IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2023, No. 4, pp. 452–488. DOI:10.46586/tosc.v2023.i4.452-488

Key Committing Security of AEZ and More

Yu Long Chen[1], Antonio Flórez-Gutiérrez[2], Akiko Inoue[3], Ryoma Ito[4], Tetsu Iwata[5], Kazuhiko Minematsu[3], Nicky Mouha[6], Yusuke Naito[7], Ferdinand Sibleyras[2] and Yosuke Todo[2]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
yulong.chen@esat.kuleuven.be
[2] NTT Social Informatics Laboratories, Musashino, Japan
antonio.florez@ntt.com,ferdinand.sibleyras@ntt.com,yosuke.todo@ntt.com
[3] NEC Corporation, Kawasaki, Japan
a_inoue@nec.com,k-minematsu@nec.com
[4] National Institute of Information and Communications Technology, Koganei, Japan
itorym@nict.go.jp
[5] Nagoya University, Nagoya, Japan
tetsu.iwata@nagoya-u.jp
[6] Strativia, Largo, MD, United States
nicky@mouha.be
[7] Mitsubishi Electric Corporation, Kanagawa, Japan
Naito.Yusuke@ce.MitsubishiElectric.co.jp

Abstract. For an Authenticated Encryption with Associated Data (AEAD) scheme, the key committing security refers to the security notion of whether the adversary can produce a pair of distinct input tuples, including the key, that result in the same output. While the key committing security of various nonce-based AEAD schemes is known, the security analysis of Robust AE (RAE) is largely unexplored. In particular, we are interested in the key committing security of AEAD schemes built on the Encode-then-Encipher (EtE) approach from a wide block cipher. We first consider AEZ v5, the classical and the first dedicated RAE that employs the EtE approach. We focus our analysis on the core part of AEZ to show our best attacks depending on the length of the ciphertext expansion. In the general case where the Tweakable Block Cipher (TBC) is assumed to be ideal, we show a birthday attack and a matching provable security result. AEZ adopts a simpler key schedule and the prove-then-prune approach in the full specification, and we show a practical attack against it by exploiting the simplicity of the key schedule. The complexity is 2^{27}, and we experimentally verify the correctness with a concrete example. We also cover two AEAD schemes based on EtE. One is built on Adiantum, and the other one is built on HCTR2, which are two wide block ciphers that are used in real applications. We present key committing attacks against these schemes when used in EtE and matching proofs for particular cases.

Keywords: AEAD · AEZ · Adiantum · HCTR2 · Key Committing Security

1 Introduction

An Authenticated Encryption with Associated Data (AEAD) scheme is a symmetric primitive for privacy and integrity. The two important security notions for AEAD are privacy and authenticity, which are the conventional security notions of AEAD schemes as a keyed primitive. The development of AEAD schemes has identified that these two notions do not necessarily capture the security requirements expected in real applications.

Received: 2023-09-01 Accepted: 2023-11-01 Published: 2023-12-08

One of the notions we focus on in this paper is the key committing security. In this notion, we consider whether a ciphertext returned by the encryption algorithm commits to a key, nonce, AD, and a plaintext, *i.e.*, we expect that decryption of the ciphertext with another key, nonce, or AD fails to return rejection. Key committing security is in the known-key or chosen-key setting. Hence, it is not covered by the classical notions of AEAD schemes. This notion was initiated by Farshim *et al.* [FOR17] and is relevant in various practical applications, including end-to-end encrypted messaging systems [GLR17, DGRW18, IIM21], key rotation in key management services, envelop encryption, Subscribe with Google (SwG) [ADG+22], and password-based encryption and key exchange [LGR21].

We consider a class of AEAD schemes that achieve the strongest robustness, called Robust Authenticated Encryption (RAE) [HKR15]. This generalizes Deterministic AE (DAE) and nonce-reuse Misuse-Resistant AE (MRAE) [RS06]. RAE schemes have several desirable features, including nonce-misuse resistance, flexibility of the ciphertext expansion, and a type of security against the release of unverified plaintexts [ABL+14]. RAE can be obtained through Encode-then-Encipher (EtE) [BR00] using a tweakable Wide-Block Cipher (WBC) as the underlying primitive. EtE is a well-known and strong approach to designing authenticated encryption, and Grubbs *et al.* [GLR17] proved a variant of key committing security (receiver-binding property) of EtE. However, the underlying WBC is assumed to be ideal. They suggested AEZ [HKR15] as one of the possibly safe instantiations for key committing EtE, however, no concrete analysis was shown. Beyond this result, the key committing security of EtE is largely unexplored, which is the focus of this paper.

In this paper, we consider three RAE schemes built on the EtE approach, AEZ, EtE-Adiantum, and EtE-HCTR2, and we analyze the key committing security of these schemes.

Committing Security. In this paper, we adopt the "committing security" notions by Bellare and Hoang [BH22]. We remark that the security formalization has evolved over the years [FOR17, GLR17, DGRW18, ADG+22, MLGR23], and we refer to Chan and Rogaway [CR22] for a summary of these works and the relationships between these security notions.

We consider the notion for DAE or MRAE following Bellare and Hoang [BH22] adapted to our target schemes. The encryption takes (K, A, M) as input, where K is a key, A is AD, and M is a message and returns a ciphertext C. In the CMT-1 notion, the adversary wins if it returns (K, A, M) and (K', A', M') such that (K, A, M) and (K', A', M') have the same ciphertext and $K \neq K'$. The CMT-4 notion is stronger, requiring only $(K, A, M) \neq (K', A', M')$ instead of $K \neq K'$. In this paper, we cover both CMT-1 and CMT-4. For the relation between these notions and practical applications, we refer to Bellare and Hoang [BH22].

AEZ. AEZ was proposed by Hoang, Krovetz, and Rogaway at Eurocrypt 2015 [HKR15]. It was submitted to CAESAR [Ber19] and advanced to the third round. It has multiple versions, v1, v1.1, v2, v3, v4, v4.1, v4.2, and v5, and our focus is on the latest version, AEZ v5, which we write as AEZ for simplicity. The changes between the versions are explained in [HKR17, Sect. 8]. AEZ uses a tweakable WBC called AEZ-core for processing plaintexts of 256 bits or longer, and AEZ-tiny for shorter ones. We apply zero appending as the encoding in EtE to obtain an AE. AEZ-core builds on EME [HR04, Hal04] and OTR [Min14], and its structure has a proof of security [HKR15]. The full specification of AEZ adopts the proof-then-prune approach to reduce the number of rounds of the primitive and uses an efficient key schedule to gain efficiency. AEZ-tiny is a type of format-preserving encryption [BRRS09] that needs a larger number of Feistel rounds, its security claim is heuristic and not supported by a proof. AEZ is the first attempt to efficiently realize the EtE approach, which is inherently an offline AE to achieve strong robustness. The

computational cost of AEZ is about one AES computation per block, so its throughput is close to AES-CTR mode.

The security of AEZ in the secret key setting has been well analyzed. See [BDD+17, CG16, FLS15, FLLW17] for analyses of earlier versions in the classical setting, [Men17] for an analysis of weak keys, and [KLLN16, Bon17] for analyses in the quantum setting. AEZ remains an important target of cryptanalysis as it is the first dedicated EtE construction to achieve RAE security. From the application perspective, AEZ has attracted several open-source projects including Tor [Mat15], and both JavaScript and WebAssembly implementations are provided by the Node.js package manager [Gug18, Mok18]. We also remark that NIST has held a workshop on block cipher modes of operation [Nat23, MD23], where tweakable wide encryption techniques were among the topics of discussion, and AEZ is a representative example of this category.

We present the analysis of AEZ in terms of key committing security. Specifically, we focus our analysis on AEZ-core, as this forms the core part of the scheme, and it comes with a proof of security.[1] We present the following results:

- We first point out that a straightforward CMT-4 attack exists against general AEZ, which is AEZ where the underlying Tweakable Block Cipher (TBC) is assumed to be ideal. The attack is efficient and works with $O(1)$ complexity.

- We then consider CMT-1 attacks. We divide our analysis into cases depending on τ, which is a parameter that specifies the stretch of the ciphertext. For n, the block length of the underlying TBC, we consider the case $\tau = n$, which is the default case, and the case $\tau < n$. We show that there is a CMT-1 attack against the default case with the birthday complexity of $O(2^{n/2})$. For the case $\tau < n$, we show CMT-1 attacks with the generic attack, the attack based on the 4-tree algorithm [Wag02], the attack based on the repeated 4-tree algorithm, and the birthday attack, depending on the value of τ. Here, the repeated 4-tree algorithm is a variant of the 4-tree algorithm [Wag02] that we formalize here. It works when the size of the lists is insufficient. Our findings are illustrated in Fig. 4.

- To see the tightness of our attack against general AEZ, we present a provable security result when $\tau = n$, assuming that the primitives are ideal. By reducing the CMT-1 attack into a collision-finding problem of a part of the output of general AEZ, we show an $O(2^{n/2})$ provable security bound, indicating the tightness of our attack for this case. We remark that, to our knowledge, this is the first provable security result of the key committing security for a non-monolithic EtE scheme.

- We then consider AEZ, where the underlying TBC follows the full specification of AEZ-core, which we call full-spec AEZ. By fully utilizing the details of the key schedule adopted in full-spec AEZ, we carefully choose distinct keys (K, K') so that the difference in certain intermediate states becomes zero with a high probability. As a result, we obtain a CMT-1 attack against full-spec AEZ with a complexity of 2^{27}, whose correctness we have experimentally verified. We provide a numerical example of the CMT-1 attack against full-spec AEZ.

Therefore, in general, the structure of AEZ is sound with respect to key committing security as the provable security result on general AEZ shows, while the full specification of AEZ is practically insecure due to the details of the key schedule, as the attack against full-spec AEZ shows.

[1] The security of AEZ-tiny is not supported by proof and its security argument is only heuristically justified by the designers. Moreover, AEZ-tiny has a very different structure compared to the other three constructions we have studied. Therefore, we leave it for future work.

EtE-Adiantum. Adiantum is a wide-block encryption scheme designed and proposed by Crowley and Biggers [CB18] and is widely deployed in practice as a disk sector encryption scheme and for filesystem-level encryption (fscrypt) on Android devices. It uses a variant of the Feistel structure called HBSH that follows [MT05]. It uses a universal hash function, a block cipher, and a stream cipher as the underlying primitive. Adiantum has a proof of security as a wide block cipher [MT05, CB18]. It was designed to be efficient on lower-end processors, and it adopts the combination of NH [BHK+99] and Poly1305 [Ber05] as the hash function, AES-256 as the block cipher, and XChaCha12 as the stream cipher to be efficient on such platforms. To our knowledge, an EtE AEAD scheme based on Adiantum, which we write EtE-Adiantum, has not been deployed to a real product. However, as mentioned, Adiantum is widely used in practice and is a high-profile target of cryptanalysis for its practical importance.

We analyze the key committing security of EtE-Adiantum. In AEZ, the message is first zero-appended to encode according to the specification. For EtE-Adiantum, it turns out that zero-prepending and zero-appending can have different security characteristics, and we consider both cases for EtE-Adiantum. We present the following results:

- We point out that an efficient CMT-4 attack is possible against both the prepending and appending cases.
- For a CMT-1 attack, we discuss that the problem is reduced to a collision-finding problem. As a result, we show that the key committing security of EtE-Adiantum is at most $O(2^{\tau/2})$ for the prepending case and $O(2^{n/2})$ for the appending case, where τ is the bit length of the zero-padding and n is the block length of the block cipher.
- We show that our attack on the prepending case is tight by showing a proof of security, assuming the cryptographic permutation inside XChaCha12 is ideal. More precisely, we consider prepending 0^τ to the input, where $\tau \leq n$, and we show $O(2^{\tau/2})$ security of EtE-Adiantum. To complete our proof, we prove a lemma showing the upper bound on the s-way collision probability of a permutation-based Davies-Meyer mode.

EtE-HCTR2. HCTR2 is a wide-block encryption scheme designed by Crowley, Huckleberry, and Biggers [CHB21]. It is based on HCTR [WFW05] to improve the security bound and to address an issue in [CN08] of the provable security result. HCTR2 uses a polynomial hash function as a universal hash function, AES as a block cipher, and a mode of stream encryption called XCTR, and it has a proof of security as a wide block cipher [CHB21]. As in the case of Adiantum, we do not know a practical deployment of EtE-HCTR2. However, HCTR2 is practically deployed as a part of fscrypt [And23] as well as Adiantum, and is under consideration in an extension of UDP proxy optimized for QUIC [PRS23]. Thus, we consider that analyzing the security of EtE-HCTR2 is of interest. As in the case of EtE-Adiantum, we consider the zero-prepending and zero-appending cases, and we present the following results:

- There is an efficient CMT-4 attack against both the prepending and appending cases.
- There is a CMT-1 attack with a birthday complexity. More precisely, following the CMT-1 attack against EtE-Adiantum, we can reduce the CMT-1 attack against EtE-HCTR2 into a collision finding problem to show that the key committing security of EtE-HCTR2 is at most $O(2^{\tau/2})$ for the appending case and $O(2^{n/2})$ for the prepending case.

See Table 1 for the summary of our results.

Table 1: Our Results, where $n = \tau = 128$. The second to fourth columns denote the attack complexity and the last denotes the bit security. CMT-x A (P) denotes CMT-x security for 0^τ Appending (Prepending). Proofs are given for general AEZ and Adiantum with 0^n prepending. For general AEZ, we present a fine-grained CMT-1 analysis for $\tau < n$ (Sect. 3.3).

Scheme	CMT-1 A	CMT-1 P	CMT-4 (A & P)	Proof
general AEZ	$O(2^{n/2})$	(not specified)	$O(1)$	$n/2$ (Sect. 7.1)
full-spec AEZ	2^{27}	(not specified)	$O(1)$	—
EtE-Adiantum	$O(2^{n/2})$	$O(2^{n/2})$	$O(1)$	$n/2$ (Sect. 7.2)
EtE-HCTR2	$O(2^{n/2})$	$O(2^{n/2})$	$O(1)$	—

Organization. In Sect. 2, we give preliminaries to define security notions and the specifications of AEZ, Adiantum, and HCTR2. In Sect. 3, we present our attacks against general AEZ, covering both CMT-4 and CMT-1 attacks. In Sect. 4, we present a practical CMT-1 attack against full-spec AEZ, including a numerical example. We cover attacks against EtE-Adiantum in Sect. 5 and against EtE-HCTR2 in Sect. 6. In Sect. 7, we present provable security results. We cover general AEZ in Sect. 7.1 and Adiantum in Sect. 7.2. We conclude the paper in Sect. 8.

2 Preliminaries

For non-negative integers $i < j$, let $[i..j] \coloneqq \{i, i+1, \ldots, j\}$. For $1 \leq i$, let $[i] \coloneqq \{1, \ldots, i\}$. Let $\{0,1\}^i$ be the set of all i-bit strings and $\{0,1\}^*$ ($\{0,1\}^{8*}$) be the set of all bit (byte) strings. For $X \in \{0,1\}^*$, $|X|$ denotes its length in bits. The empty string is denoted by ε, where $|\varepsilon| = 0$. Let $\{0,1\}^{\leq b}$ denote $\bigcup_{i=0,1,\ldots,b}\{0,1\}^i$, where $\{0,1\}^0 = \{\varepsilon\}$. For $X, Y \in \{0,1\}^*$, $X \,\|\, Y$ is their concatenation. It is also written as XY if it is clear from the context. Let 0^i be the string of i zero bits. For $X \in \{0,1\}^*$ with $|X| \geq i$, $\mathtt{msb}_i(X)$ are the first (left) i bits of X, and $\mathtt{lsb}_i(X)$ are the last (right) i bits of X. If X is uniformly chosen from the set $\mathcal{X}$, we write $X \xleftarrow{\$} \mathcal{X}$. The addition and subtraction in the group $\mathbb{Z}/2^n\mathbb{Z}$ are denoted by $\boxplus$ and $\boxminus$. Here, we use n to denote the block length and set it as $n = 128$ unless explicitly stated otherwise.

(Tweakable) Block Ciphers. A tweakable block cipher (TBC) [LRW11] is a keyed function $E : \mathcal{K} \times \mathcal{T}_w \times \mathcal{M} \to \mathcal{M}$ such that for any (key, tweak) pair $(K, T) \in \mathcal{K} \times \mathcal{T}_w$, $E(K, T, \cdot)$ is a permutation over $\mathcal{M}$. A tweak T is a public value that may be chosen by the adversary. The encryption of a plaintext $M \in \mathcal{M}$ with a key K and a tweak T is a ciphertext $C = E(K, T, M)$, which is also written as $E_K(T, X)$ or $E_K^T(X)$. Similarly, the decryption is written as $M = E^{-1}(K, T, C)$ or $E_K^{-1}(T, C)$ or $(E_K^T)^{-1}(C)$. A TBC with a singleton tweak space $|\mathcal{T}_w| = 1$ is interpreted as a conventional block cipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$. We write $E_K^{-1}(*)$ to denote the decryption function when it is clear that we use E as a (non-tweakable) block cipher.

If the message length of a TBC is variable, the scheme has various names: a variable-length TBC, a wide block cipher (WBC), or a Tweakable Enciphering Scheme. The tweak may or may not be of variable length. Throughout the paper, we adopt the name WBC. Typically, the message space of a WBC is not the full $\{0,1\}^*$ but has a certain minimum length. For example, the most common block cipher-based WBCs have a minimum message length of n bits, where n denotes the block size of the underlying block cipher.

Ideal primitives. For finite sets $\mathcal{X}$ and $\mathcal{Y}$, let $\mathsf{Func}(\mathcal{X}, \mathcal{Y})$ be the set of all the functions: $\mathcal{X} \to \mathcal{Y}$. Similarly, let $\mathsf{TPerm}(\mathcal{T}_w, \mathcal{M})$ denote the set of all tweakable permutations over

$\mathcal{M}$ with tweak space $\mathcal{T}_w$, and let $\mathsf{Perm}(\mathcal{M})$ be the set of all permutations over $\mathcal{M}$.

An ideal cipher (IC) $\overline{\pi} : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ uniformly distributes over $\mathsf{TPerm}(\mathcal{K}, \mathcal{M})$. It is used as a public function so that the adversary can query a tuple $(K, X) \in \mathcal{K} \times \mathcal{M}$ in the forward or backward direction. The oracle returns $Y = \overline{\pi}(K, X)$ ($Y = \overline{\pi}^{-1}(K, X)$) for the forward (backward) direction. Similarly, an ideal tweakable cipher $\widetilde{\pi} : \mathcal{K} \times \mathcal{T}_w \times \mathcal{M} \to \mathcal{M}$ uniformly distributes over $\mathsf{TPerm}(\mathcal{K} \times \mathcal{T}_w, \mathcal{M})$, and a (public) random permutation $\pi : \mathcal{M} \to \mathcal{M}$ uniformly distributes over $\mathsf{Perm}(\mathcal{M})$. Both accept forward and backward queries (including the tweak as a part of the query for the former) as in the case of an IC.

(Deterministic) Authenticated Encryption. Deterministic authenticated encryption (DAE) is a class of authenticated encryption (AE) introduced by [RS06]. Unlike conventional nonce-based AE schemes such as GCM or OCB, it does not necessarily require a nonce if our confidentiality/privacy goal is Deterministic Privacy [RS06] (note that a DAE still needs a non-repeating nonce to achieve the standard privacy notion). The syntax of DAE is as follows. Let $\mathsf{DAE} = (\mathsf{DAE}.\mathcal{E}, \mathsf{DAE}.\mathcal{D})$ be a DAE scheme. The (deterministic) encryption algorithm $\mathsf{DAE}.\mathcal{E}$ takes a key $K \in \mathcal{K}$ and a tuple (A, M) consisting of associated data (AD) $A \in \mathcal{A}$ and a plaintext $M \in \mathcal{M}$ as input, and returns a ciphertext $C \in \mathcal{M}$. Note that $|C| > |M|$ must hold for authenticity. The (deterministic) decryption algorithm $\mathsf{DAE}.\mathcal{D}$ takes $K \in \mathcal{K}$ and the tuple $(A, C) \in \mathcal{A} \times \mathcal{M}$ as input, and returns $M \in \mathcal{M}$ or the reject symbol $\perp$. While not mandatory, a DAE scheme may contain a nonce as an independent variable or as a part of the AD. In this case, a DAE scheme may be called a (nonce) misuse-resistant AE (MRAE) as it offers the best possible protection against nonce repetition when encrypting. Since our focus is not on standard DAE security notions for confidentiality and integrity, we refer to [RS06] for their definitions.

A secure WBC $\Pi : \mathcal{K} \times \mathcal{T}_w \times \mathcal{M} \to \mathcal{M}$ can be converted into a DAE/MRAE by the popular encode-then-encipher (EtE) approach [BR00]. For an encryption input $(A, M) \in \mathcal{A} \times \mathcal{M}$ such that $\mathcal{A} \subseteq \mathcal{T}_w$, let the ciphertext $C = \Pi_K(A, M \,\|\, 0^\tau)$ for some $\tau > 0$, assuming an encoding of A into the tweak space. Often $\tau \in [n]$ is fixed for the block size n. The verification for (A, C) is done by checking if $\mathtt{lsb}_\tau(\Pi_K^{-1}(A, C)) = 0^\tau$ holds. The DAE security of EtE with a WBC is reduced to the pseudorandomness of WBC [BR00]. The security depends on the redundancy introduced by the encoding (τ), and one can use other injective encodings of M with an appropriate checking method. Specifically, we focus on 0^τ appending ($C = \Pi_K(A, M \,\|\, 0^\tau)$) and 0^τ prepending ($C = \Pi_K(A, 0^\tau \,\|\, M)$) as the most natural options. Let EtE-Π denote the EtE using the WBC Π. We may write Π to mean EtE-Π if it is clear from the context.

We mostly follow the original notation for each of the target schemes. Hence, the notation may have slight differences. For details, we refer to each specification section.

2.1 Security Notions

We adopt the "committing security" notions by Bellare and Hoang [BH22]. They succinctly capture various goals of the key committing property. While sharing the same spirit, namely finding a pair of inputs, including the keys, that produces the same output, different notions for key committing security have been proposed in the literature [FOR17, GLR17, DGRW18, ADG+22, MLGR23]. Chan and Rogaway [CR22] provided an excellent summary of these works and the relationships between these security notions and their notions.

We describe the notions proposed by Bellare and Hoang [BH22]. For simplicity, our description is dedicated to DAE (or MRAE; in that case, AD is assumed to contain the nonce), but it is originally defined for general nonce-based AE.

Definition 1. Let $\mathsf{DAE} = (\mathsf{DAE}.\mathcal{E}, \mathsf{DAE}, \mathcal{D})$ be a DAE scheme with a key space $\mathcal{K}$, a message space $\mathcal{M}$, and an AD space $\mathcal{A}$. The CMT-1 notion is the maximum advantage of the adversary whose goal is to find two input tuples $(K, A, M), (K', A', M') \in \mathcal{K} \times \mathcal{A} \times \mathcal{M}$

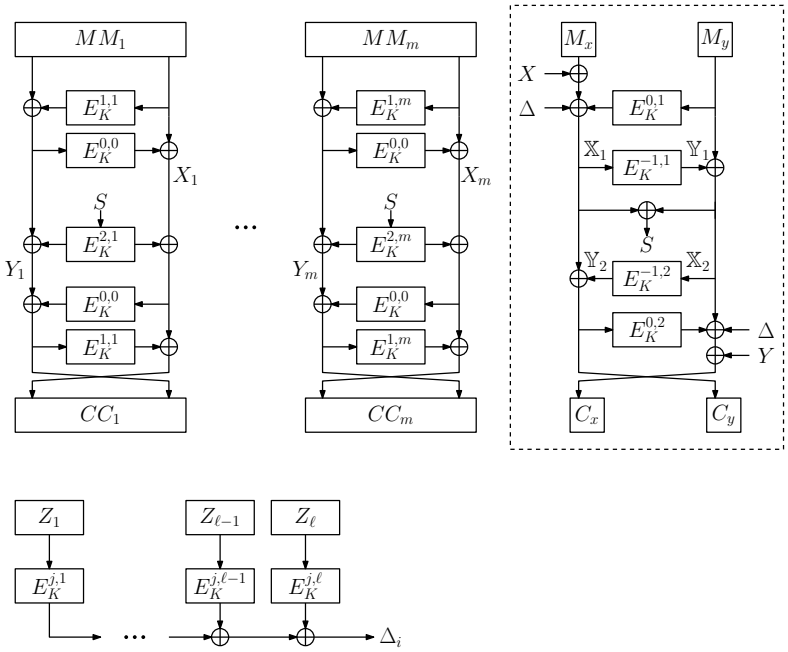

Figure 1: AEZ (AEZ-core) for the even-block case, using TBC E. Here, $X = \bigoplus_{i\in[m]} X_i$ and $Y = \bigoplus_{i\in[m]} Y_i$. $\Delta = \bigoplus_i \Delta_i$. The sequence $(Z_1 \ldots, Z_\ell)$ denotes i-th string in the vector $\mathbf{T}$ containing AD, and $j = i + 2$. Note that $E_K^{-1,*}$ denotes the forward evaluation with the tweak $(-1, *)$, not the inverse.

such that $K \neq K'$ and $\mathsf{DAE}.\mathcal{E}(K, A, M) = \mathsf{DAE}.\mathcal{E}(K', A', M')$. The CMT-1 advantage is defined as

$$\begin{aligned}\mathbf{Adv}^{\mathsf{cmt\text{-}1}}_{\mathsf{DAE}}(\mathsf{A}) \coloneqq \Pr[((K, A, M), (K', A', M')) \leftarrow \mathsf{A} \\ \text{s.t. } K \neq K', \mathsf{DAE}.\mathcal{E}(K, A, M) = \mathsf{DAE}.\mathcal{E}(K', A', M')].\end{aligned}$$

The CMT-4 notion is identical to CMT-1 except that only $(K, A, M) \neq (K', A', M')$ is required instead of $K \neq K'$. An intermediate notion, CMT-3, requires $(K, A) \neq (K', A')$. However, Bellare and Hoang [BH22] showed that it is equivalent to CMT-4. They also proposed a class of CMTD notions whose winning condition is defined via decryption functions rather than encryption functions. Bellare and Hoang [BH22] showed that CMTD-$\mathtt{x}$ is equivalent to CMT-$\mathtt{x}$ for $\mathtt{x} \in \{1, 3, 4\}$ when the target scheme is *tidy*.[2] As all the analyzed schemes in this paper are tidy, we only consider CMT-$\mathtt{x}$ notions. The same approach was taken by [MLGR23]. Note that the adversary in these notions can choose keys. Hence, proving the security under these notions will require an idealized primitive inside the target to make these notions meaningful (see *e.g.* [Rog06]).

2.2 Specification of AEZ

We study the latest specification (v5) [HKR17] of AEZ. It consists of two WBCs, AEZ-tiny and AEZ-core, where the former is used for plaintexts shorter than 256 bits, and the latter is used otherwise. The standard EtE method with zero appending is applied to the WBC to implement an AE. As the name suggests, we focus on the latter case using AEZ-core as it is the "core" construction of AEZ. AEZ-tiny is a kind of format-preserving

[2] An encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is tidy if $M = \mathsf{Dec}(K, C)$ implies $\mathsf{Enc}(K, M)$ returns C.

encryption [BRRS09] and needs a larger number of Feistel rounds. We may write AEZ to mean AEZ-core throughout the paper.

Mode. The original pseudocodes of [HKR17] are shown in Appendix B (Algs. 1, 2 and 3). See also Fig. 1. We omit the pseudocodes for AEZ-tiny and BLAKE2b [ANWW13]. The structure of (the internal WBC of) AEZ is Encrypt-Mix-Encrypt, where "Encrypt" is realized by a two-round Feistel permutation and "Mix" is a combination of a linear operation and one encryption round per diblock. It is built on the ideas of EME [HR04] and OTR [Min14].

Let $n = 128$. The encryption of AEZ (Alg. 1) takes a key $K \in \{0,1\}^{8*}$, a nonce $N \in \{0,1\}^{8*}$, an AD $\boldsymbol{A} \in (\{0,1\}^{8*})^*$, a tag length[3] $\tau \in \mathbb{N}$, and a plaintext $M \in \{0,1\}^{8*}$. It returns a ciphertext $C \in \{0,1\}^{8*}$ with $|C| = |M| + \tau$. The decryption (Alg. 2) takes $(K, N, \boldsymbol{A}, \tau, C)$ and returns M or $\perp$, an error symbol. The key length $|K|$ is recommended to be at least 128 bits, and the default length is $3n = 384$ bits. The key length does not matter for our generic attacks of Sect. 3.2. The variable τ is a multiple of 8 and denotes the bit-length overhead. The default (and the recommended maximum) value is $\tau = n$. It can vary for each message, however, the designers suggested it is typically fixed. We consider the case $\tau \in [n]$ following these suggestions. An AD in AEZ can consist of multiple bitstrings (*i.e.*, a vector). Unlike the description of Sect. 2, AEZ treats the nonce explicitly; a vector-input PRF AEZ-hash takes an encoding of $(\tau, N, \boldsymbol{A})$, denoted by $\boldsymbol{T}$. We assume that AD is a single bitstring without loss of generality for our attacks and proofs. We do not distinguish nonce and other AD blocks. Moreover, we may write A instead of $\boldsymbol{A}$ and write AEZ-hash(K, A) instead of AEZ-hash$(K, \boldsymbol{T})$. We mostly follow the original notation, however, we apply slight modifications to them for convenience. In particular, our attack descriptions will use MM_i (CC_i) to denote the $2n$-bit sequence (diblock) M_iM_i' (C_iC_i') for $i \in [m]$ in the original pseudocodes.

Primitive. AEZ is based on a TBC $E : \mathcal{K} \times \mathcal{T}_w \times \mathcal{M} \to \mathcal{M}$, where $\mathcal{K} = \{0,1\}^{3n}$, $\mathcal{M} = \{0,1\}^n$, $\mathcal{T}_w = \mathcal{I} \times \mathcal{J}$ with $\mathcal{I}, \mathcal{J} \subseteq \mathbb{Z}$; their actual ranges can be found in the pseudocode. The designers proved standard DAE security (more precisely, an even stronger one called robust AE security) assuming the underlying TBC is computationally secure, *i.e.*, a tweakable pseudorandom permutation [HKR15]. In the submission to the CAESAR competition, this TBC is instantiated by either a 4-round or 10-round AES depending on the tweak value, rather than relying on provably secure constructions such as XE [Rog04] built on (unmodified) AES. Moreover, it adopts very simple tweak and key schedules.

The concrete instantiation of TBC $E_K^{i,j}(*)$ is shown in Alg. 3, where AES10$_{\boldsymbol{K}}$ denotes 10-round AES with the list of subkeys $\boldsymbol{K}$. Similarly AES4$_{\boldsymbol{k}}$ denotes 4-round AES with the list of round keys $\boldsymbol{k}$. As the pseudocode shows, if $|K| = 384$, we first parse K into three n-bit parts, I, J, and L, and any n-bit round key in $\boldsymbol{K}$ and $\boldsymbol{k}$ is one of I or J or L. See Appendix B for more details. This instantiation implies that the reduction to AES's pseudorandomness is no longer possible. Even more, the 4-round AES can be practically distinguished from random with a chosen plaintext attack [DKR97, KW02], clearly indicating the deviation from what provable security aims at. The designers named this approach *prove-then-prune*. The security of a prove-then-prune construction depends on the interaction between the mode and the primitive instantiation; thus, it is not always safely applicable. As DAE/MRAE, the security claim of AEZ (v5) using the above TBC instantiation has been maintained since the proposal.

procedure HASHP(K_P, P)
  $K_L \leftarrow \mathtt{msb}_{128}(K_P)$
  $K_N \leftarrow \mathtt{lsb}_{8320}(K_P)$
  $R \leftarrow \mathrm{NH}_{K_N}(\mathrm{pad}(P))$
  return POLY1305$_{K_L}(R)$

procedure HASHT(K_T, ℓ, T)
  return POLY1305$_{K_T}(\ell \,\|\, T)$

procedure HASH(h, T, P)
  $K_T \leftarrow \mathtt{msb}_{128}(h)$
  $K_P \leftarrow \mathtt{lsb}_{8448}(h)$
  $H_T \leftarrow$ HASHT$(K_T, |P|, T)$
  $H_P \leftarrow$ HASHP(K_P, P)
  return $H_T \boxplus H_P$

procedure ENCRYPT(K, T, P)
  $K_E \,\|\, h \leftarrow S_K(\varepsilon, 8832)$ ▷ $|K_E| = 256$
  $P_L \,\|\, P_R \leftarrow P$ ▷ $|P_R| = n$
  $P_M \leftarrow P_R \boxplus$ HASH(h, T, P_L)
  $C_M \leftarrow E_{K_E}(P_M)$
  $C_L \leftarrow P_L \oplus S_K(C_M, |P_L|)$
  $C_R \leftarrow C_M \boxminus$ HASH(h, T, C_L)
  $C \leftarrow C_L \,\|\, C_R$
  return C

Figure 2: Adiantum with an n-bit block cipher E and a stream cipher S that accepts a nonce and an output length as arguments. K, P, and T denote the key, the plaintext, and the tweak, respectively. ε is the empty string. pad() is a padding that adds 0^* to the end of its input until its size reaches a multiple of n.

2.3 Specification of Adiantum

Adiantum is a WBC proposed by Crowley and Biggers [CB18]. It is based on three primitives: a universal hash H, a block cipher E, and a stream cipher S. The specification is given in Fig. 2. See also Fig. 6 for illustration. In particular, Adiantum implements E as the AES-256 [AES23] block cipher and S as the XChaCha12 stream cipher, which is an extension of ChaCha12 [Ber08] to accept a 192-bit nonce. Adiantum uses XChaCha12 by padding the 128-bit nonce with a one followed by 63 zeroes. The hash function relies on two hash subroutines: NH [BHK+99] and POLY1305 [Ber05].

NH is an efficient hash function that uses an $8,576$-bit key split into 32-bit blocks $k = k_0 \,\|\, \cdots \,\|\, k_{266} \,\|\, k_{267}$, and split the input into 32-bit blocks $M = M_0 \,\|\, \ldots \,\|\, M_{\ell-2} \,\|\, M_{\ell-1}$. Then, it adds and multiplies blocks of the key and the message, appending the result once all the key blocks are used:

$$\mathrm{NH}(k, M, r) = \left(\sum_{i=0}^{267} (k_{2i} + M_{r*268+2i} \bmod 2^{32})(k_{2i+1} + M_{r*268+2i+1} \bmod 2^{32})\right) \bmod 2^{64},$$
$$\mathrm{NH}_k(M) = \mathrm{NH}(k, M, 0) \,\|\, \mathrm{NH}(k, M, 1) \,\|\, \ldots \,\|\, \mathrm{NH}(k, M, \ell/(268 * r)).$$

POLY1305 first masks the 128-bit hash key $\overline{k} = k \wedge (1^{28} \,\|\, 0^6 \,\|\, 1^{26} \,\|\, 0^6 \,\|\, 1^{26} \,\|\, 0^6 \,\|\, 1^{26} \,\|\, 0^4)$ and split the input into 128-bit blocks $M = M_0 \,\|\, \ldots \,\|\, M_{\ell-2} \,\|\, M_{\ell-1}$. Then, it evaluates a polynomial with coefficients depending on M at the point $\overline{k}$:

$$\text{POLY1305}_k(M) = \left(\left(\sum_{i=0}^{\ell-1} \overline{k}^{\ell-i}(M_i + 2^{128})\right) \bmod (2^{130} - 5)\right) \bmod 2^{128}.$$

As mentioned earlier, we consider EtE-Adiantum by either prepending or appending 0^τ to the plaintext P. The tweak T will be used as an AD, which may contain a nonce. The decryption mechanism then rejects every plaintext that does not start (or end) by 0^τ.

[3] In the AEZ specification, the *authenticator length* refers to $\tau/8$ and is written as ABYTES.

```
1: procedure ENCRYPT(K, T, P)
2:     h ← E_K(0)
3:     L ← E_K(1)
4:     P_L ‖ P_R ← P                          ▷ |P_L| = n
5:     C_M ← P_L ⊕ HASH(h, T, P_R)
6:     CC ← E_K(C_M)
7:     N ← C_M ⊕ CC ⊕ L
8:     C_R ← P_R ⊕ XCTR(K, N, |P_R|)
9:     C_L ← CC ⊕ HASH(h, T, C_R)
10:    return C_L ‖ C_R
```

```
1: procedure XCTR(K, N, ℓ)
2:     S ← ε
3:     i ← 1
4:     r ← ℓ
5:     while r > n do
6:         S ← S ‖ E_K(N ⊕ i)
7:         i ← i + 1
8:         r ← r − n
9:     S ← S ‖ msb_r(E_K(N ⊕ i))
10:    return S
```

```
1: procedure HASH(h, T, M)
2:     Y ← 0
3:     A_1 ‖ A_2 ‖ ··· ‖ A_t ← pad(T)
4:     if n divides |M| then
5:         B_1 ‖ B_2 ‖ ··· ‖ B_m ← M
6:         Y ← Y ⊕ (2|T| + 2) · h
7:     else
8:         B_1 ‖ B_2 ‖ ··· ‖ B_m ← pad(M ‖ 1)
9:         Y ← Y ⊕ (2|T| + 3) · h
10:    for i ∈ [t] do
11:        Y ← Y ⊕ A_i · h
12:    for j ∈ [m] do
13:        Y ← Y ⊕ B_j · h
14:    return Y
```

Figure 3: HCTR2 [CHB21] based on an n-bit block cipher E, and processing a key K, a tweak T, and a plaintext P. The function pad() is a padding that adds 0^* to the end of its input until it reaches a size multiple of n.

2.4 Specification of HCTR2

Crowley, Huckleberry, and Biggers designed HCTR2 [CHB21], which is a WBC based on Wang *et al.*'s HCTR [WFW05]. HCTR2 consists of a variant of counter mode and a polynomial hash function. HCTR2's specifications are given in Fig. 3. See also Fig. 7 for illustration. As in the case of Adiantum, we consider EtE-HCTR2 with either prepending or appending 0^τ to the plaintext.

HCTR2 follows the same structure as HCTR with slight differences. For instance, HCTR2 derives the hash key from the block cipher's key, which may make it more resistant to attacks on key committing security. It is fairly straightforward to translate attacks on the key committing security of HCTR2 to the original HCTR; hence, we focus our study on HCTR2. Indeed, independently setting the keys gives the attacker more freedom in looking for other key materials to build a key committing attack. This stands in contrast with standard notions of security, where it is the key derivation that may introduce security flaws.

The hash function of HCTR2 can mostly be understood as a polynomial evaluated at a secret point, the secret key h, with coefficients depending on its inputs. Also, for a given length, it can be rewritten as the sum of two polynomials, one depending on the tweak T and one depending on the input M. Let us define $\text{POLY}_h(X)$ for an x-block input $X = X_1 \,\|\, X_2 \,\|\, \cdots \,\|\, X_x$ as the following polynomial with coefficients given by X and evaluated at h writing finite field multiplication as "$\cdot$":

$$\text{POLY}_h(X) = X_1 \cdot h^{x-1} \oplus X_2 \cdot h^{x-2} \oplus \cdots \oplus X_{x-1} \cdot h \oplus X_x$$

Then, for input sizes $|T|$ and $|M|$ that are multiple of n, the hash function of HCTR2 is:

$$\textsc{Hash}(h, T, M) = (2|T| + 2) \cdot h^{\frac{|T|+|M|}{n}+1} \oplus \textsc{Poly}_h(T) \cdot h^{\frac{|M|}{n}+1} \oplus \textsc{Poly}_h(M) \cdot h$$

This representation will help describe our key committing attacks in Sect. 6.

3 Key Committing Security of General AEZ

We start with attacking AEZ, assuming the underlying TBC $E : \mathcal{K} \times \mathcal{T}_w \times \mathcal{M} \to \mathcal{M}$ is an ideal tweakable cipher. We call such a version general AEZ. The CMT-4 attack is simple and works for any τ with negligible complexity. The complexity of the CMT-1 attack depends on τ. Recall that we consider $\tau \in [n]$; the default is $\tau = n$.

3.1 CMT-4 Attack

We present a CMT-4 attack against general AEZ. The adversary fixes K (of any length) and other input variables, performs encryption, and tries to find another AD that yields a collision on AEZ-hash. Since AEZ-hash is effectively a TBC-based PHASH (the message hashing part of PMAC [Rog04]), finding such an AD pair is trivial because preimages are easy to find given the key. Assuming AD is a single bitstring, an input to AEZ-hash is a vector $\boldsymbol{T}$ of three components. Then the output $\Delta = \mathsf{AEZ\text{-}hash}(K, \boldsymbol{T})$ is written as $\overline{\Delta} \oplus E_K^{3,\ell-1}(Z_{\ell-1}) \oplus E_K^{3,\ell}(Z_\ell)$, where $Z_{\ell-1}$ and Z_ℓ are the last two AD blocks and $\overline{\Delta}$ is a value computed from K and $\boldsymbol{T}$ except $Z_{\ell-1}$ and Z_ℓ. To find a collision, we modify the last two AD blocks to $Z'_{\ell-1}$ and Z'_ℓ so that

$$E_K^{3,\ell-1}(Z_{\ell-1}) \oplus E_K^{3,\ell}(Z_\ell) = E_K^{3,\ell-1}(Z'_{\ell-1}) \oplus E_K^{3,\ell}(Z'_\ell)$$

holds. Concretely, for $i = \ell - 1, \ell$, we let $B_i \leftarrow E_K^{3,i}(Z_i)$ and obtain $Z'_i \leftarrow (E_K^{3,i})^{-1}(B_i \oplus e)$ for some $e \neq 0^n$. Then, a difference e introduced in $i = \ell - 1$ is canceled in $i = \ell$. An example of a complete attack is: first, fix (K, A, M), then, encrypt (A, M) with K to obtain Δ and C, and finally, find another $A' \neq A$ that yields the same Δ and returns $((K, A, M), (K, A', M), C)$. This attack needs two forward and inverse queries with negligible computation and is thus practical.

Remark. A variant of the CMT-4 attack described above works on any EtE scheme if AD is processed by a (keyed) function f and the result is absorbed into the main encryption/decryption routine and f is weak with respect to preimage with knowledge of its key (if keyed). This holds for many WBCs, including Adiantum and HCTR2. We omit the details of the attacks as they are rather straightforward from the paragraph "Preimages using Hash Function" in Sect. 5 and Sect. 6. A cryptographically strong hashing for AD processing would prevent *this type* of CMT-4 attack but does not guarantee CMT-4 security in general.

3.2 CMT-1 Attack for $\tau = n$

Unlike CMT-4, breaking CMT-1 needs a pair of distinct keys for a pair of input tuples. Hence, the internal states are processed by independent random primitives, thereby excluding simple single-variable cancellation strategies as our CMT-4 attack. We first present an attack for the default setting $\tau = n$. The attack requires finding a collision on C_y, which is a birthday attack of $O(2^{n/2})$ time. However, we emphasize that this is not a trivial collision finding on a hash function $(K, A, M) \to C_y$ deduced from general AEZ since other ciphertext blocks must collide.

Overview. Our attack focuses on the last diblock. See the box with a dashed line in Fig. 1. For encryption, it takes the key K, the last plaintext diblock (M_x, M_y), the internal variables X and Y determined by the other diblocks, and the output of AEZ-hash taking AD, Δ. It outputs the last ciphertext diblock (C_x, C_y) and S, which is absorbed by the processing of other diblocks. Following Fig. 1, let $\mathbb{X}_1 = E_K^{0,1}(M_y) \oplus \Delta \oplus X \oplus M_x$, which is an input to $E_K^{-1,1}$. Let $\mathbb{Y}_1 = E_K^{-1,1}(\mathbb{X}_1)$ and $\mathbb{X}_2 = \mathbb{Y}_1 \oplus M_y$, $\mathbb{Y}_2 = E_K^{-1,2}(\mathbb{X}_2)$. When $\tau = n$, $M_y = 0^n$ and $\mathbb{X}_2 = \mathbb{Y}_1$. We first fix a pair of distinct keys, K and K', and the ciphertext diblocks except the last one. Next, we find a collision on C_y by sampling $2^{n/2}$ times for the internal $\mathbb{Y}_1$. Then, we can make the system of equations for the last diblock consistent by exploiting the preimage weakness of AEZ-hash.

For simplicity, we describe the case of even message blocks. Extending to odd blocks can be done by minor modifications after Step 7. For the notation, we refer to Fig. 1 and Appendix B.2.

Attack Procedure.

Step 1. Pick an arbitrary pair of distinct keys $K, K' \in \mathcal{K}$. Pick an arbitrary sequence of diblocks $CC = (CC_1, \ldots, CC_m)$ and C_x, derive $(Y_1, \ldots, Y_m)$ using K. Also derive $(Y'_1, \ldots, Y'_m)$ from CC and K'. Let $Y = Y_1 \oplus \ldots \oplus Y_m$ and $Y' = Y'_1 \oplus \ldots Y'_m$. Set $M_y = M'_y = 0^n$.

Step 2. (Randomly) sample $2^{n/2}$ values of $\widehat{\mathbb{Y}_1}$ as candidates for $\mathbb{Y}_1$, and sample $2^{n/2}$ values of $\widehat{\mathbb{Y}'_1}$ as candidates for $\mathbb{Y}'_1$.

Step 3. For each pair, derive $\widehat{\mathbb{X}_1} = (E_K^{-1,1})^{-1}(\widehat{\mathbb{Y}_1})$, $\widehat{\mathbb{Y}_2} = E_K^{-1,2}(\widehat{\mathbb{Y}_1})$, $\widehat{\mathbb{X}'_1} = (E_{K'}^{-1,1})^{-1}(\widehat{\mathbb{Y}'_1})$ and $\widehat{\mathbb{Y}'_2} = E_{K'}^{-1,2}(\widehat{\mathbb{X}'_2})$. Build two lists for $(\widehat{\mathbb{X}_1} \oplus \widehat{\mathbb{Y}_2})$ and $(\widehat{\mathbb{X}'_1} \oplus \widehat{\mathbb{Y}'_2})$, each of size $2^{n/2}$.

Step 4. Find a collision in the lists which becomes C_y: $C_y = (\widehat{\mathbb{X}_1} \oplus \widehat{\mathbb{Y}_2}) = (\widehat{\mathbb{X}'_1} \oplus \widehat{\mathbb{Y}'_2})$. Determine (the true tuple of) $(\mathbb{X}_1, \mathbb{Y}_1, \mathbb{X}_2, \mathbb{Y}_2)$ as $(\widehat{\mathbb{X}_1}, \widehat{\mathbb{Y}_1}, \widehat{\mathbb{X}_2}, \widehat{\mathbb{Y}_2})$, and $(\mathbb{X}'_1, \mathbb{Y}'_1, \mathbb{X}'_2, \mathbb{Y}'_2)$ as $(\widehat{\mathbb{X}_1}', \widehat{\mathbb{Y}_1}', \widehat{\mathbb{X}_2}', \widehat{\mathbb{Y}_2}')$. Determine $S = \mathbb{X}_1 \oplus \mathbb{X}_2$ and $S' = \mathbb{X}'_1 \oplus \mathbb{X}'_2$.

Step 5. Determine $\Delta_{\text{target}} = C_x \oplus E_K^{0,2}(C_y) \oplus Y \oplus \mathbb{Y}_1$ and $\Delta'_{\text{target}} = C_x \oplus E_{K'}^{0,2}(C_y) \oplus Y' \oplus \mathbb{Y}'_1$.

Step 6. Find $A, A' \in \{0,1\}^{8*}$ so that $\mathsf{AEZ\text{-}hash}(K, A) = \Delta_{\text{target}}$ and $\mathsf{AEZ\text{-}hash}(K', A') = \Delta'_{\text{target}}$. This can be done efficiently by using the inverse of E in the same manner as in a CMT-4 attack.

Step 7. Determine $(X_1, \ldots, X_m)$ from S, CC and K. Similarly determine $(X'_1, \ldots, X'_m)$. Let $X = X_1 \oplus \ldots \oplus X_m$, $X' = X'_1 \oplus \ldots \oplus X'_m$.

Step 8. Determine $M_x = \Delta_{\text{target}} \oplus X \oplus \mathbb{X}_1$, $M'_x = \Delta'_{\text{target}} \oplus X' \oplus \mathbb{X}'_1$.

Step 9. Determine $MM = (MM_1, \ldots, MM_m)$ from CC, S, and K. Similarly determine $MM' = (MM'_1, \ldots, MM'_m)$.

Step 10. Let $M = (MM, M_x, M_y(= 0^n))$ and $M' = (MM', M'_x, M'_y(= 0^n))$. Return $((K, A, M), (K', A', M'), C)$.

3.3 CMT-1 Attack for $\tau < n$

The complexity of a CMT-1 attack can be reduced if τ is smaller than n as it enables us to choose $\mathbb{Y}_1$ and $\mathbb{X}_2$ independently for the first $(n - \tau)$ bits. More specifically, when $\tau \leq 2n/3$, we can use a generalized birthday attack on 4XOR (4-tree algorithm) [Wag02]. The 4-tree algorithm takes four lists $\mathcal{L}_{X_1}$, $\mathcal{L}_{X_2}$, $\mathcal{L}_{X_3}$, and $\mathcal{L}_{X_4}$ as input, where each list

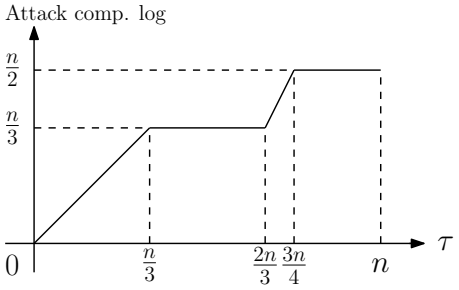

Figure 4: Overview of CMT-1 attack complexities against general AEZ. The vertical line indicates the logarithm of attack complexities (including time and memory). The $\tau \le n/3$ case shows the generic attack complexity. The $n/3 \le \tau \le 2n/3$ case shows the 4-tree algorithm complexity. The $2n/3 \le \tau \le 3n/4$ case shows the repeated 4-tree algorithm complexity. The $3n/4 \le \tau \le n$ case shows the birthday attack complexity.

includes $2^{n/3}$ items of random n-bit strings. The algorithm outputs $(X_1, X_2, X_3, X_4) \in \mathcal{L}_{X_1} \times \mathcal{L}_{X_2} \times \mathcal{L}_{X_3} \times \mathcal{L}_{X_4}$ s.t. $X_1 \oplus X_2 \oplus X_3 \oplus X_4 = 0^n$. It requires the time and memory complexities of $O(2^{n/3})$. Note that 4-tree has been used in the key committing attack on SIV [MLGR23], but ours is different and not derived from their attack. We modify the attack procedure of Sect. 3.2 as follows.

Step 2. Instead of fixing M_y and M'_y, we fix $\mathsf{cst}, \mathsf{cst}' \in \{0,1\}^\tau$ arbitrarily and (randomly) sample $\widehat{\mathbb{Y}_1}$, $\widehat{\mathbb{X}_2}$, $\widehat{\mathbb{Y}'_1}$, and $\widehat{\mathbb{X}'_2}$, each $2^{n/3}$ times (which is possible as $n - \tau \ge n/3$), so that $\mathtt{lsb}_\tau(\widehat{\mathbb{Y}_1})$ and $\mathtt{lsb}_\tau(\widehat{\mathbb{X}_2})$ are fixed to cst. Similarly, $\mathtt{lsb}_\tau(\widehat{\mathbb{Y}'_1})$ and $\mathtt{lsb}_\tau(\widehat{\mathbb{X}'_2})$ are fixed to cst'.

Step 3. We compute $\widehat{\mathbb{X}_1}$ and $\widehat{\mathbb{Y}_2}$ and $\widehat{\mathbb{X}'_1}$ and $\widehat{\mathbb{Y}'_2}$ each for $2^{n/3}$ values and build the lists of these variables, denoted by $\mathcal{L}_{\mathbb{X}_1}$, $\mathcal{L}_{\mathbb{Y}_2}$, $\mathcal{L}_{\mathbb{X}'_1}$, and $\mathcal{L}_{\mathbb{Y}'_2}$.

Step 4. We run the 4-tree algorithm of Wagner [Wag02] on these four lists. With a high probability, we find a solution, *i.e.*, $(\mathbb{X}_1, \mathbb{Y}_2, \mathbb{X}'_1, \mathbb{Y}'_2) \in \mathcal{L}_{\mathbb{X}_1} \times \mathcal{L}_{\mathbb{Y}_2} \times \mathcal{L}_{\mathbb{X}'_1} \times \mathcal{L}_{\mathbb{Y}'_2}$ such that $\mathbb{X}_1 \oplus \mathbb{Y}_2 \oplus \mathbb{X}'_1 \oplus \mathbb{Y}'_2 = 0^n$. We determine $C_y = \mathbb{X}_1 \oplus \mathbb{Y}_2$ $(= \mathbb{X}'_1 \oplus \mathbb{Y}'_2)$, $M_y = \mathbb{X}_2 \oplus \mathbb{Y}_1$, and $M'_y = \mathbb{X}'_2 \oplus \mathbb{Y}'_1$. (Note that $\mathtt{lsb}_\tau(M_y) = \mathtt{lsb}_\tau(M'_y) = 0^\tau$ holds.) Determine $S = \mathbb{X}_1 \oplus \mathbb{X}_2$ and $S' = \mathbb{X}'_1 \oplus \mathbb{X}'_2$.

The remaining steps remain the same. The time and memory complexities of this attack are $O(2^{n/3})$.

Case $2n/3 < \tau < n$. The above attack needs $\tau \le 2n/3$ to ensure we have enough samples for $\widehat{\mathbb{X}_1}$, $\widehat{\mathbb{Y}_2}$, $\widehat{\mathbb{X}'_1}$, and $\widehat{\mathbb{Y}'_2}$. However, even when $2n/3 < \tau$, we can mount a variant of the 4-tree attack by repeating the 4-tree attack with small lists until we find a solution. This repeated 4-tree algorithm works better than the birthday attack when $2n/3 < \tau < 3n/4$, and its complexity is $O(2^{2\tau-n})$ as shown in Fig. 4. See Appendix D for the details.

Case $\tau < n/3$. In case $\tau < n/3$, a simple generic attack against EtE-based AE works. We just pick random 2^τ tuples (K, A, C) and decrypt them until we find a valid plaintext M, *i.e.*, $\mathtt{lsb}_\tau(M_y) = 0^\tau$. We obtain (K, A, M, C) as a valid tuple. We repeat this procedure using another 2^τ tuples of (K', A', C) for some $K' \neq K$ and $A' \neq A$ to find the solution. The time and memory complexities are $O(2^\tau)$.

4 Key Committing Security of Full-spec AEZ

We continue to consider attacking AEZ where the underlying TBC $E : \mathcal{K} \times \mathcal{T}_w \times \mathcal{M} \to \mathcal{M}$ follows the full-spec AEZ-core. We call such a version full-spec AEZ. A CMT-4 attack against full-spec AEZ works for any τ with negligible complexity, as described in Sect. 3.1. Unlike the attack described in Sect. 3.1, we choose the key of full-spec AEZ to provide a CMT-1 attack. Note that the CMT-1 security definition does not restrict whether the attacker can or cannot choose the key. Although we need a stronger assumption, this is a practical attack of 2^{27} time for any τ. We verified the attack experimentally. We provide an example of two data sets enciphering to the same ciphertext in Appendix C.

Overview. This attack follows similar steps to the CMT-1 attack described in Sect. 3.2. Namely, this attack also focuses on the last diblock, which is illustrated as the dashed line box in Fig. 1, and finding $(\mathbb{X}_1 \oplus \mathbb{Y}_2) = (\mathbb{X}'_1 \oplus \mathbb{Y}'_2)$. The notation in this section is almost identical to Sect. 3.2, with the following differences. We use the round keys of $E_K^{-1,i}(\mathbb{X})$, *i.e.*, $\mathsf{AES10}_{\boldsymbol{K}}(\mathbb{X} \oplus \Delta)$, as $(\Delta, I, J, L, I, J, L, I, J, L, I)$ instead of $(0, I, J, L, I, J, L, I, J, L, I)$, where $\Delta = i \cdot L$. With this change, we note that the input value of $\mathsf{AES10}_{\boldsymbol{K}}$ is changed from $\mathbb{X} \oplus \Delta$ to $\mathbb{X}$. Let δ_B be an XOR difference between B and B', which are two blocks at the same location, *i.e.*, $\delta_B = B \oplus B'$.

We first fix a pair $(\mathbb{X}_1, \mathbb{X}'_1)$ and a pair (M_y, M'_y) in such a way that $\delta_{\mathbb{X}_1} = 0$ and $\delta_{M_y} = 0$, respectively. Next, we find a collision on C_y by choosing a proper pair of distinct keys, K and K', in such a way that $(\delta_{\mathbb{X}_1}, \delta_{\mathbb{Y}_1}, \delta_{\mathbb{X}_2}, \delta_{\mathbb{Y}_2}) = (0, 0, 0, 0)$ holds and the number of active S-boxes in this case are minimized. In the following paragraphs, we propose a choice of distinct keys (K, K') so that $(\delta_{\mathbb{X}_1}, \delta_{\mathbb{Y}_1}) = (0, 0)$ holds with a probability of one, whereas $(\delta_{\mathbb{X}_2}, \delta_{\mathbb{Y}_2}) = (0, 0)$ holds with a probability of at least 2^{-28}. Note that we can always get $\delta_{\mathbb{X}_2} = 0$ when $(\delta_{\mathbb{X}_1}, \delta_{\mathbb{Y}_1}) = (0, 0)$ holds since $\delta_{\mathbb{X}_2} = \delta_{\mathbb{Y}_1} \oplus \delta_{M_y}$ and $\delta_{\mathbb{Y}_1} = \delta_{M_y} = 0$. Then we can make the system for the last diblock consistent by exploiting the preimage weakness of AEZ-hash, as described in Sect. 3.2.

According to Alg. 3, the subkeys (I, J, L) are directly derived from the master key K when $|K| = 384$ as $(I \parallel J \parallel L) \leftarrow K$, however, they are generated from the BLAKE2b algorithm when $|K| \neq 384$, as $(I \parallel J \parallel L) \leftarrow \mathsf{BLAKE2b}(K)$. The BLAKE2b algorithm is not invertible. Therefore, we choose the key length as $|K| = 384$, which is the default option, so that we can choose the subkey differences $(\delta_I, \delta_J, \delta_L)$ in our attack.

How to Choose Subkey Differences. Before explaining the details, we discuss how to choose the subkey differences $(\delta_I, \delta_J, \delta_L)$. As described above, we need to choose a proper pair of distinct keys (K, K') in such a way that $(\delta_{\mathbb{X}_1}, \delta_{\mathbb{Y}_1}, \delta_{\mathbb{X}_2}, \delta_{\mathbb{Y}_2}) = (0, 0, 0, 0)$ holds and the number of active S-boxes is minimized. To find such a differential propagation from $\delta_{\mathbb{X}_1}$ to $\delta_{\mathbb{Y}_2}$, we consider a series of computations from $\mathbb{X}_1$ to $\mathbb{Y}_2$ as two consecutive $\mathsf{AES10}_{\boldsymbol{K}}$ functions with round keys $((L, I, J, L, I, J, L, I, J, L, I), (2L, I, J, L, I, J, L, I, J, L, I))$.

Focusing on the round keys of the two consecutive $\mathsf{AES10}_{\boldsymbol{K}}$ functions, the first and last two consecutive round keys are (L, I) or $(2L, I)$. In addition, unlike the original AES, $\mathsf{AES10}_{\boldsymbol{K}}$ contains MixColumns in the last round. We use these properties to analyze a differential propagation of the two consecutive $\mathsf{AES10}_{\boldsymbol{K}}$ functions. Our strategy consists of choosing the differences δ_I, δ_J, and δ_L and the values I, J, L in such a way that the difference that is introduced in one round will be canceled out by the difference in the next round with high probability. Then, there are several rounds where the input and output differences are zero. We choose the round key differences in the following steps:

1. We choose δ_L or δ_{2L} in such a way that these differences activate only one byte of the internal state. In other words, when inserting these differences into the internal state, we need to minimize the number of active S-boxes in the corresponding aesenc function. Note that choosing δ_L uniquely determines δ_{2L}.

2. We choose δ_I in such a way that the differential propagation caused by inserting δ_L or δ_{2L} in the previous round will be canceled with high probability. In other words, inserting this difference into the internal state will lead to a pair of identical internal states (zero difference, no active S-boxes) with high probability.

3. We do not need to consider a difference in J (*i.e.*, $\delta_J = 0$), since we know from Step 2 that the probability that the state difference is already zero is high. Instead, we choose the value J satisfying $\mathbb{X}_1 = \mathbb{X}'_1 = \mathsf{aesenc}(\mathsf{aesenc}(\mathsf{aesenc}(\mathbb{X}_1, L), I), J)$.

Given $\mathbb{X}_1 (= \mathbb{X}'_1)$, $\delta_{\mathbb{Y}_1} = 0$ holds by choosing round key differences and values from the above way. It implies that we do not need to rely on a probabilistic event for a differential propagation of the first $\mathsf{AES10}_{\boldsymbol{K}}$. Instead, we rely on a probabilistic event for a differential propagation of the second $\mathsf{AES10}_{\boldsymbol{K}}$, but it contains only four active S-boxes. Its differential probability is within the range of $2^{-6\times4}$ to $2^{-7\times4}$. Note that $\mathsf{AES10}_{\boldsymbol{K}}$ contains MixColumns in the last round; thus, both $\delta_{\mathbb{Y}_1} = 0$ and $\delta_{\mathbb{Y}_2} = 0$ hold simultaneously with high probability by canceling the differences with δ_I.

Attack Procedure. We execute the following procedure:

Step 1. Pick a pair $(\mathbb{X}_1, \mathbb{X}'_1)$ and a pair (M_y, M'_y) in such a way that $\delta_{\mathbb{X}_1} = 0$ and $\delta_{M_y} = 0$, respectively.

Step 2. Find a proper pair of distinct keys (K, K') in the following procedure, with the way of choosing the round key differences as described above (see Fig. 5 for a better understanding of a differential propagation used in this step):

1. Pick (L, I) randomly, and compute $I' = I \oplus \delta_I$.
2. Compute $\mathsf{aesdec}(\mathsf{aesenc}(\mathbb{X}_1 \oplus L, I) \oplus I', \mathbb{X}'_1)$ to get L', and determine $\delta_L = L \oplus L'$.
3. Select a pair (J, J') for which both $\mathsf{aesenc}(\mathsf{aesenc}(\mathbb{X}_1 \oplus L, I), J) = \mathbb{X}_1$ and $\mathsf{aesenc}(\mathsf{aesenc}(\mathbb{X}'_1 \oplus L', I'), J') = \mathbb{X}'_1$, where $J = J'$ since $\delta_J = 0$.
4. Execute the remaining first $\mathsf{AES10}_{\boldsymbol{K}}$ to get $\delta_{\mathbb{Y}_1}$. If Steps 2 and 3 hold, we know that the same will apply to the remaining rounds because of the repetition of the round subkeys and the states, and $\delta_{\mathbb{Y}_1} = 0$ holds with probability one.
5. Execute the second $\mathsf{AES10}_{\boldsymbol{K}}$ to get $\delta_{\mathbb{Y}_2}$ from $\delta_{\mathbb{X}_2}$, where $\delta_{\mathbb{X}_2} = 0$ always holds since $\delta_{\mathbb{X}_2} = \delta_{\mathbb{Y}_1} \oplus \delta_{M_y}$ and $\delta_{\mathbb{Y}_1} = \delta_{M_y} = 0$. The event that $\delta_{\mathbb{Y}_2} = 0$ is probabilistic. There are four rounds in which the difference introduced by δ_{2L} or δ_L must be canceled by δ_I, and in each one, there is only one active Sbox. The differential probability is within the range of $(2^{-6})^4 = 2^{-24}$ to $(2^{-7})^4 = 2^{-28}$.
6. Get a collision on C_y when $\delta_{\mathbb{Y}_2} = 0$ holds since $\delta_{C_y} = \delta_{\mathbb{X}_1} \oplus \delta_{\mathbb{Y}_2} = 0$, and determine $S = \mathbb{X}_1 \oplus \mathbb{X}_2$ and $S' = \mathbb{X}'_1 \oplus \mathbb{X}'_2$.

Step 3. Pick an arbitrary sequence of diblocks $CC = (CC_1, \ldots, CC_m)$ and C_x, and derive $(Y_1, \ldots, Y_m)$ using K and $(Y'_1, \ldots, Y'_m)$ from CC and K'. Note that $Y = Y_1 \oplus \ldots \oplus Y_m$ and $Y' = Y'_1 \oplus \ldots Y'_m$.

Step 4. Execute Steps 5–10 in Sect. 3.2 to return $((K, A, M), (K', A', M'), C)$.

Complexity Estimation. Following the above attack procedure, the complexity of making Substeps 2-5 of Step 2 work is dominant. The probability that Substeps 2-5 hold is within the range of 2^{-24} to 2^{-28}, therefore, the complexity of the whole attack is estimated as at most 2^{28}.

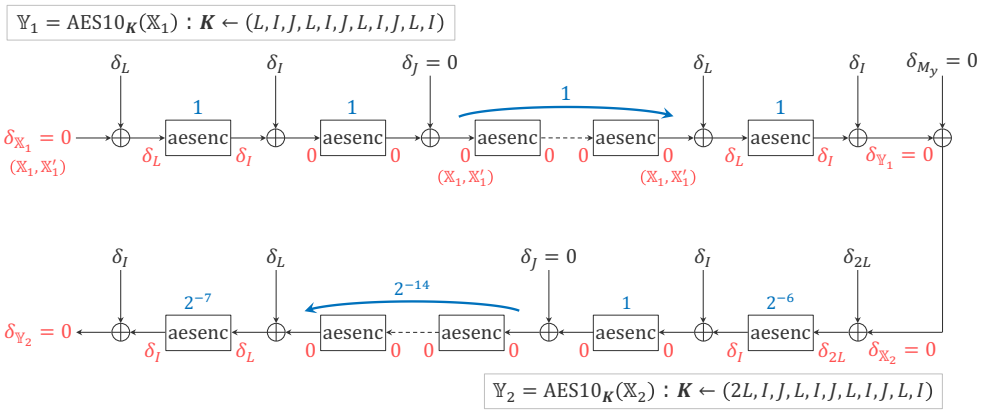

Figure 5: A differential propagation used in the proposed CMT-1 attack against full-spec AEZ. The values in red indicate the expected differences. The values in blue indicate the expected differential probabilities for each 1-round or 7-round aesenc function. The arrows in blue indicate the 7-round aesenc functions.

Experimental Verification. We experimentally verified the validity of the proposed attack. We first search for an appropriate δ_L, where the probability that the differential trail is satisfied is maximum. Clearly, δ_L must have a single active byte. Then, the active column position of δ_I is determined. Moreover, $\mathtt{msb}_1(\delta_L) = 0$. Otherwise, a different byte position is active in δ_{2L}, and it is impossible to cancel the difference by XORing δ_I. Therefore, the choice of δ_L is at most 16×127, and an exhaustive search is possible.

As a result, the best choice of δ_L satisfies the differential trail with a probability of 2^{-27}. An example solution of δ_L, δ_{2L}, δ_I, and δ_J is given as follows:

$$\begin{aligned}
\delta_L &= [\mathtt{01\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00}],\\
\delta_{2L} &= [\mathtt{02\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00}],\\
\delta_I &= [\mathtt{28\ 14\ 14\ 3C\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00}],\\
\delta_J &= [\mathtt{00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00}].
\end{aligned}$$

We provide a specific example of a key committing attack using this differential propagation in Appendix C. To summarize, our experimental verification demonstrates that the proposed attack is feasible with a time complexity of 2^{27}.

Countermeasure. The practical CMT-1 attack against full-spec AEZ exploits the weak key schedule, where the attacker can easily control the value of I, J, and L. One option to counter the practical attack is to use BLAKE2b to generate a 384-bit key even when the key length is 384 bits. Without breaking the hash function BLAKE2b, this makes it hard to choose the master key satisfying the subkey relation.

5 Key Committing Security of Adiantum

This section describes two collision-based CMT-1 attacks on EtE using Adiantum, both for prepending and appending. In both cases, the most expensive step is a collision-finding problem. Thus, both attacks work in the time and memory complexity of the chosen collision algorithm, which is, at best, birthday-bound time complexity. Our attacks show that the key committing security of EtE-Adiantum is at most $O(2^{\tau/2})$ for prepending and $O(2^{n/2})$ for appending. Figure 6 shows the collision-finding problem for each case.

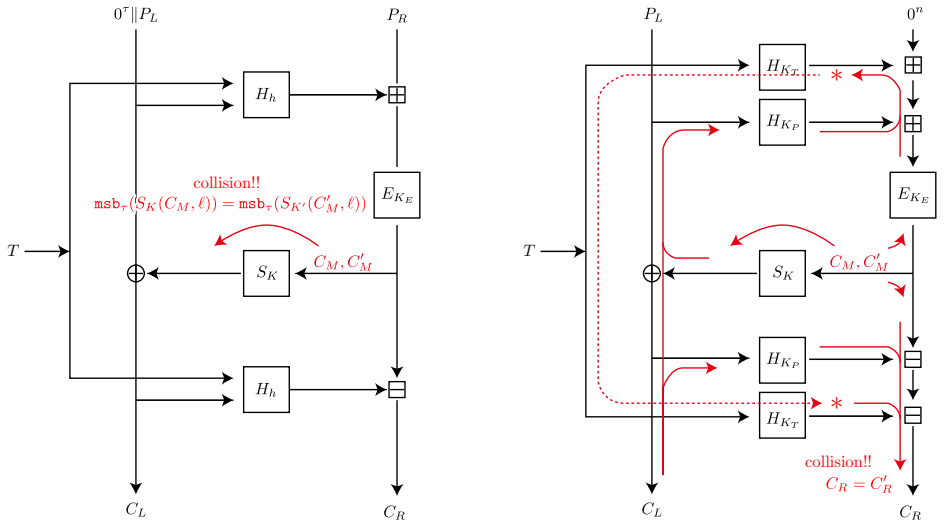

Figure 6: Collision-finding problem for CMT-1 attacks against Adiantum. The left and right figures show the prepending and appending versions, respectively.

Preimages using Hash Function. The attacks described below rely on the efficiency of finding preimages of the hash function given a key. The input of a hash function is typically much larger than its output, so there can be many solutions. The hash used by Adiantum has good properties when using a secret key, but it is easily manipulable in the known-key setting, which allows for our efficient key committing attacks. The Adiantum hash is based on both NH and Poly1305 which allow for straightforward preimages when the key is known.

In particular, steps such as "Find T s.t. $\textsc{Hash}(h, T, a) = b$" for some value h, a, b involve finding a preimage of Poly1305 as:

$$\begin{aligned}\textsc{Hash}(h,T,a) = b &\Leftrightarrow \textsc{HashT}(K_T, |a|, T) \boxplus \textsc{HashP}(K_P, a) = b,\\ &\Leftrightarrow \textsc{HashT}(K_T, |a|, T) = b \boxminus \textsc{HashP}(K_P, a),\\ &\Leftrightarrow \textsc{Poly1305}_{K_T}(|a| \,\|\, T) = b \boxminus \textsc{Poly1305}_{K_L}\left(\mathrm{NH}_{K_N}(\mathrm{pad}(a))\right).\end{aligned}$$

Indeed, given a known masked hash key $\overline{k} \neq 0$, it is trivial to manipulate two or more blocks of T to obtain any 128-bit value. Since $2^{130} - 5$ is prime, $(\overline{k}^{i+1}(T_i + 2^{128}) \mod (2^{130} - 5))$ can have 2^{128} different values, one for every choice of T_i. Then, when reducing to mod 2^{128}, a maximum of 4 possible values might map to a single one. Let us give an easy algorithm: given a target a, arbitrarily choose all blocks of T except T_i for some i, and compute the required value. This works with a probability of at least $1/4$, and this can be repeated by fixing other values for T.

Note that finding a preimage implies a CMT-4 attack (see Sect. 3.1). We can break the CMT-4 notion by constructing a pair of distinct T and T' such that the hash function's outputs are identical.

Goal of CMT-1 Attack. The goal of our attacks is to find a non-trivial solution to the equation:

$$\textsc{Encrypt}(K, T, P_L \,\|\, P_R) = \textsc{Encrypt}(K', T', P'_L \,\|\, P'_R)$$

These can be seen as two equations for the left and right parts of the ciphertext, C_L and C_R, respectively. We further break down these by introducing the intermediate values C_M

and C'_M so that we get four equations:

$$\begin{cases} C_M = E_{K_E}\left(\textsc{Hash}(h, T, P_L) \boxplus P_R\right), \\ C'_M = E_{K'_E}\left(\textsc{Hash}(h', T', P'_L) \boxplus P'_R\right), \\ P_L \oplus S_K(C_M, |P_L|) = P'_L \oplus S_K(C'_M, |P'_L|), & (C_L = C'_L), \\ C_M \boxminus \textsc{Hash}(h, T, C_L) = C'_M \boxminus \textsc{Hash}(h', T', C_L), & (C_R = C'_R). \end{cases} \tag{1}$$

5.1 CMT-1 Attack of Adiantum with 0^τ Prepending

Overview. The attack finds a non-trivial solution to Equation (1) where P_L and P'_L are replaced by $0^\tau \parallel P_L$ and $0^\tau \parallel P'_L$, respectively, with $|0^\tau \parallel P_L| = |0^\tau \parallel P'_L| = \ell$. The left figure of Fig. 6 shows the high-level idea of our attack. We find C_M and C'_M satisfying $\mathtt{msb}_\tau(S_K(C_M, \ell)) = \mathtt{msb}_\tau(S_{K'}(C'_M, \ell))$, which is the collision-finding problem, and we can find such a pair with a complexity of $O(2^{\tau/2})$. Suppose we have such a pair, (C_M, C'_M) with distinct keys, K and K'. Then, because we can easily compute the preimage of the hash function given a key, we can construct (T, P_L, P_R) and (T', P'_L, P'_R) whose ciphertexts are identical using the following procedure.

Attack Procedure.

Step 1. Pick a pair of distinct keys K and K' arbitrarily. Pick P_L and P_R arbitrarily. Compute $K_E \parallel h \leftarrow S_K(\varepsilon, 8832)$ and $K'_E \parallel h' \leftarrow S_{K'}(\varepsilon, 8832)$.

Step 2. Define $f_1(x) \leftarrow \mathtt{msb}_\tau(S_K(x, \ell))$ and $f_2(x) \leftarrow \mathtt{msb}_\tau(S_{K'}(x, \ell))$. Find a collision such that $f_1(C_M) = f_2(C'_M)$.

Step 3. Let $C_L \leftarrow (0^\tau \parallel P_L) \oplus S_K(C_M, \ell)$. Then, let $0^\tau \parallel P'_L \leftarrow C_L \oplus S_{K'}(C'_M, \ell)$, where matching 0^τ is guaranteed by the collision, $f_1(C_M) = f_2(C'_M)$.

Step 4. Find T s.t. $\textsc{Hash}(h, T, 0 \parallel P_L) = E^{-1}_{K_E}(C_M) \boxminus P_R$.

Step 5. Determine $C_R \leftarrow C_M \boxminus \textsc{Hash}(h, T, C_L)$.

Step 6. Find T' s.t. $\textsc{Hash}(h', T', C_L) = C'_M \boxminus C_R$.

Step 7. Determine $P'_R \leftarrow E^{-1}_{K'_E}(C'_M) \boxminus \textsc{Hash}(h', T', 0 \parallel P'_L)$.

Step 8. Return $((K, T, 0^\tau \parallel P_L \parallel P_R), (K', T', 0^\tau \parallel P'_L \parallel P'_R), (C_L, C_R))$.

5.2 CMT-1 Attack of Adiantum with 0^n Appending

Overview. The attack finds a non-trivial solution to Equation (1) where P_R and P'_R are replaced by 0^n. The right figure of Fig. 6 shows the high-level idea of our attack. We exploit the following property of the hash function.

$$\begin{aligned} & \textsc{Hash}(h, T, P_L) = E^{-1}_{K_E}(C_M), \\ \Longrightarrow\ & \textsc{HashT}(K_T, \ell, T) \boxplus \textsc{HashP}(K_P, P_L) = E^{-1}_{K_E}(C_M), \\ \Longrightarrow\ & \textsc{HashT}(K_T, \ell, T) = E^{-1}_{K_E}(C_M) \boxminus \textsc{HashP}(K_P, P_L), \\ \Longrightarrow\ & \textsc{Hash}(h, T, C_L) = E^{-1}_{K_E}(C_M) \boxminus \textsc{HashP}(K_P, P_L) \boxplus \textsc{HashP}(K_P, C_L). \end{aligned}$$

As shown in Fig. 6, C_R is computed from C_M and C_L as follows:

$$C_R = C_M \boxminus \textsc{HashP}(K_P, C_L) \boxminus \left(E^{-1}_{K_E}(C_M) \boxminus \textsc{HashP}(K_P, C_L \oplus S_K(C_M, \ell))\right).$$

Therefore, given an arbitrary C_L, we find C_M and C'_M satisfying $C_R = C'_R$, which is the collision-finding problem, and we can find such a pair with a complexity of $O(2^{n/2})$. Once we find such a pair, (C_M, C'_M) with distinct keys K and K', we can construct (T, P_L, P_R) and (T', P'_L, P'_R) whose ciphertexts are identical using the following procedure because it is easy to find a preimage of the hash function given a key.

Attack Procedure.

Step 1. Pick a pair of distinct keys K and K' arbitrarily. Pick C_L arbitrarily and $\ell \leftarrow |C_L|$. Compute $K_E \| h \leftarrow S_K(\varepsilon, 8832)$ and $K'_E \| h' \leftarrow S_{K'}(\varepsilon, 8832)$. Moreover, $K_T \| K_P \leftarrow h$ and $K'_T \| K'_P \leftarrow h'$.

Step 2. Define $f_1(x) \leftarrow x \boxminus E^{-1}_{K_E}(x) \boxplus \textsc{HashP}(K_P, C_L \oplus \mathtt{msb}_\ell(S_K(x, \ell))) \boxminus \textsc{HashP}(K_P, C_L)$ and $f_2(x) \leftarrow x \boxminus E^{-1}_{K'_E}(x) \boxplus \textsc{HashP}(K'_P, C_L \oplus \mathtt{msb}_\ell(S_{K'}(x, \ell))) \boxminus \textsc{HashP}(K'_P, C_L)$. Find a collision such that $f_1(C_M) = f_2(C'_M)$. Then, determine $C_R \leftarrow f_1(C_M)$.

Step 3. Determine $P_L \leftarrow C_L \oplus S_K(C_M, \ell)$ and $P'_L \leftarrow C_L \oplus S_{K'}(C'_M, \ell)$.

Step 4. Find T s.t. $\textsc{Hash}(h, T, P_L) = E^{-1}_{K_E}(C_M)$. Find T' s.t. $\textsc{Hash}(h', T', P'_L) = E^{-1}_{K'_E}(C'_M)$.

Step 5. Return $((K, T, P_L \,\|\, 0^n), (K', T', P'_L \,\|\, 0^n), (C_L, C_R))$.

6 Key Committing Security of HCTR2

This section describes two CMT-1 attacks on authenticated encryption using HCTR2, one for the prepend version and one for the append version. These attacks have much in common with the ones of Sect. 5 on Adiantum. Note that the roles of the left and right branches are reversed compared to Adiantum. The 0^τ appending and 0^n prepending attacks against HCTR2 correspond to 0^τ prepending and 0^n appending attacks against Adiantum, respectively. Just as in the attack on Adiantum, the most expensive step in both cases is a collision-finding problem. Thus, both attacks work with the time and memory complexity of the chosen collision algorithm, which is, at best, birthday-bound time complexity. Therefore, the key committing security of HCTR2 is at most $O(2^{\tau/2})$ and $O(2^{n/2})$ for the 0^τ appending and 0^n prepending versions, respectively. Figure 7 shows the collision-finding problem for each case.

Preimages using Hash Function. The attacks described below rely on the efficiency of finding preimages of the hash function given a key. The input of a hash function is typically much larger than its output, so there can be many solutions. The hash used by HCTR2 has good properties when using a secret key, but it is easily manipulable in the known-key setting, which allows for our efficient key committing attacks.

In particular, steps such as "Find T s.t. $\textsc{Hash}(h, T, a) = b$" for some value h, a, b involve finding a preimage of $\textsc{Poly}$ as:

$$\begin{aligned}\textsc{Hash}(h, T, a) = b &\Leftrightarrow (2|T| + 2) \cdot h^{\frac{|T|+|a|}{n}+1} \oplus \textsc{Poly}_h(T) \cdot h^{\frac{|a|}{n}+1} \oplus \textsc{Poly}_h(a) \cdot h = b, \\ &\Leftrightarrow \textsc{Poly}_h(T) = \left(b \oplus (2|T| + 2) \cdot h^{\frac{|T|+|a|}{n}+1} \oplus \textsc{Poly}_h(a) \cdot h\right) \cdot h^{-\frac{|a|}{n}+1},\end{aligned}$$

where $h \neq 0$ is known, and the right-hand side can be computed down to a single value. Finding this preimage is easy. For instance, one can arbitrarily set every block of T except one and compute the required value for the remaining block by simple algebra.

Again, similarly to Adiantum, we can easily break CMT-4 by constructing a pair of distinct T and T' such that the hash function's outputs are identical.

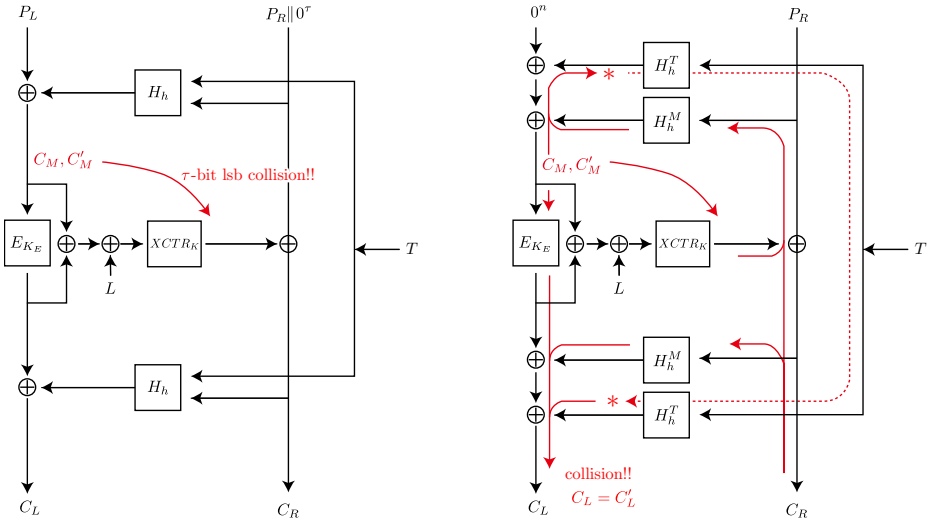

Figure 7: Collision-finding problem for CMT-1 attacks for HCTR2. The left and right figures show the appending and prepending versions, respectively.

Goal of CMT-1 Attack. The goal of our attacks is to find a non-trivial solution to the equation:

$$\textsc{Encrypt}(K, T, P_L \parallel P_R) = \textsc{Encrypt}(K', T', P'_L \parallel P'_R).$$

These can be seen as two equations for the left and right parts of the ciphertext, C_L and C_R, respectively. We further break down these by introducing the intermediate values C_M and C'_M so that we get four equations where h and L are derived from K (h' and L' from K'):

$$\begin{cases} C_M = P_L \oplus \textsc{Hash}(h, T, P_R), \\ C'_M = P'_L \oplus \textsc{Hash}(h', T', P'_R), \\ P_R \oplus \mathrm{XCTR}(K, L \oplus C_M \oplus E_K(C_M), |P_R|), & (2) \\ \quad = P'_R \oplus \mathrm{XCTR}(K', L' \oplus C'_M \oplus E_{K'}(C'_M), |P'_R|), & (C_R = C'_R), \\ E_K(C_M) \oplus \textsc{Hash}(h, T, C_R) = E_{K'}(C'_M) \oplus \textsc{Hash}(h', T', C_R), & (C_L = C'_L). \end{cases}$$

6.1 CMT-1 Attack of HCTR2 with 0^τ Appending

Overview. The attack finds a non-trivial solution to Equation (2) where P_R and P'_R are replaced by $0^\tau \parallel P_R$ and $0^\tau \parallel P'_R$, respectively, with $|0^\tau \parallel P_R| = |0^\tau \parallel P'_R| = \ell$. The left figure of Fig. 7 shows the high-level idea of our attack. We find C_M and C'_M satisfying

$$\mathtt{lsb}_\tau(\mathrm{XCTR}(K, L \oplus C_M \oplus E_K(C_M), \ell)) = \mathtt{lsb}_\tau(\mathrm{XCTR}(K', L' \oplus C'_M \oplus E_{K'}(C'_M), \ell)),$$

which is the collision-finding problem, and we can detect such a pair with a complexity of $O(2^{\tau/2})$. Suppose we have such a pair, (C_M, C'_M) with distinct keys, K and K'. Then, because we can easily compute the preimage of the hash function given a key, we can construct (T, P_L, P_R) and (T', P'_L, P'_R) whose ciphertexts are identical as follows.

Attack Procedure.

Step 1. Pick a pair of distinct keys K and K' arbitrarily. Pick P_L and P_R arbitrarily. Compute $h \leftarrow E_K(0)$ and $h' \leftarrow E_{K'}(0)$. Compute $L \leftarrow E_K(1)$ and $L' \leftarrow E_{K'}(1)$. Set $\ell \leftarrow |P_R| + \tau$.

Step 2. Define $f_1(x) \leftarrow \mathtt{lsb}_\tau(\mathrm{XCTR}(K, L \oplus x \oplus E_K(x), \ell))$. Then, define $f_2(x) \leftarrow \mathtt{lsb}_\tau(\mathrm{XCTR}(K', L' \oplus x \oplus E_{K'}(x), \ell))$. Find a collision such that $f_1(C_M) = f_2(C'_M)$.

Step 3. Determine $C_R \leftarrow (P_R \,\|\, 0^\tau) \oplus \mathrm{XCTR}(K, L \oplus C_M \oplus E_K(C_M), \ell)$. Then, determine $P'_R \| 0^\tau \leftarrow C_R \oplus \mathrm{XCTR}(K', L' \oplus C'_M \oplus E_{K'}(C'_M), \ell)$, where matching 0^τ is guaranteed by the collision, $f_1(C_M) = f_2(C'_M)$.

Step 4. Find T s.t. $\textsc{Hash}(h, T, P_R \,\|\, 0^\tau) = C_M \oplus P_L$.

Step 5. Determine $C_L \leftarrow \textsc{Hash}(h, T, C_R) \oplus E_K(C_M)$.

Step 6. Find T' s.t. $\textsc{Hash}(h', T', C_R) = E_{K'}(C'_M) \oplus C_L$.

Step 7. Determine $P'_L \leftarrow \textsc{Hash}(h', T', P'_R \,\|\, 0^\tau) \oplus C'_M$.

Step 8. Return $((K, T, P_L \,\|\, P_R \,\|\, 0^\tau), (K', T', P'_L \,\|\, P'_R \,\|\, 0^\tau), (C_L, C_R))$.

6.2 CMT-1 Attack of HCTR2 with 0^n Prepending

Overview. The attack finds a non-trivial solution to Equation (2) where P_L and P'_L are replaced by 0^n. The right figure of Fig. 7 shows the high-level idea of our attack. We exploit the following property of the hash function.

$$
\begin{aligned}
&\textsc{Hash}(h, T, P_R) = C_M, \\
\Longrightarrow\ &(2|T| + 2) \cdot h^{\frac{|T|+\ell}{n}+1} \oplus \textsc{Poly}_h(T) \cdot h^{\frac{\ell}{n}+1} \oplus \textsc{Poly}_h(P_R) \cdot h = C_M, \\
\Longrightarrow\ &(2|T| + 2) \cdot h^{\frac{|T|+\ell}{n}+1} \oplus \textsc{Poly}_h(T) \cdot h^{\frac{\ell}{n}+1} = C_M \oplus \textsc{Poly}_h(P_R) \cdot h, \\
\Longrightarrow\ &\textsc{Hash}(h, T, C_R) = C_M \oplus \textsc{Poly}_h(P_R) \cdot h \oplus \textsc{Poly}_h(C_R) \cdot h, \\
\Longrightarrow\ &\textsc{Hash}(h, T, C_R) = C_M \oplus \textsc{Poly}_h(P_R \oplus C_R) \cdot h, \\
\Longrightarrow\ &\textsc{Hash}(h, T, C_R) = C_M \oplus \textsc{Poly}_h(\mathrm{XCTR}(K, L \oplus C_M \oplus E_K(C_M), \ell)) \cdot h.
\end{aligned}
$$

As shown in Fig. 7, C_L is computed from C_M as follows:

$$C_L = \textsc{Poly}_h(\mathrm{XCTR}(K, L \oplus C_M \oplus E_K(C_M), \ell)) \cdot h \oplus C_M \oplus E_K(C_M).$$

Note that C_R is not used here because the hash function output related to C_R is canceled out. Therefore, we find C_M and C'_M satisfying $C_L = C'_L$, which is the collision-finding problem, and we can find such a pair with a complexity of $O(2^{n/2})$. Once we find such a pair, (C_M, C'_M) with distinct keys K and K', we can construct (T, P_L, P_R) and (T', P'_L, P'_R) whose ciphertexts are identical using the following procedure because it is easy to find a preimage of the hash function given a key.

Attack Procedure.

Step 1. Pick a pair of distinct keys K and K'. Pick C_R arbitrarily. Compute $h \leftarrow E_K(0)$ and $h' \leftarrow E_{K'}(0)$. Compute $L \leftarrow E_K(1)$ and $L' \leftarrow E_{K'}(1)$. Set $\ell \leftarrow |P_R|$.

Step 2. Define $f_1(x) \leftarrow \textsc{Poly}_h(\mathrm{XCTR}(K, L \oplus x \oplus E_K(x), \ell)) \cdot h \oplus x \oplus E_K(x)$ and $f_2(x) \leftarrow \textsc{Poly}_{h'}(\mathrm{XCTR}(K', L' \oplus x \oplus E_{K'}(x), \ell)) \cdot h' \oplus x \oplus E_{K'}(x)$. Find a collision such that $f_1(C_M) = f_2(C'_M)$. Then, determine $C_L = f_1(C_M)$.

Step 3. Determine $P_R \leftarrow C_R \oplus \mathrm{XCTR}(K, L \oplus C_M \oplus E_K(C_M), \ell)$ and $P'_R \leftarrow C_R \oplus \mathrm{XCTR}(K', L' \oplus C'_M \oplus E_{K'}(C'_M), \ell)$.

Step 4. Find T s.t. $\textsc{Hash}(h, T, P_R) = C_M$. Find T' s.t. $\textsc{Hash}(h', T', P'_R) = C'_M$.

Step 5. Return $((K, T, 0^n \,\|\, P_R), (K', T', 0^n \,\|\, P'_R), (C_L, C_R))$.

7 Security Proofs of General AEZ and Adiantum with Prepending

We present provable security results for general AEZ with $\tau = n$ and Adiantum with the 0^τ prepending case, assuming their primitives are ideal. We prove birthday-type security bounds for them, which indicates the tightness of our attacks for these cases. To our knowledge, they are the first results to show meaningful key committing security for the existing non-monolithic EtE schemes. We start with a definition of multi-collision resistance, which will be used in our proofs.

Multi-collision Resistance. Let $H\colon \mathcal{X} \to \mathcal{Y}$ be a function. Let $s \geq 2$ be an integer. An s-way collision for H is a tuple $(X_1, \ldots, X_s)$ of distinct points in $\mathcal{X}$ such that $H(X_1) = \cdots = H(X_s)$. For an adversary $\mathcal{A}$, define its advantage in breaking the s-way multi-collision resistance of H built on a primitive Π as

$$\mathbf{Adv}^{\mathsf{Coll}}_{H,s}(\mathcal{A}) = \Pr[(X_1, \ldots, X_s) \leftarrow \mathcal{A}^{\Pi^{\pm}} : X_1 \neq \ldots \neq X_s, H(X_1) = \cdots = H(X_s)]$$

where the probability is over $(X_1, \ldots, X_s) \xleftarrow{\$} \mathcal{A}$. Here Π is an ideal primitive which can be a (tweakable) block cipher or permutation, and $\pm$ means that the adversary is getting both forward and inverse access to this primitive. When $s = 2$, we recover the classical notion of collision resistance and simply write $\mathbf{Adv}^{\mathsf{Coll}}_{H,s}(\mathcal{A})$ as $\mathbf{Adv}^{\mathsf{Coll}}_{H}(\mathcal{A})$.

7.1 Security Proof of general AEZ

We prove the CMT-1 security of general AEZ (gAEZ) when $\tau = n$.

Theorem 1.

$$\mathbf{Adv}^{\mathsf{CMT\text{-}1}}_{\mathsf{gAEZ}}(\mathcal{A}) \leq \frac{p(p-1)}{2^n},$$

where p is the maximum number of queries by $\mathcal{A}$ to the underlying ideal tweakable cipher E.

Proof strategy. The CMT-1 security can be seen as the collision probability of the whole ciphertext. We prove that this probability is small by showing that the collision probability of a *part* of the ciphertext is sufficiently small by focusing on C_y, the last ciphertext block.

Proof. For distinct input tuples of gAEZ, (K, A, M), $(K', A', M') \in \{0,1\}^k \times \{0,1\}^* \times \{0,1\}^*$ such that $K \neq K'$, define $C_y = \mathtt{lsb}_n(\mathsf{gAEZ}_K(A, M))$, $C'_y = \mathtt{lsb}_n(\mathsf{gAEZ}_{K'}(A', M'))$. We obtain

$$\mathbf{Adv}^{\mathsf{CMT\text{-}1}}_{\mathsf{gAEZ}}(\mathcal{A}) \leq \Pr[\mathcal{A}' \to ((K, A, M), (K', A', M')) \text{ s.t. } K \neq K', C_y = C'_y)].$$

We extract a function outputting C_y from gAEZ and define it as $\mathsf{FPTP2'} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Thus, we obtain[4]

$$C_y = \mathsf{FPTP2'}_K(Z) \coloneqq E_K^{-1,2}(E_K^{-1,1}(Z)) \oplus Z,$$

where $Z = M_x \oplus X \oplus \Delta \oplus E_K^{0,1}(0^n)$. Then we obtain

$$\Pr[\mathcal{A} \to ((K, A, M), (K', A', M')) \text{ s.t. } K \neq K', C_y = C'_y)] \leq \mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{FPTP2'}}(\mathcal{A}'), \quad (3)$$

[4]The name FPTP2' comes from the resemblance to the Feed-forward Permutation-Tweak-Permutation proposed by Chen and Tessaro [CT21].

for some $\mathcal{A}'$ who uses p queries to the primitive. Here, $\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{FPTP2'}}(\mathcal{A}) \coloneqq \Pr[\mathcal{A} \to ((K,Z),(K',Z'))$ s.t. $(K,Z) \neq (K',Z'), \mathsf{FPTP2'}_K(Z) = \mathsf{FPTP2'}_{K'}(Z')]$. In the game Coll, $\mathcal{A}$ has access to $E^{-1,1}$ and $E^{-1,2}$, which are modeled as (tweakable) ideal ciphers.

Next, we evaluate $\max_{\mathcal{A}'}\left(\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{FPTP2'}}(\mathcal{A}')\right)$. FPTP2' is the DM construction with double encryption and the proof is almost the same as the original DM's proof (just taking into account the structure of the double encryption). Without loss of generality, assume that an adversary makes no repeated queries. In this evaluation, we permit an adversary $\mathcal{A}'$ to obtain the following additional responses of the underlying primitives. In the additional setting, if there is an input-output tuple of $E^{-1,i}_{K_1}$ where $i \in \{1,2\}$, then an additional input-output tuple of the other primitive is defined by following the structure of FPTP2'.

- For a forward query (K_1, X_1) to the forward oracle of $E^{-1,1}_{K_1}$ or an inverse query (K_1, Y_1) to the inverse oracle of $E^{-1,1}_{K_1}$,
  1. the query-response tuple (K_1, X_1, Y_1) is defined by $E^{-1,1}_{K_1}$,
  2. the additional query (K_1, Y_1) is made to $E^{-1,2}_{K_1}$ and then the query-response tuple (K_1, Y_1, Z_1) is defined,
  3. the two responses Y_1 (resp., X_1) and Z_1 are returned to the adversary making the forward query (resp., the inverse query).
- For a forward query (K_2, Y_2) to the forward oracle of $E^{-1,2}_{K_2}$ or an inverse query (K_2, Z_2) to the inverse oracle of $E^{-1,2}_{K_2}$,
  1. the query-response tuple (K_2, Y_2, Z_2) is defined,
  2. the additional query (K_2, Y_2) is made to the inverse oracle of $E^{-1,1}_{K_2}$ and then the query-response tuple (K_2, X_2, Y_2) is defined,
  3. the two responses X_2 and Z_2 (resp., Y_2) are returned to the adversary making the forward query (resp., the inverse query).

The additional queries ensure that for each query made by the adversary, one input-output tuple of FPTP2' is defined: $(K_1, X_1, X_1 \oplus Z_1)$ by the query to $E^{-1,1}_{K_1}$ or $(K_2, X_2, X_2 \oplus Z_2)$ by the query to $E^{-1,2}_{K_1}$. Regarding the tuple $(K_1, X_1, X_1 \oplus Z_1)$ that is defined by the i-th query, at least one of X_1 or Z_1 is chosen uniformly at random from at least $2^n - p$ elements in $\{0,1\}^n$, thus the probability that $X_1 \oplus Z_1$ collides with one of the previous outputs of FPTP2' is at most $\frac{i-1}{2^n-p}$. The same evaluation holds for the tuple $(K_2, X_2, X_2 \oplus Z_2)$. Summing the bound for each i, we have

$$\max_{\mathcal{A}'}\left(\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{FPTP2'}}(\mathcal{A}')\right) \leq \sum_{i=1}^{p} \frac{i-1}{2^n-p} = \frac{0.5p(p-1)}{2^n-p} \leq \frac{p(p-1)}{2^n} \ , \text{ assuming } p \leq 2^{n-1}.$$

Remark. Inequality (3) does not hold when evaluating CMT-4 security. In the case of CMT-1 security, $K \neq K'$ necessarily holds (the left side of the inequation), and thus $(K,Z) \neq (K',Z')$ holds, which is required for the definition of $\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{FPTP2'}}(\mathcal{A}')$. However, in the case of CMT-4 security, $(K,Z) \neq (K',Z')$ in $\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{FPTP2'}}(\mathcal{A}')$ does not necessarily hold. For example, as we showed in Sect. 3.1, the CMT-4 adversary can exploit two distinct AD values to collide Δ. In such a case, $(K,Z) = (K',Z')$ holds, and we cannot take an upper bound of collision probability of C_y with that of FPTP2'.

7.2 Security Proof of Adiantum with Prepending

Let $c = 512$. In this section, we use $X[i..j]$ to denote the i-th to j-th bit of X and $X^*[i..j]$ to denote the remainder of X by truncating the i-th to j-th bit of X. We here prove

the CMT-1 security of Adiantum with 0^τ prepending, assuming the c-bit permutation underlying XChaCha12 (*i.e.*, c-bit 12-round ChaCha permutation) is ideal. We do not change other components. We denote it by Adiantum-pre. Note that when $\tau = n$, we obtain birthday-bound security. The first c-bit output block of XChaCha12 taking 128-bit input x with 256-bit key K is denoted as

$$S_K(x, c) \coloneqq \pi(\mathtt{cst}_1 \parallel L \parallel \mathtt{cst}_2) +_{32} (\mathtt{cst}_1 \parallel L \parallel \mathtt{cst}_2),$$

with 256-bit $L = \pi(\mathtt{cst}_1 \parallel K \parallel x)^*[c/4 + 1..3c/4]$, where $\pi : \{0,1\}^c \to \{0,1\}^c$ is an ideal permutation. Here, $+_{32}$ refers to 16 additions of 32-bit words, $\mathtt{cst}_1$ is the 128-bit XChaCha12 constant and $\mathtt{cst}_2$ is another 128-bit constant that is equal to $\mathrm{ctr} \parallel \mathrm{NUL}^{(4)} \parallel 1 \parallel 0^{63}$, where $\mathrm{ctr} = 0^{31} \parallel 1$, and $\mathrm{NUL}^{(4)}$ is the concatenation of four constant bytes. This is because XChaCha12 uses the first 128 bits of its input for subkey derivation, while the remaining 64 bits are used together with the subkey to derive the final output. In the case of Adiantum, the 128-bit input is padded with a one followed by 63 zeroes to generate the 192-bit XChaCha12 input.

Theorem 2.

$$\mathbf{Adv}^{\mathsf{CMT\text{-}1}}_{\mathsf{Adiantum\text{-}pre}}(\mathcal{A}) \le \frac{2p}{2^{c/4}} + \frac{p(p-1)}{2^{c/2}} + \frac{p(p-1)}{2^\tau},$$

where p is the maximum number of queries to π.

Note that in our case, $c = 512$ and $n = 128$. Hence, the first term of the security bound is negligible.

We need the following result for our security proof on the permutation-based variant of truncated Davies-Meyer (DM) construction. In particular, let $m \le n$ be an integer. The truncated DM based on a permutation $\pi\colon \{0,1\}^c \to \{0,1\}^c$ is defined as $\mathsf{pDM}_{\pi,m}(X) \coloneqq (\pi(X) +_{32} X)[1..m]$. We write pDM_π for the special case $m = c$ (meaning there is no truncation).

Lemma 1 (ω-way collision of permutation-based Davies-Meyer)**.** *Let $\pi\colon \{0,1\}^c \to \{0,1\}^c$ be an ideal permutation. Let $\omega \ge 2$ and $m \le c$ be integers. For an adversary $\mathcal{A}$ that makes at most $p \le 2^{n-1} - \omega$ ideal permutation queries, we have*

$$\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{pDM}_{\pi,\omega}}(\mathcal{A}) \le \binom{p}{\omega} 2^{(1-m)(\omega-1)}.$$

The proof of Lemma 1 can be found in Appendix A and is similar to the one of the block cipher-based DM given in [BH22]. Note that the same bound holds when $+_{32}$ is replaced by $\oplus$.

Proof strategy. As in the case of general AEZ, we prove the CMT-1 security by showing that the collision probability of a *part* of the ciphertext is sufficiently small. Focusing on $\mathtt{msb}_\tau(C_L)$, we can see that this value is almost the same as the output of the permutation-based DM construction if the subkey L is always fresh. The subkey is calculated by evaluating S_K for the input $x = E_{K_E}(P_R \boxplus \textsc{Hash}(h, T, 0^\tau \parallel P_L))$. See the left figure of Fig. 6.

Proof. For distinct input tuples of Adiantum-pre, (K, T, P) and (K', T', P') such that $K \ne K'$, let us define $C = (C_L \parallel C_R) = \mathsf{Adiantum\text{-}pre}_K(T, P)$, $C' = (C'_L \parallel C'_R) = \mathsf{Adiantum\text{-}pre}_K(T', P')$, $C_f = \mathtt{msb}_\tau(C_L)$, and $C'_f = \mathtt{msb}_\tau(C'_L)$. We obtain

$$\mathbf{Adv}^{\mathsf{CMT\text{-}1}}_{\mathsf{Adiantum\text{-}pre}}(\mathcal{A}) \le \Pr[\mathcal{A}' \to ((K, P), (K', P')) \ \text{ s.t. } \ K \ne K', C_f = C'_f].$$

We extract a function outputting C_f from Adiantum and define it as $\mathsf{tpDM} : \{0,1\}^n \to \{0,1\}^\tau$. Thus, we obtain

$$C_f = \mathsf{tpDM}_K(C_M) \coloneqq \mathtt{msb}_\tau(S_K(C_M, c)),$$

where $C_M = E_{K_E}(P_R \boxplus \textsc{Hash}(h, T, 0^\tau \parallel P_L))$. Then, we obtain the following inequation.

$$\Pr[\mathcal{A} \to ((K,T,P),(K',T',P')) \text{ s.t. } K \neq K', C_f = C'_f] \leq \mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{tpDM}}(\mathcal{A}'),$$

for some $\mathcal{A}'$ who uses p queries to the primitive. Here, $\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{tpDM}}(\mathcal{A}) \coloneqq \Pr[\mathcal{A} \to ((K,C_M),(K',C'_M)) \text{ s.t. } (K,C_M) \neq (K',C'_M), \mathsf{tpDM}_K(C_M) = \mathsf{tpDM}_{K'}(C'_M)]$. In the game Coll, $\mathcal{A}$ has both forward and backward (inverse) access to the ideal permutation π. Next, we evaluate $\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{tpDM}}(\mathcal{A}')$, maximized over all possible $\mathcal{A}'$. Without loss of generality, assume that an adversary makes no repeated query. In this evaluation, we permit an adversary $\mathcal{A}'$ to obtain the following additional responses of the underlying primitive. For a forward query X to π or an inverse query Y to π^{-1},

1. the query-response tuple (X, Y) is defined by π which is returned to the adversary,
2. the additional forward query $(\mathtt{cst}_1 \parallel Y^*[c/4+1..3c/4] \parallel \mathtt{cst}_2)$ is made to π and then the query-response tuple $((\mathtt{cst}_1 \parallel Y^*[c/4+1..3c/4] \parallel \mathtt{cst}_2), Z)$ is defined,
3. the two responses Y (resp., X) and Z are returned to the adversary making the forward query (resp., the inverse query).

The additional queries ensure that for each query made by the adversary, one input-output tuple of tpDM is defined.

Let $s \in \{+,-\}$ denote the direction of the primitive queries; $+$ for forward and $-$ for backward. We define the following events: for any distinct $(i, X_i, Y_i, s_i), (j, X_j, Y_j, s_j)$ tuples in the transcript such that one of the following conditions hold:

$$\mathbf{Bad}_1 \colon [s_i = s_j = +] \ \wedge \ \left[Y_i^*[c/4+1..3c/4] = Y_j^*[c/4+1..3c/4]\right],$$
$$\mathbf{Bad}_2 \colon [s_i = s_j = -] \ \wedge \ \left[X_i^*[1..c/4] = X_j^*[1..c/4]\right].$$

Let $\mathbf{Bad} = \mathbf{Bad}_1 \vee \mathbf{Bad}_2$. Next, we show that

$$\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{tpDM}}(\mathcal{A}') \leq \Pr[\mathbf{Bad}] + \mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{pDM}}(\mathcal{B}).$$

Here, the probability where $\mathcal{B}$ is determined by $\mathcal{A}'$ and uses total p queries to the primitive. What is now left is to bound the two probabilities. We will first bound the probability of the event $\mathbf{Bad}$. We first start with $\mathbf{Bad}_1$. For the i-th forward entry (X_i, Y_i), given X_i and all prior queries/answers before the i-th query, Y_i is uniformly distributed over a set of at least $2^c - p \geq 2^{c-1}$ values, and thus the conditional probability that $Y_i^*[c/4+1..3c/4] = Y_j^*[c/4+1..3c/4]$ (for $j < i$) is at most $2^{c/2}/2^{c-1} = 2^{1-c/2}$. We have

$$\Pr[\mathbf{Bad}_1] \leq \sum_{i=1}^{p} \frac{i-1}{2^{c/2-1}} = \frac{0.5p(p-1)}{2^{c/2-1}} \leq \frac{p(p-1)}{2^{c/2}}.$$

Now we consider $\mathbf{Bad}_2$. For the i-th backward entry is (X_i, Y_i), given Y_i and all prior queries/answers before the i-th query, X_i is uniformly distributed over a set of at least $2^c - p \geq 2^{c-1}$ values, and thus the conditional probability that $X_i^*[1..c/4] = X_j^*[1..c/4]$ (for $j < i$) is at most $2^{3c/4}/2^{c-1} = 2^{1-c/4}$. We have

$$\Pr[\mathbf{Bad}_2] \leq \frac{2p}{2^{c/4}}.$$

Together, we have

$$\Pr[\mathbf{Bad}] \leq \frac{p(p-1)}{2^{c/2}} + \frac{2p}{2^{c/4}}. \tag{4}$$

From Lemma 1 by setting $m = \tau$ and $s = 2$, we obtain

$$\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{pDM}}(\mathcal{B}) \leq \frac{p(p-1)}{2^{\tau}}, \tag{5}$$

assuming $p \leq 2^{n-1}$. The theorem is proven by combining Equations (4) and (5). □

8 Conclusions and Future Work

We have studied the key committing security of encode-then-encipher schemes built on wide block ciphers. Taking three well-known schemes as our targets, we have shown several new results, both from the attack and the provable security sides.

Admittedly, our analysis is not comprehensive, in particular for the provable security side. The missing cases are left as future work. One reason for this is while conducting our research, we realized that a small detail that has little impact on the standard model security can significantly impact the key committing security. This can greatly increase the difficulty of analysis from both the attack and the proof sides. For example, our CMT-1 attack against Adiantum with 0^n appending can be greatly sped up if we replace the modular additions of the NH hash function with simple XORs. Such a function would still be a universal hash, but it now allows a very efficient attack. Therefore, it is impossible to prove the key committing security of Adiantum with 0^n appending up to birthday bound using only the fact that NH is a universal hash function. Conversely, HCTR2 uses XCTR which cannot be idealized as a PRF unlike in Adiantum where our proof idealizes XChaCha as a PRF. Indeed, HCTR2 produces a counter N for XCTR based on the same block cipher as XCTR itself. Thus, the adversary can manipulate and control N setting the same input to the block cipher and forcing it to repeat even if the block cipher itself is idealized. That is why we judged that it is challenging to prove birthday-bound security. In other words, proving key committing security can be difficult for schemes that are not designed for that purpose. Nonetheless, we think our research will give insights and help further analysis of the key committing security of EtE schemes and WBCs.

References

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 105–125. Springer, Heidelberg, December 2014.

[ADG+22] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to abuse and fix authenticated encryption without key commitment. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 3291–3308. USENIX Association, August 2022.

[AES23] Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197-upd1, U.S. Department of Commerce, November, 2001; Updated May, 2023.

[And23] Android Open Source Project. File-based encryption. https://source.android.com/docs/security/features/encryption/file-based, 2023.

[ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 119–135. Springer, Heidelberg, June 2013.

[BDD+17] Xavier Bonnetain, Patrick Derbez, Sébastien Duval, Jérémy Jean, Gaëtan Leurent, Brice Minaud, and Valentin Suder. An easy attack on AEZ. Rump session talk at FSE 2017, 2017.

[Ber05] Daniel J. Bernstein. The poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of *LNCS*, pages 32–49. Springer, Heidelberg, February 2005.

[Ber08] Daniel J. Bernstein. ChaCha, a variant of Salsa20. https://cr.yp.to/chacha/chacha-20080128.pdf, January 2008.

[Ber19] Daniel J. Bernstein. Caesar: Competition for authenticated encryption: Security, applicability, and robustness. https://competitions.cr.yp.to/caesar.html, 2019.

[BH22] Mihir Bellare and Viet Tung Hoang. Efficient schemes for committing authenticated encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 845–875. Springer, Heidelberg, May / June 2022.

[BHK+99] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 216–233. Springer, Heidelberg, August 1999.

[Bon17] Xavier Bonnetain. Quantum key-recovery on full AEZ. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 394–406. Springer, Heidelberg, August 2017.

[BR00] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 317–330. Springer, Heidelberg, December 2000.

[BRRS09] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *SAC 2009*, volume 5867 of *LNCS*, pages 295–312. Springer, Heidelberg, August 2009.

[CB18] Paul Crowley and Eric Biggers. Adiantum: length-preserving encryption for entry-level processors. *IACR Trans. Symm. Cryptol.*, 2018(4):39–61, 2018.

[CG16] Colin Chaigneau and Henri Gilbert. Is AEZ v4.1 sufficiently resilient against key-recovery attacks? *IACR Trans. Symm. Cryptol.*, 2016(1):114–133, 2016. https://tosc.iacr.org/index.php/ToSC/article/view/538.

[CHB21] Paul Crowley, Nathan Huckleberry, and Eric Biggers. Length-preserving encryption with HCTR2. Cryptology ePrint Archive, Report 2021/1441, 2021. https://eprint.iacr.org/2021/1441.

[CN08] Debrup Chakraborty and Mridul Nandi. An improved security bound for HCTR. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 289–302. Springer, Heidelberg, February 2008.

[CR22] John Chan and Phillip Rogaway. On committing authenticated-encryption. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part II*, volume 13555 of *LNCS*, pages 275–294. Springer, Heidelberg, September 2022.

[CT21] Yu Long Chen and Stefano Tessaro. Better security-efficiency trade-offs in permutation-based two-party computation. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 275–304. Springer, Heidelberg, December 2021.

[DGRW18] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186. Springer, Heidelberg, August 2018.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 149–165. Springer, Heidelberg, January 1997.

[FLLW17] Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Reforgeability of authenticated encryption schemes. In Josef Pieprzyk and Suriadi Suriadi, editors, *ACISP 17, Part II*, volume 10343 of *LNCS*, pages 19–37. Springer, Heidelberg, July 2017.

[FLS15] Thomas Fuhr, Gaëtan Leurent, and Valentin Suder. Collision attacks against CAESAR candidates - forgery and key-recovery against AEZ and Marble. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 510–532. Springer, Heidelberg, November / December 2015.

[FOR17] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symm. Cryptol.*, 2017(1):449–473, 2017.

[GLR17] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 66–97. Springer, Heidelberg, August 2017.

[Gug18] Oliver Gugger. AEZ implementation for node. https://www.npmjs.com/package/aez, 2018.

[Hal04] Shai Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 315–327. Springer, Heidelberg, December 2004.

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, April 2015.

[HKR17] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. AEZ v5: Authenticated encryption by enciphering. A submission to CAESAR competition, March 2017. https://web.cs.ucdavis.edu/~rogaway/aez/aez.pdf.

[HR04] Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *CT-RSA 2004*, volume 2964 of *LNCS*, pages 292–304. Springer, Heidelberg, February 2004.

[IIM21] Takanori Isobe, Ryoma Ito, and Kazuhiko Minematsu. Security analysis of SFrame. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021, Part II*, volume 12973 of *LNCS*, pages 127–146. Springer, Heidelberg, October 2021.

[KLLN16] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 207–237. Springer, Heidelberg, August 2016.

[Knu98] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1998.

[KW02] Lars R. Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, Heidelberg, February 2002.

[LGR21] Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning oracle attacks. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 195–212. USENIX Association, August 2021.

[LRW11] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. *Journal of Cryptology*, 24(3):588–613, July 2011.

[Mat15] Nick Mathewson. AEZ for relay cryptography. https://lists.torproject.org/pipermail/tor-dev/2015-December/010080.html, 2015.

[MD23] Nicky Mouha and Morris Dworkin. Report on the block cipher modes of operation in the NIST SP 800-38 series. National Institute of Standards and Technology, NIST IR 8459 ipd, U.S. Department of Commerce, March 2023.

[Men17] Bart Mennink. Weak keys for AEZ, and the external key padding attack. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 223–237. Springer, Heidelberg, February 2017.

[Min14] Kazuhiko Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 275–292. Springer, Heidelberg, May 2014.

[MLGR23] Sanketh Menda, Julia Len, Paul Grubbs, and Thomas Ristenpart. Context discovery and commitment attacks - how to break CCM, EAX, SIV, and more. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part IV*, volume 14007 of *LNCS*, pages 379–407. Springer, Heidelberg, April 2023.

[Mok18] Nazar Mokrynskyi. AEZ cipher compiled to WebAssembly. https://www.npmjs.com/package/aez.wasm, 2018.

[MT05] Kazuhiko Minematsu and Yukiyasu Tsunoo. Hybrid symmetric encryption using known-plaintext attack-secure components. In *ICISC*, volume 3935 of *LNCS*, pages 242–260. Springer, 2005.

[Nat23] National Institute of Standards and Technology. The third NIST workshop on block cipher modes of operation 2023. https://csrc.nist.gov/Events/2023/third-workshop-on-block-cipher-modes-of-operation, 2023.

[PRS23] Tommy Pauly, Eric Rosenberg, and David Schinazi. Quic-aware proxying. IETF 116, https://datatracker.ietf.org/meeting/116/materials/slides-116-masque-quic-aware-proxying-00, 2023.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, Heidelberg, December 2004.

[Rog06] Phillip Rogaway. Formalizing Human Ignorance. In *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006.

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002.

[WFW05] Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *Information Security and Cryptology*, pages 175–188, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

A Proof of Lemma 1

An adversary has oracle access to the underlying permutation π. It can make forward and inverse queries to its oracles, and the queries are stored in a query history τ. For the i-th query of the adversary, if it is a forward query $Y \leftarrow \pi(X)$ then we store an entry $(i, X, Y, +)$. Otherwise, if it is a backward query $X \leftarrow \pi^{-1}(Y)$, then we store a corresponding entry $(i, X, Y, -)$. In this work, we consider information-theoretic adversaries, which have unbounded computational power.

Let S be the collection of ordered subsets $\mathcal{I} = (r_1, \ldots, r_s)$ in $\{1, \ldots, p\}$, with $r_1 < \cdots < r_s$. For $\mathcal{I} = (r_1, \ldots, r_s)$ with $(r_i, X_i, Y_i, *)$ as the corresponding entry of the r_i-th queries in the transcript, let $\mathbf{Bad}(\mathcal{I})$ be the event that

$$(Y_1 +_{32} X_1)[1..m] = \cdots = (Y_s +_{32} X_s)[1..m].$$

Let

$$\mathbf{Bad} = \bigcup_{\mathcal{I} \in S} \mathbf{Bad}(\mathcal{I}).$$

It is easy to see that if **Bad** does not happen, then $\mathcal{A}$ cannot produce an s-way multi-collision. Suppose that **Bad** indeed does not happen. Let $(X_1, \ldots, X_s)$ be the output of $\mathcal{A}$. For each X_i, there must be a corresponding entry $(r_i, X_i, Y_i, *)$ in the transcript (since Y_i is uniquely determined by X_i and vice-versa). Without loss of generality, assume that $r_1 < \cdots < r_s$, and let $\mathcal{I} = (r_1, \ldots, r_s)$. Then $(X_1, \ldots, X_s)$ is an s-way multi-collision if and only if $\mathbf{Bad}(\mathcal{I})$ happens. As Bad does not happen, the adversary does not create an s-way multi-collision. Hence, we have

$$\mathbf{Adv}^{\mathsf{Coll}}_{\mathsf{pDM},s}(\mathcal{A}) \leq \Pr[\mathbf{Bad}(\mathcal{I})].$$

Bounding the Chance of Bad. We have the following claim.

Claim. *For each $\mathcal{I} \in S$,*

$$\Pr[\mathbf{Bad}(\mathcal{I})] \leq \frac{2^{s-1}}{2^{(s-1)m}}.$$

Proof of the claim. Fix $\mathcal{I} = (r_1, \ldots, r_s) \in S$, and let $(r_i, X_i, Y_i, *)$ be the corresponding entry of the r_i-th queries in the transcript. Fix $i \in \{2, \ldots, s\}$. We consider the following cases.

- The r_i-th entry is $(r_i, X_i, Y_i, +)$. Then, given X_i and all prior queries/answers before the r_i-th query, Y_i is uniformly distributed over a set of at least $2^n - p \geq 2^{n-1}$ values, and thus the conditional probability that $(Y_1 +_{32} X_1)[1..m] = \cdots = (Y_s +_{32} X_s)[1..m]$ is at most $2^{n-m}/2^{n-1} = 2^{1-m}$.

- The r_i-th entry is $(r_i, X_i, Y_i, -)$. Then, given Y_i and all prior queries/answers before the r_i-th query, X_i is uniformly distributed over a set of at least $2^n - p \geq 2^{n-1}$ values, and thus the conditional probability that $(Y_1 +_{32} X_1)[1..m] = \cdots = (Y_s +_{32} X_s)[1..m]$ is at most $2^{n-m}/2^{n-1} = 2^{1-m}$.

Multiplying these conditional probabilities for all $i \in \{2, \ldots, s\}$, we obtain

$$\Pr[\mathbf{Bad}(\mathcal{I})] \leq 2^{(s-1)(1-m)}.$$

□

By the union bound, we have

$$\Pr[\mathbf{Bad}] = \Pr[\bigcup_{\mathcal{I}\in S} \mathbf{Bad}(\mathcal{I})],$$
$$\leq \sum_{\mathcal{I}\in S} \Pr[\mathbf{Bad}(\mathcal{I})] \leq \binom{p}{s}\frac{2^{s-1}}{2^{(s-1)m}} = \binom{p}{s}2^{(1-m)(s-1)}.$$

B More Detailed Specification of AEZ

B.1 Pseudocode

The pseudocode of AEZ enciphering and deciphering algorithms is described in Algs. 1, 2, and 3. We omit the pseudocode of the AEZ-tiny enciphering, AEZ-tiny deciphering, and BLAKE2b algorithms because these are out of scope in our analysis.

B.2 Notation

Symbols and Strings. The following symbols and strings are used in the pseudocode:

$|X|$: The length of a string X.
ϵ : An empty string, *i.e.*, $|\epsilon| = 0$.
$A \parallel B$: The concatenation of strings A and B.
X^n : A string X repeated n times, *e.g.*, $0^3 = 000$ and $(01)^2 = 0101$.
$\mathbf{0}$: 0^{128}.
$X0^*$: $X0^p$ with p the smallest number such that 128 divides $|X| + p$.
$X[i..j]$: $X[i]\cdots X[j]$, where $X[i]$ is the ith bit of X and $\mathsf{msb}(X) = 1$.
$[n]_t$: A t-bit string representing n mod 2^t.
$\lceil x \rceil$: Mapping x to the least integer greater than or equal to x.
K : Key, $|K| \geq 128$ is recommended, the default is $|K| = 384$.
N : Nonce, $|N| \leq 128$ is recommended.
$\boldsymbol{A}$: Associated data, which is regarded as a one-element vector.
τ : Tag length, $|\tau| = 8 \cdot$ ABYTES.
ABYTES : An authenticator length, the default is 16.
$\boldsymbol{T}$: Tweak, which encodes N, $\boldsymbol{A}$, and ABYTES.
M : Plaintext, $M_1M_1'\cdots M_mM_m'M_{uv}M_xM_y \leftarrow M$,
where $|M_1| = \cdots |M_m'| = |M_x| = |M_y| = 128$ and $|M_{vu}| < 256$.
C : Ciphertext, $C_1C_1'\cdots C_mC_m'C_{uv}C_xC_y \leftarrow C$,
where $|C_1| = \cdots |C_m'| = |C_x| = |C_y| = 128$ and $|C_{vu}| < 256$.

Multiplication. $\mathbb{N}$ means a non-negative integer, *i.e.*, $\mathbb{N} = \{0, 1, 2, \dots\}$. A 128-bit string X is denoted as $x_1 \cdots x_{128}$; then, $X \ll 1 = x_2 \cdots x_{128}0$. For $n \in \mathbb{N}$ and $X \in \{0,1\}^{128}$, a multiplication $n \cdot X$ is defined by asserting that $0 \cdot X = \mathbf{0}$, $1 \cdot X = X$, $2 \cdot X = (X \ll 1) \oplus [135 \cdot \mathtt{msb}_1(X)]_{128}$, $2n \cdot X = (2 \cdot (n \cdot X))$, and $(2n+1) \cdot X = (2n \cdot X) \oplus X$.

AES4 and AES10. For $X, K \in \{0,1\}^{128}$, $\mathsf{aesenc}(X, K)$ is defined as a single round of AES, *i.e.*, it permutes X by performing SubBytes, ShiftRows, MixColumns, and then AddRoundKey with K. For $\boldsymbol{k} = (K_0, K_1, K_2, K_3, K_4)$, $\mathsf{AES4}_{\boldsymbol{k}}(X)$ is defined as

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

For $\boldsymbol{K} = (K_0, K_1, \ldots, K_{10})$, $\mathsf{AES10}_{\boldsymbol{K}}(X)$ is defined in the same manner as $\mathsf{AES4}_{\boldsymbol{k}}(X)$. Note that AES4 and AES10 contain MixColumns in the last round, unlike the original AES.

Others. See [HKR17] for more details.

Algorithm 1 AEZ enciphering routines

procedure Encrypt$(K, N, \boldsymbol{A}, \tau, M)$ ▷ AEZ authenticated encryption
  $X \leftarrow M \,\|\, 0^\tau$; $(A_1, \ldots, A_a) \leftarrow \boldsymbol{A}$
  $\boldsymbol{T} \leftarrow ([\tau]_{128}, N, A_1, \ldots, A_a)$
  if $M = \epsilon$ **then**
    return AEZ-prf$(K, \boldsymbol{T}, \tau)$
  return Encipher$(K, \boldsymbol{T}, X)$

procedure Encipher$(K, \boldsymbol{T}, X)$ ▷ AEZ enciphering
  if $|X| < 256$ **then**
    return Encipher-AEZ-tiny$(K, \boldsymbol{T}, X)$
  if $|X| \geq 256$ **then**
    return Encipher-AEZ-core$(K, \boldsymbol{T}, X)$

procedure Encipher-AEZ-core$(K, \boldsymbol{T}, M)$ ▷ AEZ-core enciphering
  $\Delta \leftarrow$ AEZ-hash$(K, \boldsymbol{T})$
  $M_1 M_1' \cdots M_m M_m' M_{uv} M_x M_y \leftarrow M$; $d \leftarrow |M_{uv}|$
  if $d \leq 127$ **then**
    $M_u \leftarrow M_{uv}$; $M_v \leftarrow \epsilon$
  else
    $M_u \leftarrow M_{uv}[1..128]$; $M_v \leftarrow M_{uv}[129..|M_{uv}|]$
  for $i \leftarrow 1$ **to** m **do**
    $W_i \leftarrow M_i \oplus E_K^{1,i}(M_i')$; $X_i \leftarrow M_i' \oplus E_K^{0,0}(W_i)$
  if $d = 0$ **then**
    $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathbf{0}$
  else if $d \leq 127$ **then**
    $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus E_K^{0,4}(M_u 10^*)$
  else
    $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus E_K^{0,4}(M_u) \oplus E_K^{0,5}(M_v 10^*)$
  $S_x \leftarrow M_x \oplus \Delta \oplus X \oplus E_K^{0,1}(M_y)$; $S_y \leftarrow M_y \oplus E_K^{-1,1}(S_x)$; $S \leftarrow S_x \oplus S_y$
  for $i \leftarrow 1$ **to** m **do**
    $S' \leftarrow E_K^{2,i}(S)$; $Y_i \leftarrow W_i \oplus S'$; $Z_i \leftarrow X_i \oplus S'$
    $C_i' \leftarrow Y_i \oplus E_K^{0,0}(Z_i)$; $C_i \leftarrow Z_i \oplus E_K^{1,i}(C_i')$
  if $d = 0$ **then**
    $C_u \leftarrow C_v \leftarrow \epsilon$; $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathbf{0}$
  else if $d \leq 127$ **then**
    $C_u \leftarrow M_u \oplus E_K^{-1,4}(S)$; $C_v \leftarrow \epsilon$; $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus E_K^{0,4}(C_u 10^*)$
  else
    $C_u \leftarrow M_u \oplus E_K^{-1,4}(S)$; $C_v \leftarrow E_K^{-1,5}(S)$
    $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus E_K^{0,4}(C_u) \oplus E_K^{0,5}(C_v 10^*)$
  $C_y \leftarrow S_x \oplus E_K^{-1,2}(S_y)$; $C_x \leftarrow S_y \oplus \Delta \oplus Y \oplus E_K^{0,2}(C_y)$
  return $C_1 C_1' \cdots C_m C_m' C_{uv} C_x C_y$

Algorithm 2 AEZ deciphering routines

procedure Decrypt$(K, N, \boldsymbol{A}, \tau, C)$ ▷ AEZ authenticated decryption
  $(A_1, \ldots, A_a) \leftarrow \boldsymbol{A}$; $\boldsymbol{T} \leftarrow ([\tau]_{128}, N, A_1, \ldots, A_a)$
  if $|C| < \tau$ **then**
    return $\perp$
  if $|C| = \tau$ **then**
    if $C = \mathsf{AEZ\text{-}prf}(K, \boldsymbol{T}, \tau)$ **then**
      return ϵ
    else
      return $\perp$
  $X \leftarrow \mathsf{Decipher}(K, \boldsymbol{T}, C)$
  $M \parallel Z \leftarrow X$ where $|Z| = \tau$
  if $Z = 0^\tau$ **then**
    return M
  else
    return $\perp$

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

procedure Decipher-AEZ-core$(K, \boldsymbol{T}, C)$ ▷ AEZ-core deciphering
  $\Delta \leftarrow \mathsf{AEZ\text{-}hash}(K, \boldsymbol{T})$
  $C_1 C_1' \cdots C_m C_m' C_{uv} C_x C_y \leftarrow C$; $d \leftarrow |C_{uv}|$
  if $d \leq 127$ **then**
    $C_u \leftarrow C_{uv}$; $C_v \leftarrow \epsilon$
  else
    $C_u \leftarrow C_{uv}[1..128]$; $C_v \leftarrow C_{uv}[129..|C_{uv}|]$
  for $i \leftarrow 1$ **to** m **do**
    $W_i \leftarrow C_i \oplus E_K^{1,i}(C_i')$; $Y_i \leftarrow C_i' \oplus E_K^{0,0}(W_i)$
  if $d = 0$ **then**
    $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus \mathbf{0}$
  else if $d \leq 127$ **then**
    $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus E_K^{0,4}(C_u 10^*)$
  else
    $Y \leftarrow Y_1 \oplus \cdots \oplus Y_m \oplus E_K^{0,4}(C_u) \oplus E_K^{0,5}(C_v 10^*)$
  $S_x \leftarrow C_x \oplus \Delta \oplus Y \oplus E_K^{0,2}(C_y)$; $S_y \leftarrow C_y \oplus E_K^{-1,2}(S_x)$; $S \leftarrow S_x \oplus S_y$
  for $i \leftarrow 1$ **to** m **do**
    $S' \leftarrow E_K^{2,i}(S)$; $X_i \leftarrow W_i \oplus S'$; $Z_i \leftarrow Y_i \oplus S'$
    $M_i' \leftarrow X_i \oplus E_K^{0,0}(Z_i)$; $M_i \leftarrow Z_i \oplus E_K^{1,i}(M_i')$
  if $d = 0$ **then**
    $M_u \leftarrow M_v \leftarrow \epsilon$; $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus \mathbf{0}$
  else if $d \leq 127$ **then**
    $M_u \leftarrow C_u \oplus E_K^{-1,4}(S)$; $M_v \leftarrow \epsilon$; $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus E_K^{0,4}(M_u 10^*)$
  else
    $M_u \leftarrow C_u \oplus E_K^{-1,4}(S)$; $M_v \leftarrow E_K^{-1,5}(S)$
    $X \leftarrow X_1 \oplus \cdots \oplus X_m \oplus E_K^{0,4}(M_u) \oplus E_K^{0,5}(M_v 10^*)$
  $M_y \leftarrow S_x \oplus E_K^{-1,1}(S_y)$; $M_x \leftarrow S_y \oplus \Delta \oplus X \oplus E_K^{0,1}(M_y)$
  return $M_1 M_1' \cdots M_m M_m' M_{uv} M_x M_y$

Algorithm 3 AEZ's hash, PRF, TBC, and key-derivation algorithms

procedure AEZ-hash$(K, \boldsymbol{T})$ ▷ Atmost XOR universal hash
  $(T_1, \ldots, T_t) \leftarrow \boldsymbol{T}$
  for $i \leftarrow 1$ **to** t **do**
    $\ell \leftarrow \max(1, \lceil |T_i|/128 \rceil)$; $j \leftarrow i + 2$
    $Z_1 \cdots Z_\ell \leftarrow T_i$ where $|Z_1| = \cdots = |Z_{\ell-1}| = 128$
    if $|Z_\ell| = 128$ **then**
      $\Delta_i \leftarrow E_K^{j,1}(Z_1) \oplus \cdots \oplus E_K^{j,\ell}(Z_\ell)$
    if $|Z_\ell| < 128$ **then**
      $\Delta_i \leftarrow E_K^{j,1}(Z_1) \oplus \cdots \oplus E_K^{j,\ell-1}(Z_{\ell-1}) \oplus E_K^{j,\ell}(Z_\ell 10^*)$
  return $\Delta_1 \oplus \cdots \oplus \Delta_t \oplus \mathbf{0}$

procedure AEZ-prf$(K, \boldsymbol{T}, \tau)$ ▷ PRF used when $M = \epsilon$
  $\Delta \leftarrow$ AEZ-hash$(K, \boldsymbol{T})$
  return $(E_K^{-1,3}(\Delta) \,\|\, E_K^{-1,3}(\Delta \oplus [1]_{128}) \,\|\, E_K^{-1,3}(\Delta \oplus [2]_{128}) \,\|\, \cdots)[1..\tau]$

procedure $E_K^{j,i}(X)$ ▷ Scalled-down TBC
  $I \,\|\, J \,\|\, L \leftarrow \mathsf{Extract}(K)$
  $\boldsymbol{K} \leftarrow (\mathbf{0}, I, J, L, I, J, L, I, J, L, I)$; $\boldsymbol{k} \leftarrow (\mathbf{0}, J, I, L, \mathbf{0})$
  if $j = -1$ **then**
    $\Delta \leftarrow i \cdot L$
    return $\mathsf{AES10}_{\boldsymbol{K}}(X \oplus \Delta)$
  $\Delta \leftarrow j \cdot J \oplus 2^{\lceil i/8 \rceil} \cdot I \oplus (i \bmod 8) \cdot L$
  return $\mathsf{AES4}_{\boldsymbol{k}}(X \oplus \Delta)$

procedure Extract(K) ▷ Key-derivation function
  if $|K| = 384$ **then**
    return K
  else
    return BLAKE2b(K)

C Example of Different Keys Enciphering to the Same Ciphertexts

We provide two data sets containing a key, a nonce, a vector of additional data and a plaintext that will produce the same ciphertext when enciphering with AEZ where $\tau = 128$.

```
[ Common Ciphertext ]
    e79c7ff859956834ae3fc030089a688cec2b3982873599ac0cfc8d057276d6e9
    5cc71d5d72c23eeddbb27fcfe9357ba795b2f393e7ef14ecae553923d4e6304c

[ First Data ]
key:
    03378c1350f6417f30bcd233d04a41ef
    55314b2cb7b7fa1d5d0fb626e7bd893f
    b1f649e86148f99bc50ef46154ee4fee
nonce: ec96b26c88627a1a7d2ec67203e3af5c
AD: [
    05, 05, 05, 05, 4c, 4c, 4c, 4c, 4c, 05, 4c, 05, 05, 05, 4c, 4c, 4c,
    4c, 05, 4c, 4c, 4c, 05, 05, 4c, 4c, 05, 4c, 4c, 05, 05, 4c, 05, 05,
    4c, 4c, 05, 05, 4c, 4c, 05, 05, 05, 4c, 4c, 05, 4c, 4c, 05, 05, 4c,
```

```
    4c, 4c, 4c, 4c, 05, 4c, 4c, 05, 4c, 4c, 05, 4c, 4c, 05, 4c, 05, 05,
    05, 05, 4c, 05, 4c, 05, 05, 05, 05, 05, 4c, 05, 4c, 4c, 4c, 4c, 4c,
    4c, 05, 4c, 05, 4c, 4c, 4c, 05, 05, 05, 05, 05, 05, 05, 05, 4c, 05,
    4c, 4c, 4c, 4c, 4c, 4c, 05, 4c, 4c, 4c, 05, 05, 4c, 05, 05, 05, 05,
    05, 05, 05, 4c, 4c, 4c, 05, 05, 05
]
plaintext:
    f4f20e8cce629a4685837f7d52bb35c4cffeac9f90b6d589
    d725fce1daa1fdc86ee2ed059d0b87422fe704040e2d0d17

[ Second Data ]
key:
    2b23982f50f6417f30bcd233d04a41ef
    55314b2cb7b7fa1d5d0fb626e7bd893f
    b0f649e86148f99bc50ef46154ee4fee
nonce: 7905db71fc712d6eb65e15a32b1b6ee5
AD: [
    4c, 4c, 05, 4c, 4c, 4c, 05, 4c, 4c, 05, 4c, 4c, 05, 05, 05, 4c, 05,
    05, 05, 4c, 05, 4c, 05, 05, 4c, 4c, 4c, 05, 05, 05, 05, 05, 05, 4c,
    4c, 05, 4c, 4c, 05, 4c, 4c, 05, 05, 4c, 05, 05, 4c, 05, 4c, 05, 4c,
    05, 05, 4c, 05, 4c, 4c, 05, 05, 4c, 05, 4c, 4c, 05, 05, 05, 4c, 4c,
    4c, 05, 4c, 05, 05, 05, 4c, 4c, 4c, 4c, 4c, 4c, 05, 05, 4c, 05, 05,
    05, 05, 05, 4c, 4c, 4c, 4c, 4c, 05, 05, 4c, 4c, 4c, 4c, 4c, 4c, 4c,
    05, 4c, 4c, 4c, 4c, 4c, 05, 05, 4c, 05, 05, 4c, 05, 05, 05, 4c, 05,
    05, 05, 05, 4c, 4c, 05, 05, 05, 4c
]
plaintext:
    40256d76012001a6623635a4cec61500a4e3d498aa0947cf
    8aae7bf5f683814643773e6d6696003f56f43245e0376835
}
```

D Repeated 4-tree algorithm

We here show an algorithm and complexity of the 4-tree attack shown in Sect. 3.3 when we do not have enough samples for $\widehat{\mathbb{X}_1}$, $\widehat{\mathbb{Y}_2}$, $\widehat{\mathbb{X}'_1}$, and $\widehat{\mathbb{Y}'_2}$. We set $2n/3 < \tau < n$ and $|\mathcal{L}_{\mathbb{X}_1}| = |\mathcal{L}_{\mathbb{Y}_2}| = |\mathcal{L}_{\mathbb{X}'_1}| = |\mathcal{L}_{\mathbb{Y}'_2}| = 2^{n-\tau}$. The algorithm is as follows.

1. Creating a list $\mathcal{L}_{\mathbb{X}_1\mathbb{Y}_2}$ which contains all pairs of $\langle\widehat{\mathbb{X}_1}, \widehat{\mathbb{Y}_2}\rangle$ satisfying $\widehat{\mathbb{X}_1} \in \mathcal{L}_{\mathbb{X}_1}$, $\widehat{\mathbb{Y}_2} \in \mathcal{L}_{\mathbb{Y}_2}$, and $\mathtt{msb}_{n-\tau}(\widehat{\mathbb{X}_1}) = \mathtt{msb}_{n-\tau}(\widehat{\mathbb{Y}_2})$. We also create $\mathcal{L}_{\mathbb{X}'_1\mathbb{Y}'_2}$ in the same way.
2. Find $\langle\widehat{\mathbb{X}_1}, \widehat{\mathbb{Y}_2}\rangle \in \mathcal{L}_{\mathbb{X}_1\mathbb{Y}_2}$ and $\langle\widehat{\mathbb{X}'_1}, \widehat{\mathbb{Y}'_2}\rangle \in \mathcal{L}_{\mathbb{X}'_1\mathbb{Y}'_2}$ such that $\mathtt{lsb}_\tau(\widehat{\mathbb{X}_1} \oplus \widehat{\mathbb{Y}_2}) = \mathtt{lsb}_\tau(\widehat{\mathbb{X}'_1} \oplus \widehat{\mathbb{Y}'_2})$. If we can find such a pair, the tuple of $(\widehat{\mathbb{X}_1}, \widehat{\mathbb{Y}_2}, \widehat{\mathbb{X}'_1}, \widehat{\mathbb{Y}'_2})$ satisfies $\widehat{\mathbb{X}_1} \oplus \widehat{\mathbb{Y}_2} \oplus \widehat{\mathbb{X}'_1} \oplus \widehat{\mathbb{Y}'_2} = 0^n$, we terminate this algorithm.
3. If we fail in the previous step, we resample $\widehat{\mathbb{X}_1}$, $\widehat{\mathbb{Y}_2}$, $\widehat{\mathbb{X}'_1}$, and $\widehat{\mathbb{Y}'_2}$ and create new four lists that are independent of previous lists. Then, we go back to step one.

Next, we evaluate the complexity of the above algorithm. The way to evaluate is almost the same as that of the original 4-tree algorithm [Wag02]. We would like to note that the list values are not truly random since they are permutation outputs. However, we can assume that they are uniformly random since the distinguishing probability between them and random values is $O(2^{n/2})$, and we are interested in only the case where the complexity

of the algorithm is up to $O(2^{n/2})$. The complexity of step one can be evaluated as $O(2^{n-\tau})$ due to $|\mathcal{L}_{\mathbb{X}_1}| = |\mathcal{L}_{\mathbb{Y}_2}| = |\mathcal{L}_{\mathbb{X}'_1}| = |\mathcal{L}_{\mathbb{Y}'_2}| = 2^{n-\tau}$ and their randomness[5] Also, we can expect $|\mathcal{L}_{\mathbb{X}_1\mathbb{Y}_2}|$ would be $|\mathcal{L}_{\mathbb{X}_1}| \times |\mathcal{L}_{\mathbb{Y}_2}|/2^{n-\tau} = 2^{n-\tau}$, and also $|\mathcal{L}_{\mathbb{X}'_1\mathbb{Y}'_2}|$ has expected size $2^{n-\tau}$. Thus, the complexity of step two can be evaluated as $O(2^{n-\tau})$. The probability that we can find the desired tuple $(\widehat{\mathbb{X}_1}, \widehat{\mathbb{Y}_2}, \widehat{\mathbb{X}'_1}, \widehat{\mathbb{Y}'_2})$ in step two is $|\mathcal{L}_{\mathbb{X}_1\mathbb{Y}_2}| \times |\mathcal{L}_{\mathbb{X}_1\mathbb{Y}_2}|/2^{\tau} = 2^{2n-3\tau}$. We can expect to terminate the algorithm in step two by repeating the above steps $2^{3\tau-2n}$ times. Thus, the required complexity of whole steps is $O(2^{n-\tau}) \times 2^{3\tau-2n} = O(2^{2\tau-n})$, which includes query complexities for (re-)sampling $\widehat{\mathbb{X}_1}$, $\widehat{\mathbb{Y}_2}$, $\widehat{\mathbb{X}'_1}$, and $\widehat{\mathbb{Y}'_2}$.

[5]The complexity becomes $\widetilde{O}(2^{n-\tau}) = O(2^{n-\tau}\log(2^{n-\tau}))$ using a simple sort algorithm. However, it is well-known that the complexity is reduced to $O(2^{n-\tau})$ when the elements of the lists are random [Knu98].