insecure as a PRP: it is easily broken by the "Square" attack [14]. Use of an AES4-based TBC *despite* this fact is where the scaling-down has been done in AEZ.

The key $K \in \text{BYTE}^*$ is mapped to three 16-byte subkeys (I, J, L) using the key-derivation function (KDF) named Extract that is called at line 401. The definition of Extract is omitted from the figures and regarded as orthogonal to the rest of AEZ. See the AEZ spec [26] for the current Extract: $\text{BYTE}^* \to \text{BYTE}^{48}$. In our view, it is an unresolved matter what the security properties (and even what signature) of a good KDF should be. Work has gone off in very different directions [33,46,61], and the area is currently the subject of a Password Hashing Competition (PHC) running concurrently with CAESAR.

Note the mod 8's at lines 406 and 408. Unlike the offset sequence used for OCB [32], we limit ourselves to eight successive J values; after that, we add in the next power-of-two times L. This allows a small table of $2^j \cdot J$ values to be

precomputed and used regardless of the length of the message. In this way we limit the frequency of doublings yet avoid number-of-trailing-zeros calculations.

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. This usage limit stems from the existence of birthday attacks on AEZ, as well as the use of AES4 to create a universal hash function.

Cost accounting. Let us summarize the computational cost of AEZ in "AES-equivalents," where 1 AES-equivalent is 10 AES rounds. Assume a message of m blocks, the last of which may be fragmentary. To encipher or decipher $m \geq 2$ blocks takes at most $m+2.4$ AES-equivalents (latency 3.6). This assumes K, N, τ, and A have already been processed. To encrypt or decrypt $m \geq 2$ blocks: at most $m+3.8$ AES-equivalents (latency 3.6). This assumes that K, A, and τ have already been processed and that $|N| \leq 128$ and $\tau = 128$. To reject an invalid ciphertext of $m \geq 2$ blocks: at most $0.4m + 2.4$ AES-equivalents (latency 2.8). Same assumptions. To setup an m block key: $1.2m$ AES-equivalents (latency 0.4). This assumes that needed constants have been precomputed. To setup a string-values AD: $0.4m$ (latency 0.4). To encipher or decipher messages of 1–15 bytes is somewhat slower: 10, 6.8, and 4.4 AES-equivalents for 1, 2, and 3 bytes.

Parameterized counterparts. For a TBC-parameterized generalization of AEZ, let AEZ[$\widetilde{E}$] be identical to AEZ except for using the TBC $\widetilde{E}\colon \mathcal{K} \times \mathcal{T}_{\text{aez}} \times \{0,1\}^{128} \to \{0,1\}^{128}$ in place of E (assume the correct tweak-space $\mathcal{T}_{\text{aez}}$). The key space of $\widetilde{E}$ is then taken as the key space for the constructed RAE scheme. Note that AEZ = AEZ[E] with E the algorithm defined by lines 400–409.

Taking the above a step further, given a conventional blockcipher $E\colon \mathcal{K} \times \{0,1\}^{128} \to \{0,1\}^{128}$ we can define AEZ[E] as AEZ[$\widetilde{E}$] where $\widetilde{E}_K^{i,j}(X) = E_K(X \oplus (i+1)I \oplus jJ)$ for $I = E_K(\mathbf{0})$ and $J = E_K(\mathbf{1})$. The scheme AEZ[AES] can be regarded as a natural "scaled up" version of AEZ. We emphasize that AEZ is *not* AEZ[AES], which is about 2.5 times as slow.

Schemes AEZ[$\widetilde{E}$] and AEZ[E] are close to AEZ, but enjoy conventional provable-security guarantees, as we now describe.

8 Security of AEZ[$\widetilde{E}$] and AEZ[E]

We show that if $\widetilde{E}$ is secure as a tweakable PRP then AEZ[$\widetilde{E}$] is RAE-secure. In fact, the statement holds even if the decryption algorithm is modified so as to leak the entire improperly encoded string obtained by deciphering an invalid ciphertext. So, for the remainder of this section, assume the modification of AEZ in which the **else** clause of line 115 returns the deciphered message X rather than $\perp$. This change only makes our results stronger, explicitly modeling the *possibility* of a decryption implementation leaking some or all of X. The *actual* decryption algorithm returns $\perp$.

Our provable-security results for AEZ need to assume that the adversary avoids enciphering or deciphering extremely short strings—at least those under

16 bytes, say, for which AEZ-tiny, a Feistel-based construction, will not enjoy a desirable bound. While provably-secure options are now available for enciphering very short strings, they still do not have competitive efficiency.

As the alphabet for AEZ is $\Sigma = \text{BYTE}$, in this this section we write $|x|$ for the byte length of x. For an encryption query (N, A, λ, M), define the number of blocks processed as $\lceil |N|/16 \rceil + \sum_i \lceil |A_i|/16 \rceil + \lceil (|M| + \lambda)/16 \rceil$. This query is *small* if $M \neq \varepsilon$ and $16 \leq |M| + \lambda < 32$, and *tiny* if $M \neq \varepsilon$ and $|M| + \lambda < 16$. Likewise, for a decryption query (N, A, λ, C), the number of blocks processed is $\lceil |N|/16 \rceil + \sum_i \lceil |A_i| \rceil + \lceil (|C|)/16 \rceil$. The query is *small* if $16 \leq |C| < 32$ and $|C| \neq \lambda$, and *tiny* if $|C| \neq \lambda$ and $|C| < 16$. The proof for the following is in the full version [28].

Theorem 5. Let $\widetilde{E} : \mathcal{K} \times \mathcal{T}_{\text{aez}} \times \{0,1\}^{128} \to \{0,1\}^{128}$ be a TBC and $\Pi = \text{AEZ}[\widetilde{E}]$. Then there are efficient, explicitly given algorithms $\mathcal{R}$ and S with the following property. Let $\mathcal{A}$ be an adversary for attacking Π. Assume it never asks any small or tiny query. Then $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}^{\text{rae}}_{\Pi,S}(\mathcal{A}) \leq 3.5s^2/2^{128} + \mathbf{Adv}^{\widetilde{\text{prp}}}_{\widetilde{E}}(\mathcal{B})$, where s is the total number of processed blocks, plus 2 blocks per message. Adversary $\mathcal{B}$ makes at most $2.5s$ queries and has about the same running time as $\mathcal{A}$.

An alternative approach to justifying the security of AEZ is to speak of the security of AEZ[E], the cousin of AEZ defined from a conventional blockcipher E using the XE construction to make the needed TBC. Its security can be captured by the following result. The proof is in the full version [28].

Theorem 6. Let $E : \mathcal{K} \times \{0,1\}^{128} \to \{0,1\}^{128}$ be a blockcipher and $\Pi = \text{AEZ}[E]$. Then there are efficient, explicitly given algorithms $\mathcal{R}$ and S with the following property. Let $\mathcal{A}$ be an adversary for attacking Π. Assume it never asks a small or tiny query. Then $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}^{\text{rae}}_{\Pi,S}(\mathcal{A}) \leq 13s^2/2^{128} + \mathbf{Adv}^{\text{prp}}_{E}(\mathcal{B})$, where s is the total number of processed blocks, plus 2 blocks per message. Adversary $\mathcal{B}$ makes at most $2.5s$ queries and has about the same running time as $\mathcal{A}$.

If one wants to accommodate small queries then we still have a provable, albeit much inferior result. Let Feistel$[r, n]$ denote an ideal r-round Feistel network on $\{0,1\}^{2n}$. The best known provable bound for Feistel networks [43, Theorem 7] states that if an adversary makes $q \leq \frac{2^n}{128n}$ queries then $\mathbf{Adv}^{\pm\text{prp}}_{\text{Feistel}[6,n]}(\mathcal{A}) \leq \frac{8q}{2^n} + \frac{q^2}{2^{2n+1}}$. Translating this to our setting, one is bound to make at most $q \leq \frac{2^{64}}{128 \cdot 64} = 2^{51}$ small queries, and the security advantage is $q/2^{61} + 4s^2/2^{128}$. These restrictions seem to be more of the artifacts of the analysis in [43, Theorem 7] than reflecting the actual security of Feistel networks: assuming that the round functions of Feistel$[6, n]$ are instantiated from full AES, the fastest known attack, for $n \geq 64$, is still the exhaustive key search on AES.

9 Estimated Security of AEZ Itself

Consider enciphering a message M, $|M| \geq 256$, by AEZ[AES] (which, recall, is not AEZ, but a scaled-up version using an AES-based TBC). The design would seem excessive: each block M_i would be subjected to 30 rounds of AES (ten shared with a neighboring block), not counting the additional AES rounds to produce the highly unpredictable, M-dependent value S, a value derived from which gets injected into the process while 20 rounds yet remain. It is in light of such apparent overkill that AEZ selectively prunes some of the AES calls that AEZ[AES] would perform. In particular, we prune invocations where we aim 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 are the TBCs used to mask u- and v-blocks.

To estimate the security of AEZ it seems appropriate to replace the $s^2/2^{128}$ term of Theorem 5 by $s^2/2^{113}$, resulting in the bound $4s^2/2^{113} + t/2^{128}$, because of the higher maximal expected differential probability of AES4 [31] compared to an ideal hash or cipher, where t is the time (including the description size) in which the adversary runs.

Moreover, we contend that the assumption that the adversary avoids asking tiny or small queries can be lifted. To justify this heuristically, 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, due to Patarin [41], that distinguishes them from a family of independent, truly random even permutations requires at least $2^{(k-4)n}$ plaintext/ciphertext pairs. From our choice of the number of rounds, this attack needs at least 2^{72} plaintext/ciphertext pairs, and thus doesn't violate our up-to-the-birthday-bound security goal.

AEZ was specifically designed so that scaling-down most of its AES calls would seem safe. This is design-specific; one cannot indiscriminately scale a scheme's primitives. A previous design, where AEZ-core followed the NR approach [39,40], could not be as effectively scaled-down.

10 Software Performance

The development of AEZ has generally presumed an instruction set architecture (ISA) with round-level support for AES, such as Intel's AES-NI or ARM's version 8 ISA. On these systems the AES unit can be kept busy processing several AES4 computations in parallel while idle processing units handle load, store, and xor overhead. On Intel's Haswell architecture, for example, unrelated AES rounds can issue every cycle and take seven cycles to retire, so seven parallel AES4 calculations can complete in 34 CPU cycles, while idle superscalar processing units can handle other computations. This observation has led us to design AEZ to conveniently process eight blocks at a time.

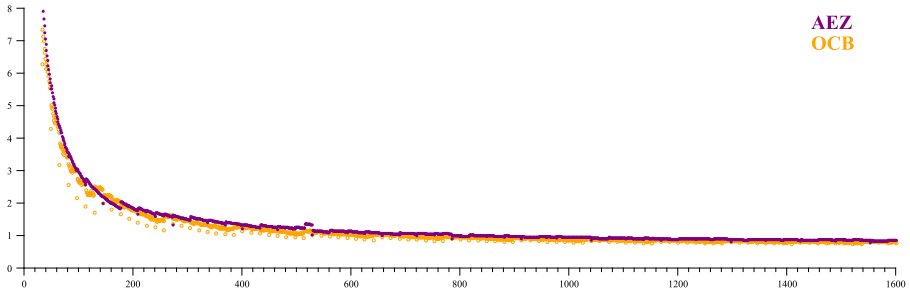


Fig. 8. AEZ vs. OCB performance. The x-axis is message length, in bytes, and the y-axis is cycles per byte (cpb). The graph is best viewed in color: solid purple circles are for AEZ; unfilled yellow circles are for OCB3 [32]. Performance of the two is close, both having peak speeds around 0.7 cpb and being similar on most shorter messages as well. The execution vehicle is an Intel Haswell processor using AES-NI.

AEZ overhead beyond AES rounds has been minimized. As an example of this, our AES4 key schedule omits the final round key, allowing `aesenc`'s included xor operation to be used for other purposes. Such optimizations lead to AEZ peak speeds, on Haswell, of around 0.72 cpb—not far from the theoretical maximum for the number of rounds executed of 0.63 cpb.

Fig. 8 compares the performance of AEZ and OCB on messages of all byte lengths up to 1600 bytes. The two are not only similar for long messages but for short strings too. Only when messages are shorter than 16 bytes, where AEZ-tiny increases the number of AES4 calls used, does OCB become significantly faster.

The performance of AEZ is on par with OCB even on processors that are not superscalar or do not support AES rounds at the assembly level. On a Marvell 88F6283 embedded CPU—a single-issue, 32-bit, ARM version 5 ISA—we see an experimental version of AEZ peaking at 86 cpb while OCB's optimized reference code runs at 84 cpb. For comparison, GCM, CCM and CTR run at 124, 134 and 67 cpb, respectively. The figures use the OpenSSL libraries.

One might expect the two-pass nature of AEZ to be a performance burden because data must be dragged into cache from memory twice. We have found that modern processors, like Intel's Haswell, have such efficient hardware prefetching that bringing data into cache twice, in a sequential streaming fashion, is not expensive at all. It requires no explicit prefetching. Encrypting 1MB on Haswell is as efficient as encrypting 32KB despite 1MB exceeding the 256KB level-2 cache. Two passes may have a more significant cost on systems with poor prefetching facilities, although this might be mitigated by software prefetching.

Another benefit of AEZ's two passes is that the second pass is not needed to discover that a ciphertext is inauthentic, leading to message rejection costing as little as 0.28 cpb on Haswell. On long messages, approximately 2/5 of AES4 calls are performed during the first pass, which aligns perfectly with the peak times we've observed for encryption and fast-rejection.

All Haswell timings reported in this paper were gathered on a 2.9 GHz Intel Core i5-4570S CPU using its time-stamp counter to gather elapsed CPU cycles

over encryption calls. Our implementation is written in C using "intrinsic" functions to access CPU-specific functionality. It was compiled using GCC 4.9 with options `-march=native -O3`. Our optimized implementation will be made publicly available and freely licensed.

Acknowledgments. Many thanks to Tom Shrimpton, who provided important interaction on RAE definitions and their implications. Liden Mu and Chris Patton proofread our specification document and did implementations that helped verify our own. We received good comments and corrections Danilo Gligoroski, Tom Ristenpart, and Yusi (James) Zhang. Thanks to Dustin Boswell for an April 2013 email on the importance of making AE easier to use, Stefan Lucks for a Jan 2012 discussion on the problem unverified plaintexts, and René Struik for an August 2013 DIAC presentation on the utility of minimizing ciphertext expansion. Thanks to Terence Spies for catalyzing the idea of unifying AE and blockciphers both in definition and schemes.

Part of this work was done when Tung was a postdoc at UC San Diego and Phil was visiting ETH Zürich. Many thanks to Mihir Bellare for that postdoc, and many thanks to Ueli Maurer for hosting that sabbatical.

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.

A More on Related Work

RAE and AEZ build on a large body of related work. While we have summarized much of this throughout this paper, here we give some additional context and high points.

Blockciphers accommodating truly arbitrary inputs were first realized by Schroeppel's Hasty Pudding Cipher (HPC) [55]. Ahead of its time, the work not only built a blockcipher on all of $\{0,1\}^*$, but also provided it a tweak. If one were to first overcome the problem that HPC's tweak is limited in length, it could be used with the EtE construction to make an RAE scheme.

The problem of constructing from conventional blockciphers those with arbitrary or near-arbitrary domains was first identified Bellare and Rogaway [4], who wanted to construct these objects with a conventional-looking mode. But the mechanism they suggested was somewhat slow, was limited to a domain of $(\{0,1\}^n)^+$, and only achieves conventional (not strong) PRP security.

In a follow-up paper [5] the same authors evidenced the utility of arbitrary-input-length blockciphers by explaining how semantic security could be achieved by enciphering messages with novelty, and they showed how authenticity could be achieved by enciphering messages with redundancy (this time using a *strong* PRP). These observations formed the basis for our work.

Around the same time as the last two work, Naor and Reingold (NR) constructed a blockcipher on $(\{0,1\}^n)^+$ by sandwiching a layer of ECB between layers of a "blockwise-universal" hashing [39,40]. The approach came to be used in many proposals, including XCB [35], which was standardized in the IEEE [29].

The other method inspiring further wide-block blockciphers was EME [24], which involves two layers of blockcipher-based enciphering and a light layer of mixing in between. A follow-on design, EME2 [22], become the other wide-block blockcipher of IEEE 1619.2 [29], Both it and XCB are tweakable and operate on a message space of $\{0,1\}^{\geq n}$. EME/EME2 provides the starting point for AEZ-core.

As for extending blockciphers to short blocks, a different line of work was begun [9]. *Format-preserving encryption* aimed to deal not only with small domains but also those defined as arbitrary finite sets, sets of numbers $[0..N-1]$, or strings over arbitrary alphabets. Adapting Feistel designs to arbitrary alphabets, realizations of FFX [6], now a draft NIST standard [17], would form the basis of AEZ-tiny.

Meanwhile, notions of AE were appearing. Probabilistic versions came first [5, 27], then a nonce-based version [50], then AD finally appeared [49]. Next the MRAE goal—RAE's closest definition counterpart—was defined [51]. The main motivation for that work was to minimize the damage that could be done by nonce-reuse.

Other authors had the same concern but weren't willing to use two-pass schemes. Fleischmann *et. al* [20] built on Bellare *et. al* [3] to define a security notion for online-AE intended to confer some lower level of nonce-reuse misuse-resistance. The approach has gained popularity—many CAESAR submissions follow it, especially after COPA [2] made clear that one could achieve this weakened flavor of nonce-reuse misuse-resistance with a parallelizable scheme. The RAE definition goes a different direction, strengthening instead of weakening the original MRAE definition.

Following up on directions from prior work [10,20,21], AE security in the face of decryption-algorithm leakage was studied by Andreeva *et. al* [1] in work concurrent with our own. A principle motivation for those authors has been to express when it is OK for an online decryption algorithm to be incrementally releasing unverified plaintext. For us, this is a direction not taken, for such leakage can never be generically harmless [47]. In effect, leaking equality of message prefixes is leaking an enormous amount of information.

Ferguson made clear early on that AE algorithms could fail badly when tags are too short [18]. Still, no definitions for AE security were ever offered appropriate to the short-tag setting. But the general concern for making short MACs work well goes back to Black and Cochran [8] and Wang *et. al* [59].

Some examples of using AES4 where AES itself would do include ALRED, LETTERSOUP, MARVIN, and Pelican [15,16,57]. These inspired our predilection to cut certain AES rounds even when provable security couldn't promise this was fine. The approach should not be confused with that of Minematsu and Tsunoo [37], where AES4 provably *does* suffice for the protocol devised [37]. The approach leverages the low MEDP for AES4, a line of work culminating in the bound of Keliher and Sui [31].

Many authors have proposed ideas to eliminate use of the inverse-direction of a blockcipher in modes that previously needed this. The method we us in AEZ is inspired by Minematsu's OTR [36].

The CAESAR competition [7], organized by Dan Bernstein, was the proximal motivation to define RAE and to try to develop a nice scheme for achieving it.

B Deferred Proofs

B.1 Proof of Theorem 1

It suffices to show that

$$\left|\Pr[\mathcal{A}^{\mathbf{Ideal}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{PRI}_{\Pi}} \Rightarrow 1]\right| \leq (r^2 + r)/|\Sigma|^{\lambda + m_{\min} + 1} + 2q/|\Sigma|^{\lambda} \ .$$

Without loss of generality, assume that $q \leq |\Sigma|^{\lambda-1}$; otherwise the claim is trivial. Consider games G_1–G_4 in Fig. 9. Game G_1 corresponds to game $\mathbf{Ideal}_{\Pi}$ and game G_4 corresponds to game $\mathbf{PRI}_{\Pi}$. We explain the game chain up to the terminal one. Game G_2 is identical to game G_1, except that in procedure Enc, it ensures that ciphertexts C are distinct. Partition the encryption queries based on the nonce, the associated data, and the size of the message. Suppose that in game G_1 we have p partitions of size $s_1, \ldots, s_p \geq 1$. Games G_1 and G_2 are identical-until-bad, and thus

$$\begin{aligned}
\left|\Pr[\mathcal{A}^{G_1} \Rightarrow 1] - \Pr[\mathcal{A}^{G_2} \Rightarrow 1]\right| &\leq \Pr[\mathcal{A}^{G_1} \text{ sets } \mathsf{bad}] \\
&\leq \sum_{i=1}^{p} \frac{s_i(s_i-1)}{|\Sigma|^{m_{\min}+\lambda+1}} \\
&= \sum_{i=1}^{p} \frac{(s_i-1)^2 + (s_i-1)}{|\Sigma|^{m_{\min}+\lambda+1}} \leq \frac{r^2+r}{|\Sigma|^{m_{\min}+\lambda+1}};
\end{aligned}$$

the last inequality is due to the fact that $(s_1-1)+\cdots+(s_p-1) = r$. Game G_3 is a simplified version of game G_2; the change is conservative. Game G_4 is identical to game G_3, except that in procedure Dec, it samples a λ-character string v and returns a non-$\perp$ answer if $v = \mathbf{0}^{\lambda}$, where $\mathbf{0}$ is a canonical point in Σ. Let L' be the multiset of $|C|$ in $\mathcal{A}$'s decryption queries in game G_4, and let L be the multiset $\{\ell \mid \ell \geq 0 \text{ and } \ell + \lambda \in L'\}$. Then

$$\begin{aligned}
\left|\Pr[\mathcal{A}^{G_3} \Rightarrow 1] - \Pr[\mathcal{A}^{G_4} \Rightarrow 1]\right| &\leq \Pr[\mathcal{A}^{G_3} \text{ sets } \mathsf{bad}] \\
&\leq \sum_{\ell \in L} \frac{|\Sigma|^{\ell}}{|\Sigma|^{\ell+\lambda} - q} \\
&= \sum_{\ell \in L} \frac{1}{|\Sigma|^{\lambda} - (q/|\Sigma|^{\ell})} \\
&\leq \sum_{\ell \in L} \frac{1}{|\Sigma|^{\lambda} - q} \leq \frac{q}{|\Sigma|^{\lambda} - q} \leq \frac{2q}{|\Sigma|^{\lambda}};
\end{aligned}$$

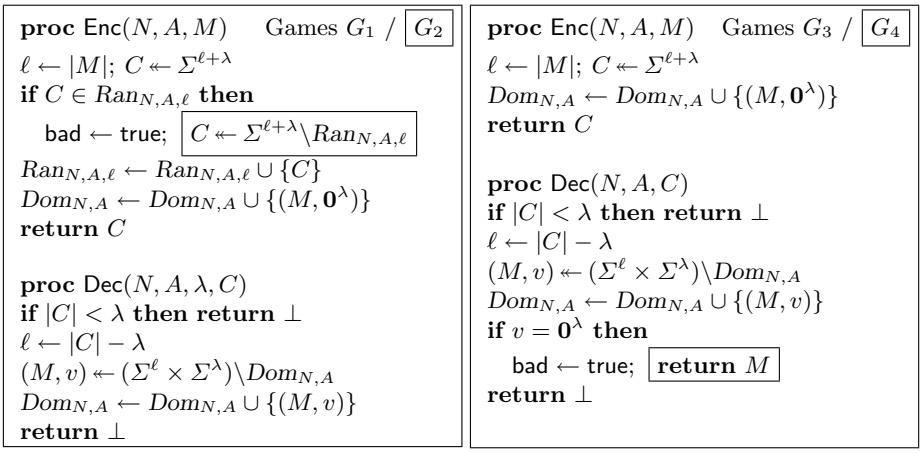


Fig. 9. Games used to prove Theorem 1. Here $\mathbf{0}$ is a canonical element of Σ. Games G_2 and G_4 contain the boxed statements, but games G_1 and G_3 do not.

the last inequality is due to the assumption that $q \leq |\Sigma|^{\lambda-1}$. Summing up,

$$\begin{aligned}\left|\Pr[\mathcal{A}^{\mathbf{Ideal}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{PRI}_{\Pi}} \Rightarrow 1]\right| &\leq \sum_{i=1}^{3}\left|\Pr[\mathcal{A}^{G_i} \Rightarrow 1] - \Pr[\mathcal{A}^{G_{i+1}} \Rightarrow 1]\right| \\ &\leq \frac{r^2 + r}{|\Sigma|^{\lambda+m_{\min}+1}} + \frac{2q}{|\Sigma|^{\lambda}}\end{aligned}$$

as claimed.

B.2 Proof of Theorem 2

The reduction $\mathcal{R}$ creates from $\mathcal{A}$ the adversary $\mathcal{B}$ as follows. It runs $\mathcal{A}$. When the latter makes an encryption query (N, A, λ, M), if $v(M) = 1$ then the former sends the same query to its encryption oracle and returns the answer to $\mathcal{A}$; otherwise it returns $\perp$. When $\mathcal{A}$ makes a query (N, A, λ, C), adversary $\mathcal{B}$ sends the same query to its decryption oracle to get M. If $|M| = |C| - \lambda$ and $v(M) \neq 1$ then it returns $\mathbf{0} \,\|\, M$ to $\mathcal{A}$, where $\mathbf{0}$ is a canonical point in Σ. Otherwise, it returns M. Finally, it outputs the same guess as $\mathcal{A}$.

For any query (N, A, λ, C) that it receives, S' stores $(N, A, \lambda, |C|)$ in a set L_λ. It also maintains, for each (N, A, λ, ℓ) in L_λ, a set $B_{N,A,\lambda,\ell}$. Initially, $B_{N,A,\lambda,\ell} = \Sigma^{\ell-\lambda} \setminus \mathcal{M}_v$. The simulator S' works by running the simulator S. For each query (N, A, λ, C), the simulator S' tosses a biased coin, heads landing land with probability $|B_{N,A,\lambda,\ell}|/(|B_{N,A,\lambda,\ell}| + |\Sigma|^{\ell} - |\Sigma|^{\ell-\lambda})$, where $\ell = |C|$. If heads shows up, simulator S' will sample $M \twoheadleftarrow B_{N,A,\lambda,\ell}$, remove M from $B_{N,A,\lambda,\ell}$, and return M. Otherwise, it runs S on query (N, A, λ, C) and output whatever S returns. Then

$$\begin{aligned}\Pr[\mathcal{A}^{\mathbf{REAL}_{\Pi_v,S'}} \Rightarrow 1] &= \Pr[\mathcal{B}^{\mathbf{REAL}_{\Pi,S}} \Rightarrow 1], \text{ and} \\ \Pr[\mathcal{A}^{\mathbf{RAE}_{\Pi_v,S'}} \Rightarrow 1] &= \Pr[\mathcal{B}^{\mathbf{RAE}_{\Pi,S}} \Rightarrow 1] \ .\end{aligned}$$

Subtracting, we get $\mathbf{Adv}^{\mathrm{rae}}_{\Pi,S}(\mathcal{B}) = \mathbf{Adv}^{\mathrm{rae}}_{\Pi_v,S'}(\mathcal{A})$.

C An Insecure Variant of AEZ-core

Numerous variants of AEZ-core were considered to arrive at AEZ-core. Most simplifications of the final version do not work. As an example, consider trying to cheapen the design by using $c_i \cdot f_{\mathsf{aa},1}(S)$ instead of $f_{\mathsf{aa},i}(S)$ to whiten the middle of each Feistel network, where each c_i is a public constant, and the dot is the multiplication in $\mathrm{GF}(2^n)$. For example, one might hope this works for $c_i = 1$ or $c_i = i$. But this modification is insecure for any choice of c_i values.

For each $L \subseteq \{1, \ldots, n+1\}$ let $\theta(L) = \oplus_{i \in L}\ c_i$. Let $D \neq \emptyset$ be a subset of $\{1, \ldots, n+1\}$ such that $\theta(D) = 0^n$. Such a set D must exists. Assume to the contrary that $\theta(L) \neq 0^n$ for all nonempty $L \subseteq \{1, \ldots, n+1\}$. Then for any distinct nonempty subsets $L, L' \subseteq \{1, \ldots, n+1\}$, we have $\theta(L) \neq \theta(L')$. This means that for $2^{n+1}-1$ nonempty subsets $L \subseteq \{1, \ldots, n+1\}$ we have $2^{n+1}-1 > 2^n$ corresponding distinct elements $\theta(L)$ of $\mathrm{GF}(2^n)$, which is a contradiction.

We now describe an attack to the modified AEZ-core. Our attack only uses strings of length $\ell = 2n(n+3)$. Let M and $\tilde{M}$ be arbitrary distinct ℓ-bit strings such that they agree everywhere except the last two blocks. Query M and $\tilde{M}$ to the first oracle to get answers C and $\tilde{C}$ respectively. In the real game, we'll have $X_i = \tilde{X}_i$ and $\tilde{Y}_i = Y_i \oplus (c_i \cdot (S \oplus \tilde{S}))$ for every $1 \leq i \leq n+2$. Next, let C^* be the "mixed" ciphertext such that, for every $1 \leq i \leq n+3$, the $(2i-1)$'th and $2i$'th blocks of C^* are the same as those of $\tilde{C}$ if $i \in D$, otherwise C^* would borrow the corresponding two blocks of C. Query C^* to the second oracle to get an answer M^*. Let $\overline{D} = \{1, \ldots, n+2\} \backslash D$. In the real game, the query C^* will generate $Y_i^* = \tilde{Y}_i$ for every $i \in D$, and $Y_i^* = Y_i$ for every $i \in \overline{D}$. Then

$$Y^* = \bigoplus_{i \in D} \tilde{Y}_i \oplus \bigoplus_{j \in \overline{D}} Y_j = Y \oplus \bigoplus_{i \in D} ((S \oplus \tilde{S}) \cdot c_i) = Y \quad .$$

Consequently, $S^* = S$ and thus M^* and M agree at the $(2n+3)$th and $(2n+4)$th blocks. The latter event happens with probability at mos 2^{-n} in the random game. Hence this attack wins with advantage at least $1 - 2^{-n}$.

References

1. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. Cryptology ePrint report 2014/144, February 25, 2014
2. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 424–443. Springer, Heidelberg (2013)
3. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: Online ciphers and the hash-CBC construction. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 292–309. Springer, Heidelberg (2001)
4. Bellare, M., Rogaway, P.: On the construction of variable-input-length ciphers. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 231–244. Springer, Heidelberg (1999)

5. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: how to exploit nonces or redundancy in plaintexts for efficient cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer, Heidelberg (2000)
6. Bellare, M., Rogaway, P., Spies, T.: The FFX mode of operation for format-preserving encryption. Draft 1.1. Submission to NIST, February 20, 2010
7. Bernstein, D.: Cryptographic competitions: CAESAR call for submissions, final, January 27, 2014. http://competitions.cr.yp.to/caesar-call.html
8. Black, J., Cochran, M.: MAC reforgeability. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 345–362. Springer, Heidelberg (2009)
9. Black, J.A., Rogaway, P.: Ciphers with arbitrary finite domains. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 114–130. Springer, Heidelberg (2002)
10. Boldyreva, A., Degabriele, J., Paterson, K., Stam, M.: On symmetric encryption with distinguishable decryption failures. Cryptology ePrint Report 2013/433 (2013)
11. Chakraborty, D., Nandi, M.: An improved security bound for HCTR. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 289–302. Springer, Heidelberg (2008)
12. Chakraborty, D., Sarkar, P.: HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. IEEE Transactions on Information Theory **54**(4), 1683–1699 (2008)
13. Chakraborty, D., Sarkar, P.: A new mode of encryption providing a tweakable strong pseudo-random permutation. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 293–309. Springer, Heidelberg (2006)
14. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag, Heidelberg (2002)
15. Daemen, J., Rijmen, V.: A new MAC construction ALRED and a specific instance ALPHA-MAC. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 1–17. Springer, Heidelberg (2005)
16. Daemen, J., Rijmen, V.: The Pelican MAC function. Cryptology ePrint report 2005/088 (2005)
17. Dworkin, M.: Recommendation for block cipher modes of operation: methods for format-preserving encryption. NIST Special Publication 800–38G: Draft, July 2013
18. Ferguson, N.: Authentication weaknesses in GCM. Manuscript, May 20, 2005
19. Fisher, R., Yates, F.: Statistical tables for biological, agricultural and medical research. Oliver & Boyd, London (1938)
20. Fleischmann, E., Forler, C., Lucks, S.: McOE: a family of almost foolproof on-line authenticated encryption schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
21. Fouque, P., Joux, A., Martinet, G., Valette, F.: Authenticated on-line encryption. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 145–159. Springer, Heidelberg (2004)
22. Halevi, S.: EME*: extending EME to handle arbitrary-length messages with associated data. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 315–327. Springer, Heidelberg (2004)
23. Halevi, S.: Invertible universal hashing and the TET encryption mode. Cryptology ePrint report 2007/014
24. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 292–304. Springer, Heidelberg (2004)
25. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003)

26. Hoang, V.T., Krovetz, T., Rogaway, P.: AEZ v3: authenticated encryption by enciphering. CAESAR submission (2014)
27. Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer, Heidelberg (2001)
28. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption: AEZ and the problem that it solves. Cryptology ePrint report 2014/793, January 2015 (Full version of this paper)
29. IEEE. 1619.2-2010 - IEEE standard for wide-block encryption for shared storage media. IEEE press (2010)
30. Kaliski Jr., B.S., Rivest, R.L., Sherman, A.T.: Is DES a Pure Cipher? (Results of more cycling experiments on DES). In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 212–226. Springer, Heidelberg (1986)
31. Keliher, L., Sui, J.: Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. IET Information Security **1**(2), 53–57 (2007)
32. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)
33. Krawczyk, H.: Cryptographic extraction and key derivation: the HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg (2010)
34. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
35. McGrew, D.A., Fluhrer, S.R.: The security of the extended codebook (XCB) mode of operation. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 311–327. Springer, Heidelberg (2007)
36. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 275–292. Springer, Heidelberg (2014)
37. Minematsu, K., Tsunoo, Y.: Provably secure MACs from differentially-uniform permutations and AES-based implementations. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 226–241. Springer, Heidelberg (2006)
38. Nandi, M.: Improving upon HCTR and matching attacks for Hash-Counter-Hash approach. Cryptology ePrint report 2008/090, February 28, 2008
39. Naor, M., Reingold, O.: On the construction of pseudo-random permutations: Luby-Rackoff revisited. Journal of Cryptology **12**(1), 29–66 (1999)
40. Naor, M., Reingold, O.: The NR mode of operation. Undated manuscript realizing the mechanism of [39]
41. Patarin, J.: Generic attacks on feistel schemes. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 222–238. Springer, Heidelberg (2001)
42. Patel, S., Ramzan, Z., Sundaram, G.S.: Efficient constructions of variable-input-length block ciphers. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 326–340. Springer, Heidelberg (2004)
43. Patarin, J.: Security of balanced and unbalanced Feistel schemes with linear non equalities. Cryptology ePrint report 2010/293, May 2010
44. Patarin, J.: Security of random feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 106–122. Springer, Heidelberg (2004)

45. Patarin, J., Gittins, B., Treger, J.: Increasing block sizes using feistel networks: the example of the AES. In: Naccache, D. (ed.) Cryphtography and Security: From Theory to Applications. LNCS, vol. 6805, pp. 67–82. Springer, Heidelberg (2012)
46. Percival, C.: Stronger key derivation via sequential memory-hard functions. The BSD Conference (BSDCan), May 2009
47. Reyhanitabar, R., Vizár, D.: Careful with misuse resistance of online AEAD. Unpublished manuscript distributed on the `crypto-competitions` mailing list. August 24, 2014
48. Rogaway, P.: Authenticated-encryption with associated-data. In: ACM CCS 2002, pp. 98–107. ACM Press (2002)
49. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
50. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: A block-cipher mode of operation for efficient authenticated encryption. In: ACM CCS, pp. 196–205 (2001)
51. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
52. Sarkar, P.: Efficient tweakable enciphering schemes from (block-wise) universal hash functions. Cryptology ePrint report 2008/004
53. Sarkar, P.: Improving upon the TET mode of operation. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 180–192. Springer, Heidelberg (2007)
54. Sarkar, P.: Tweakable enciphering schemes using only the encryption function of a block cipher. Cryptology ePrint report 2009/216
55. Schroeppel, R.: Hasty Pudding Cipher Specification. AES candidate submitted to NIST, June 1998. http://richard.schroeppel.name/hpc/hpc-spec (revised May 1999)
56. Shrimpton, T., Terashima, R.S.: A modular framework for building variable-input-length tweakable ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 405–423. Springer, Heidelberg (2013)
57. Simplício, M., Barbuda, P., Barreto, P., Carvalho, T., Margi, C.: The MARVIN message authentication code and the LETTERSOUP authenticated encryption scheme. Security and Communications Networks **2**(2), 165–180 (2009)
58. Struik, R.: AEAD ciphers for highly constrained networks. DIAC 2013 presentation, August 13, 2013
59. Wang, P., Feng, D., Lin, C., Wu, W.: Security of truncated MACs. In: Yung, M., Liu, P., Lin, D. (eds.) Inscrypt 2008. LNCS, vol. 5487, pp. 96–114. Springer, Heidelberg (2009)
60. Wang, P., Feng, D., Wu, W.: HCTR: a variable-input-length enciphering mode. In: Feng, D., Lin, D., Yung, M. (eds.) CISC 2005. LNCS, vol. 3822, pp. 175–188. Springer, Heidelberg (2005)
61. Yao, F., Yin, Y.L.: Design and analysis of password-based key derivation functions. IEEE Trans. on Information Theory **51**(9), 3292–3297 (2005)