



On Committing Authenticated-Encryption

John Chan(✉) and Phillip Rogaway

Department of Computer Science, University of California, Davis, USA
jmachan@ucdavis.edu, rogaway@cs.ucdavis.edu

Abstract. We provide a strong definition for *committing* authenticated-encryption (cAE), as well as a framework that encompasses earlier and weaker definitions. The framework attends not only to *what* is committed but also the extent to which the adversary knows or controls keys. We slot into our framework strengthened cAE-attacks on GCM and OCB. Our main result is a simple and efficient construction, CTX, that makes a nonce-based AE (nAE) scheme committing. The transformed scheme achieves the strongest security notion in our framework. Just the same, the added computational cost (on top of the nAE scheme’s cost) is a single hash over a short string, a cost independent of the plaintext’s length. And there is *no* increase in ciphertext length compared to the base nAE scheme. That such a thing is possible, let alone easy, upends the (incorrect) intuition that you can’t commit to a plaintext or ciphertext without hashing one or the other. And it motivates a simple and practical tweak to AE-schemes to make them committing.

Keywords: AEAD · Authenticated encryption · Committing encryption · Key-robustness

1 Introduction

A natural misconception about authenticated encryption (AE) is the belief that a ciphertext produced by encrypting a plaintext with a key, nonce, and associated data (AD) effectively *commits* to those things: decrypting it with some *other* key, nonce, or AD will usually fail, the transmission deemed invalid. And why not? One wouldn’t expect to successfully open a lock when using an incorrect key. The intuition is even memorialized in the name *authenticated* encryption: things aren’t just private, the name implies, but authentic.

Yet Farshim, Orlandi, and Roşie [9] (FOR17) point out that AE provides no such guarantee—not if the adversary can select any keys. Subsequent work demonstrated that just *knowing* the keys suffices to construct a ciphertext that decrypts into different valid messages [7, 11]. A variety of work has also made clear just how wrong things can go when designers implicitly and incorrectly assume that their encryption *is* committing [2, 7, 13].

We call the event of a ciphertext being “explained” in multiple and valid ways a *misattribution*. The cited works offer definitions and schemes that seek

to protect against misattribution. But these definitions are mostly incomparable, weak, and fold in aims beyond avoiding misattribution.

DEFINITIONAL FRAMEWORK. To begin, we revisit definitions for committing AE. We offer a definitional framework that unifies and strengthens previous definitions targeting misattribution. We call the security goals *committing-AE* (cAE). The framework applies to schemes for *nonce-based AE with associated data* (nAE). Encryption takes in a key K , a nonce N , an associated data A , and a message M , and outputs a ciphertext C . Under our framework, an adversary succeeds in an attack when it creates a misattribution. That happens when C results from known and distinct tuples (K, N, A, M) and (K', N', A', M') for valid messages M, M' . We say “results from” because C could be output by encryption *or* input to decryption—anything that results in adversarial knowledge of the pair (K, N, A, M) , (K', N', A', M') .

Previous definitions consider only some forms of misattribution. For example, the *full robustness* and *key-commitment* notions [2, 9] require that the keys differ, $K \neq K'$, but ignore the possibility of misattribution under the same key. Our framework can encompass all possible types of misattribution. That said, we regard the desired target as the *strongest* definition, AE that is *fully committing*, where the adversary wins if it manages *any* form of misattribution.

Our framework attends also to the status of keys held by parties. To model different levels of adversarial activity, we include a definitional parameter \mathbf{t} . This two-character string dictates what types of keys the adversary might employ for a misattribution to occur. Keys are either: *honest* (represented by the character 0), meaning they are generated uniformly at random and remain unknown to the adversary; *revealed* (represented by a 1), meaning they were honestly generated, but the adversary knows their value; or *corrupted* (an X), meaning the adversary itself chose the key. This gives rise to six different definitions. This “knob” is useful for describing and understanding attacks. The weakest of these notions models when both keys are honest. We show that ordinary nAE-security implies this notion assuming the adversaries do not repeat nonces for the same key. Many applications that require cAE security would work just fine with 0X-security, and stronger quantitative bounds might be obtained for this case.

MAIN CONSTRUCTION. Our main result is a method to convert an arbitrary (tag-based) nAE scheme into a similarly efficient cAE scheme. We set high bars for security and efficiency. Security is with respect to the strongest form of commitment: K , N , A , and M must all be “fixed” by a ciphertext, even if the adversary controls all keys.

Our CTX construction is extremely simple. Starting from an nAE scheme whose encryption algorithm $\mathcal{E}(K, N, A, M)$ produces a ciphertext $\mathcal{C} = C \parallel T$ consisting of a ciphertext core C (with $|C| = |M|$) and a tag T (with $|T| = \tau$), just replace the tag T with an alternative tag $T^* = H(K, N, A, T)$ (this tag of length μ). Decryption does the obvious, verifying T^* . The function H is a cryptographic hash function that, in the security proofs, is modeled as a random oracle. The remarkable fact is that this extremely simple tweak to the nAE scheme not only works to commit to K , N , and A , but also to the underlying

message M . This ultimately follows from the injectivity of the map from the ciphertext core C to the plaintext M when K , N , and A are all fixed.

The CTX construction is computationally efficient insofar as the work on top of the base nAE scheme is a hash computation over a string that does not grow with the plaintext or ciphertext. And the nAE scheme’s minimal ciphertext expansion is preserved, going from the τ (typically 128) extra bits that are needed to provide authenticity to the μ (typically 160) extra bits that are needed to provide authenticity *and* the binding (commitment) of all inputs.

ATTACKS ON GCM AND OCB. Previous misattribution attacks on GCM were mounted with adversarial control of the keys [7, 11]. It is mentioned by those same authors that knowledge of the keys is sufficient. Under our terminology, this would be a CAE_{xx} -attack and a CAE_{11} -attack respectively.

We present a new attack on GCM for a weaker adversary, a CAE_{01} -attack. That is, the adversary can create a misattribution knowing just one key. For any ciphertext C generated under a perfectly honest key, one can find a valid decryption for it under a known key. The attack strategy involves computing an AD that validates the decryption of the ciphertext. Intuitively, for any key, nonce, message, and ciphertext, there are an infinite number of ADs that validly decrypt the ciphertext—we only need to find one of them. The strategy extends to mounting a CAE_{01} -attack on OCB as well. These attacks demonstrate that nAE-security is insufficient for even CAE_{01} -security.

RELATED WORK. Prior work has been leading towards a definition for fully committing AE (the cAE-xx notion), but didn’t quite get there. There has also been movement towards efficient schemes for this end.

The notion of committing encryption goes back to 2003 with Gertner and Herzberg [10], who consider the problem in both the symmetric and asymmetric settings. The authors do not look at deterministic or authenticated encryption.

Abdalla, Bellare, and Neven give definitions for what they call *robustness* [1] in the asymmetric setting. Their notion requires an adversary to produce a ciphertext that validly decrypts under two different keys. It encompasses keys that are honestly generated. Later, Farshim, Libert, Paterson, and Quaglia observe that, for some applications, robustness against adversarially-chosen keys is critical [8]. They then strengthen Abdalla et al.’s notion accordingly.

Farshim, Orlandi, and Roşie (FOR17) [9] contextualized Abdalla et al.’s robustness in the AE setting, initializing the study of what we call committing AE. Shortly after, Grubbs, Lu, and Ristenpart (GLR17) [11] defined a variant of committing AE with the goal of constructing schemes that support *message franking*. Dodis, Grubbs, Ristenpart, and Woodage (DGRW18) [7] also target message franking and further develop GLR17’s definitions. These two works have goals beyond preventing misattributions. We are after simpler aims, with the syntax of classical nAE. Albertini, Duong, Gueron, Kölbl, Luykx, and Schmiege (ADGKLS20) [2] observe the possibility of mitigating the attacks described by GLR17 and DGRW18 under a weaker form of misattribution prevention. Their observation led them to develop a more efficient construction—one that avoids additional passes over the message.

Bellare and Hoang (BH22), in a contemporary work, offer a range of committing AE definitions, with starting points of both standard nAE and misuse-resistant AE [5]. The strongest of their definitions, like ours, requires that the ciphertext commit to *everything*—the key, nonce, AD, and plaintext. They also consider *multi-input committing security*, where an adversary is required to create misattributions of more than just two valid explanations.

Len, Grubbs, and Ristenpart demonstrate password-recovery attacks on non-committing password-based AEAD schemes [13]. Their attacks are built on efficiently creating ciphertexts that successfully decrypt under many keys.

A more detailed comparison of some of the cited works is in Sect. 6.

2 Preliminaries

COLLISION-RESISTANT HASH FUNCTIONS. A *hash function* $H: \mathcal{D} \rightarrow \{0, 1\}^h$ maps strings from some domain $\mathcal{D} \subset \{0, 1\}^*$ to strings of length h . Informally, a hash function is collision-resistant if it is difficult for an adversary \mathcal{A} to find two unique inputs that map to the same output. This notion is captured by a collision resistance game CR where \mathcal{A} is ran and outputs a pair (M, M') . The game outputs true if $H(M) = H(M')$ and $M \neq M'$. The adversary \mathcal{A} 's advantage against H is then quantified as $\mathbf{Adv}_H^{\text{col}}(\mathcal{A}) = \Pr[\text{CR}_H^{\text{col}} \Rightarrow \text{true}]$. This definition of collision-resistance of unkeyed hash functions follows the human-ignorance approach of [17].

NONCE-BASED AE. An nAE scheme, or a *nonce-based authenticated encryption-scheme supporting associated data (AD)* is a pair of functions $(\mathcal{E}, \mathcal{D})$. The former, the *encryption algorithm*, is a deterministic function $\mathcal{E}: \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \rightarrow \mathcal{C}$ that takes in a key, a nonce, an AD, and a message, and outputs a ciphertext. The latter \mathcal{D} , the *decryption algorithm*, is a deterministic function $\mathcal{D}: \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{M} \times \{\perp\}$. We sometimes write $\mathcal{E}_K^{N,A}(M)$ and $\mathcal{D}_K^{N,A}(C)$ to denote $\mathcal{E}(K, N, A, M)$ and $\mathcal{D}(K, N, A, C)$. An nAE scheme is *correct* if $\mathcal{D}_K^{N,A}(\mathcal{E}_K^{N,A}(M)) = M$ for all K, N, A, M . A notable property of correct schemes is how encryption is injective from \mathcal{M} to \mathcal{C} when K, N, A are fixed. $K \in \mathcal{K}, N \in \mathcal{N}, A \in \mathcal{A}, M \in \mathcal{M}$ and $\mathcal{D}_K^{N,A}(C) = \perp$ otherwise. We assume that the message space $\mathcal{M} \subseteq \{0, 1\}^*$ is a set of strings where $M \in \mathcal{M}$ implies $\{0, 1\}^{|M|} \in \mathcal{M}$. We insist of an nAE scheme that there is an associated value, its *expansion*, which is a constant τ such that $|\mathcal{E}_K^{N,A}(M)| = |M| + \tau$ for all K, N, A, M .

A customary formulation of nAE security asks an adversary attacking the nAE scheme $\Pi = (\mathcal{E}, \mathcal{D})$ to distinguish between a pair of oracles [16]. The “real” oracles use Π 's algorithms while the “ideal” or “fake” oracles only give bogus responses. For an adversary \mathcal{A} attacking Π , its advantage is defined as follows:

$$\begin{aligned} \mathbf{Adv}_\Pi^{\text{nae}}(\mathcal{A}) = & \Pr[K \leftarrow \mathcal{K}; \mathcal{A}^{E_K(\cdot, \cdot, \cdot), D_K(\cdot, \cdot, \cdot)} \Rightarrow 1] \\ & - \Pr[\mathcal{A}^{\mathcal{S}(\cdot, \cdot, \cdot), \perp(\cdot, \cdot, \cdot)} \Rightarrow 1]. \end{aligned}$$

Its interaction with the E_K and D_K oracles, the “real” oracles, begins with the uniformly random sampling of a key K . An oracle query then $E_K(N, A, M)$

returns $\mathcal{E}_K(N, A, M)$, while an oracle query $D_K(N, A, C)$ returns $\mathcal{D}_K(N, A, C)$. In contrast, an “ideal” oracle query of $\$(N, A, M)$ returns a uniformly random string of length $|M| + \tau$, while $\perp(N, A, C)$ always returns \perp .

The adversary is forbidden from querying its decryption oracles with N, A, C if it acquired C from its encryption oracles using N, A as doing so would allow it to trivially win. Similarly, the definition demands that adversaries are *nonce-respecting*, meaning that they never repeat a nonce in its encryption queries.

We will find it useful to define a variant of nAE security that directly models multiple keys, which was first formalized in [6]. In this variant, an infinite number of keys are uniformly randomly generated for the real oracles at the initialization of the security game. Each oracle takes in an additional parameter, an index, that the adversary uses to specify which key to use for its query. Its advantage notion in this game is:

$$\begin{aligned} \text{Adv}_{\Pi}^{\text{nae}^*}(\mathcal{A}) = & \Pr[\mathbf{K} \leftarrow \mathcal{K}^\infty; \mathcal{A}^{E_K(\cdot, \cdot, \cdot), D_K(\cdot, \cdot, \cdot)} \Rightarrow 1] \\ & - \Pr[\mathcal{A}^{\$, \perp(\cdot, \cdot, \cdot)} \Rightarrow 1]. \end{aligned}$$

Similarly, the adversary is restricted from querying (i, N, A, C) to its decryption oracle if C is the result of some (i, N, A, M) . The adversary is nonce-respecting in this case if it never repeats the same nonce for the same key when querying the encryption oracle.

3 Committing AE

COMMITTING AE. Informally, we call an nAE scheme a *committing AE scheme* (cAE) if it commits to any of the elements used to produce a ciphertext. We are primarily interested in cAE schemes that commit to all of these elements. By the definition of nAE in Sect. 2, those elements would be the key, nonce, AD, and message. The CAE game that captures this property is presented in Fig. 1.

An adversary attacking the CAE-security of an nAE scheme Π aims to produce a ciphertext C that has two distinct valid “explanations.” That is, ciphertext C could decrypt to a messages M using (K_i, N, A) , or it could decrypt to a message M' using (K_j, N', A') such that $(K_i, N, A, M) \neq (K_j, N', A', M')$ and $M, M' \neq \perp$. When either of these occur, we say that the ciphertext is *misattributed*. We sometimes refer to C as the *colliding ciphertext* and the (K, N, A, M) associated to it as one of its attributions. In the game code, we write $S \stackrel{\cup}{\leftarrow} \{x\}$ as shorthand for $S \stackrel{\cup}{\leftarrow} S \cup \{x\}$, adding x to the set S .

The adversary initializes the game with the Initialize procedure, which generates an infinite number of uniformly random keys indexed by the natural numbers. Several sets are also initialized, one of which is the set S which keeps track of (K, N, A, M, C) tuples that constitute encryption and decryption queries and responses made by the adversary. The game terminates with the Finalize procedure, which checks the tuples of S in a pairwise fashion for an adversarial win. That is, it searches for a pair of tuples where the ciphertexts are equivalent and that the explanations are distinct and valid. There is an additional condition checked that pertains to the function chk that we describe later.

There are four other game procedures surfaced to the adversary: ENC, DEC, REV, COR. These are the encryption, decryption, reveal, and corruption oracles. The first two oracles let the adversary use Π 's encryption and decryption algorithms using a key specified by an index i . Any ciphertext or message generated by the call to Π 's algorithms is stored alongside the queried K_i, N, A, M (or C) are stored in the set S . The reveal oracle allows the adversary to query an index i and learn the key K_i . For the corruption oracle, the adversary queries an index i and a key K and supplants K_i with K . Keys that are affected by these two oracles are added to the sets K_r and K_c respectively.

Note that ENC queries are restricted to be *nonce-respecting* for honest keys. That is, an adversary cannot repeat nonces for its encryption queries to an honest key K_i . This is reflected in the game code. The purpose of this is to prevent possibilities of an adversary in learning an honest key through abuse of the nonce as this would otherwise blur the distinction between revealed and honest keys.

When the adversary yields a colliding ciphertext with two distinct valid explanations, there is one more condition to check before the adversary is considered to have won. That is, the `chk` function is ran on the keys of the explanations. This function is the *collision check* function and relies on a parameter of the CAE game, \mathbf{t} , which we refer to as the *collision type*. For any key K_i in the game, the key can either be *corrupted*, *revealed*, or *honest*. A key is corrupted when it is added to the game through the corruption oracle COR and thus part of the set K_c . A key is revealed when the adversary learns of it through the reveal oracle REV and thus part of the set K_r . If the key is part of neither set, meaning it was chosen uniformly at random and unaffected by the adversary, then it is considered honest. Whether keys are corrupted, revealed, and honest are represented by \mathbf{X} , $\mathbf{1}$, and $\mathbf{0}$ bits respectively. Six different types of collisions arise from these types of keys. The parameter \mathbf{t} is a two-bit string that describes the kind of collision the adversary may win with.

Finally, the advantage of an adversary \mathcal{A} attacking the CAE-security of an nAE scheme Π in regards to a collision type \mathbf{t} is quantified as $\mathbf{Adv}_{\Pi, \mathbf{t}}^{\text{cae}}(\mathcal{A}) = \Pr[\text{CAE}_{\Pi, \mathbf{t}}^{\mathcal{A}} \rightarrow 1]$. When discussing CAE-security with a specific type of collision, we denote the collision with a subscript i.e. $\text{CAE}_{\mathbf{xx}}$ -security.

OTHER COMMITTING NOTIONS. Most other committing AE definitions focused on cases where the adversary has control over both keys when creating a colliding ciphertext, which would be a corrupted-corrupted (or $\mathbf{t} = \mathbf{xx}$) collision [2, 7, 11]. Farshim et al. consider one definition of key-robustness, called *semi-full robustness*, where the adversary is asked to come up with a ciphertext that decrypts under an honest key and a key that it knows (what we would call a $\mathbf{01}$ -collision) [9]. Bellare gives another robustness notion for randomized symmetric encryption called *random-key robustness* in [4] that is comparable to the $\text{CAE}_{\mathbf{00}}$ notion and shows that authenticity implies random-key robustness. Our definitional framework can be tuned to consider these collisions and more, allowing flexibility when using the definition to model real systems.

<u>CAE_{II,t}</u>	
<pre> procedure Initialize() 00 for $i \in \mathbb{N}$ do $K_i \leftarrow \mathcal{K}$; $\mathbf{N}_i \leftarrow \emptyset$ 01 $\mathbf{S}, \mathbf{K}_c, \mathbf{K}_r \leftarrow \emptyset$ procedure Finalize() 10 ret $\exists (K_i, N, A, M, C), (K_j, N', A', M', C') \in \mathbf{S}$ s.t. 11 $(M \neq \perp \wedge M' \neq \perp) \wedge$ 12 $(C = C') \wedge$ 13 $(K_i, N, A, M) \neq (K_j, N', A', M') \wedge$ 14 $(\text{chk}(K_i, K_j) \vee \text{chk}(K_j, K_i))$ chk(K_i, K_j) 16 if $\mathbf{t} = 00 \wedge K_i \notin \mathbf{K}_c \cup \mathbf{K}_r \wedge K_j \notin \mathbf{K}_c \cup \mathbf{K}_r$ then ret 1 17 if $\mathbf{t} = 01 \wedge K_i \notin \mathbf{K}_r \cup \mathbf{K}_c \wedge K_j \notin \mathbf{K}_c$ then ret 1 18 if $\mathbf{t} = 0X \wedge K_i \notin \mathbf{K}_r \cup \mathbf{K}_c$ then ret 1 19 if $\mathbf{t} = 11 \wedge K_i \notin \mathbf{K}_c \wedge K_j \notin \mathbf{K}_c$ then ret 1 1A if $\mathbf{t} = 1X \wedge K_i \notin \mathbf{K}_c$ then ret 1 1B if $\mathbf{t} = XX$ then ret 1 1C ret 0 </pre>	<pre> procedure ENC(i, N, A, M) 20 if $K_i \notin \mathbf{K}_r \cup \mathbf{K}_c \wedge N \in \mathbf{N}_i$ 21 then ret \perp 22 $C \leftarrow \Pi.\mathcal{E}(K_i, N, A, M)$ 23 $\mathbf{S} \stackrel{\cup}{\leftarrow} \{(K_i, N, A, M, C)\}$ ret C procedure DEC(i, N, A, C) 30 $M \leftarrow \Pi.\mathcal{D}(K_i, N, A, C)$ 31 $\mathbf{S} \stackrel{\cup}{\leftarrow} \{(K_i, N, A, M, C)\}$ 32 ret M procedure REV(i) 40 $\mathbf{K}_r \stackrel{\cup}{\leftarrow} \{K_i\}$; ret K_i procedure COR(i, K) 50 $K_i \leftarrow K$; $\mathbf{K}_c \stackrel{\cup}{\leftarrow} \{K_i\}$ </pre>

Fig. 1. The CAE-security game. The encryption, decryption, reveal, and corruption oracles are on the right. On the left, the game finalization procedure depends on the collision check function chk , which in turn relies on the collision type parameter \mathbf{t} of the game. This function places restrictions on the keys that the adversary may win with.

Our definition considers the strongest level of misattributions. That is, we require that $(K_i, N, A, M) \neq (K_j, N', A', M')$. This means the adversary wins as long as one of the inputs to encryption differ when creating the colliding ciphertext. We call a cAE scheme that attends to all encryption inputs *fully committing*.

Most other works only consider sub-tuples. For example, the notion of key commitment from Albertini et al. only requires that $K_i \neq K_j$ from the adversary when it creates a collision [2]. They show that key commitment is important for several real world systems. Nonetheless, this definition does not capture colliding ciphertexts that are generated under the same key. In Sect. 4, we give a transform as efficient as theirs while protecting against misattributions over the entire tuple of encryption inputs. (In a way, our transform is more efficient as it does not need to re-key every encryption call).

Contemporary work from Bellare and Hoang considers fully committing cAE schemes as well as sub-tuples [5]. Their argument for fully committing schemes is to provide ease of use. Prior definitions required different inputs to be committed to achieve the different security goals demanded by their relevant applications. The designer of an application may not know exactly what they need to be committed. So, if full commitment is inexpensive, then one should aim to do so.

Nonetheless, we provide an alternative CAE-security game that considers weaker misattributions, which we present explicitly give in the full paper. It uses an additional parameter allowing the specification of which encryption inputs are important when considering misattributions. However, we do note our construction CTX presented in Sect. 4 achieves full commitment efficiently.

RELATIONSHIP WITH NAE SECURITY. Previously, DGRW18 show how to construct a ciphertext for AES-GCM in such a way that it decrypts validly under two different keys [7]. This shows that nae-secure schemes do not achieve CAE_{xx} -security. In fact, the attack presented by DGRW18 does not require the adversary to have full control over the keys; it is possible to do the attack with only knowledge of the keys, which means nae-secure schemes do not achieve CAE_{11} -security either.

We show that nAE schemes that are nae-secure in the multi-key sense—nae*-secure, are already CAE_{00} -secure. That is, when an adversary may not affect the keys in any way, it is already difficult to find colliding ciphertexts for nae-secure schemes.

Theorem 1. *Any authenticated encryption scheme Π that is nae*-secure is also CAE_{00} -secure. That is, for any adversary \mathcal{A} attacking the CAE_{00} -security of Π , there exists an adversary \mathcal{B} against the nae*-security of Π such that*

$$\text{Adv}_{\Pi}^{\text{cae}-00}(\mathcal{A}) \leq 3 \cdot \text{Adv}_{\Pi}^{\text{nae}*}(\mathcal{B}) + \frac{q_e^2}{2^{\tau+1}}$$

where q_e is the number of encryption queries made by \mathcal{A} and τ is the expansion of the scheme Π . Furthermore, \mathcal{B} makes the same number of encryption and decryption queries that \mathcal{A} makes. That is, \mathcal{B} makes $\Theta(q_e)$ encryption queries and $\Theta(q_d)$ decryption queries where q_d is the number of decryption queries made by \mathcal{A} .

Proof. Let \mathcal{A} be an adversary attacking the CAE_{00} -security of Π . We assume that \mathcal{A} is nonce-respecting and that it does not query output of encryption to decryption as it would already know the answers of those queries and those queries would not help \mathcal{A} in obtaining a win. We also assume that \mathcal{A} never calls the reveal or corruption oracles as it can only win with a collision on a pair of honest keys. We can construct an adversary \mathcal{B} attacking the nae-security of Π as follows. Adversary \mathcal{B} sets up the CAE game as described in Fig. 1, maintaining its own set S to keep track of query and response tuples. Whenever \mathcal{A} makes encryption or decryption queries, \mathcal{B} queries its own encryption and decryption oracles to provide a response for \mathcal{A} . When \mathcal{A} terminates, \mathcal{B} checks S to see if

\mathcal{A} has created a colliding ciphertext. If it has, then \mathcal{B} returns 1. Otherwise, it returns 0.

Consider the three different ways that \mathcal{A} can add a winning ciphertext and its associated explanations to \mathbf{S} . Either they were added through two decryption queries, an encryption and a decryption query, or two encryption queries. Let E_1, E_2, E_3 be the events that those yielded \mathcal{A} a win respectively. As these three events are all the ways to win, \mathcal{A} 's advantage is upper-bounded by the sum of the probabilities that each of these occur.

We bound the probabilities of each event by examining what happens when \mathcal{B} 's oracles are real or fake. For E_1 it is impossible for a winning tuple to be added to \mathbf{S} when \mathcal{B} 's decryption oracle is fake as that oracle only ever returns \perp and a winning tuple must have a valid message. As such, \mathcal{B} only ever returns 0. However, if \mathcal{B} 's oracle is real, then \mathcal{B} will return 1. Hence, $\Pr[E_1] \leq \mathbf{Adv}_{\mathcal{B}}^{\text{nae}}$.

For E_2 a fake decryption oracle for \mathcal{B} makes winning through this event impossible by the same reasoning as that of E_1 , meaning \mathcal{B} only returns 0 here as well. Similarly, \mathcal{B} will return 1 if its oracles are real in this event. As such, the probability follows: $\Pr[E_2] \leq \mathbf{Adv}_{\mathcal{B}}^{\text{nae}}$.

For E_3 , \mathcal{B} can return 1 with a fake encryption oracle so long as \mathcal{A} gets a collision through its encryption queries. We can get the probability that this occurs by a birthday bound on the number of encryption queries made by \mathcal{A} . The birthday bound is over the random ciphertexts generated by the fake encryption oracle. With a real encryption oracle, \mathcal{B} always returns 1. This gives the probability $\Pr[E_3] \leq \mathbf{Adv}_{\mathcal{B}}^{\text{nae}} + \frac{q_e^2}{2^{\tau+1}}$. \square

4 The CTX Construction

THE CTX SCHEME. Recall that a cAE scheme is *fully committing* if it commits to the key, nonce, AD, and message and not some subset of them. We say that a scheme is *efficient* if its cost of getting cAE security on top of nAE security is independent of the message length. We call a scheme *strong* if it achieves CAE_{xx} security. Our CTX construction is fully committing, efficient, and strong.

Let $\Pi = (\mathcal{E}, \mathcal{D})$ be a tag-based nAE scheme. That is, ciphertexts it outputs consist of a ciphertext core C and an authentication tag T . We assume that the encryption algorithm \mathcal{E} can be split into two independent algorithms \mathcal{E}_1 and \mathcal{E}_2 such that on inputs K, N, A, M , \mathcal{E}_1 produces the core C and \mathcal{E}_2 outputs the tag T . The core C is the same length as M . As such, \mathcal{E}_1 is bijective when K, N, A are fixed. The inverse of \mathcal{E}_1 is then decryption's subroutine \mathcal{D}_1 , which takes in K, N, A and just the core C , and outputs M . That is, $\mathcal{D}_1(K, N, A, \mathcal{E}_1(K, N, A, M)) = M$. Common schemes like GCM and OCB satisfy these structural demands.

From such an nAE scheme Π and a collision-resistant hash function H , we can construct a CAE_{xx} -secure cAE scheme, $\text{CTX}[\Pi, H]$. CTX's main mechanism is hashing the authentication tag T along with K, N, A into a new tag T^* . This effectively makes T^* function as the nAE authenticity check and a commitment to K, N, A . The name CTX captures the scheme's ciphertext structure, which is

$\text{CTX}.\mathcal{E}(K, N, A, M)$	$\text{CTX}.\mathcal{D}(K, N, A, C)$
20 $C \leftarrow \Pi.\mathcal{E}_1(K, N, A, M)$	30 $C \parallel T \leftarrow \mathcal{C}$
21 $T \leftarrow \Pi.\mathcal{E}_2(K, N, A, M)$	31 $M \leftarrow \Pi.\mathcal{D}_1(K, N, A, C)$
22 $T^* \leftarrow H(K, N, A, T)$	32 $T' \leftarrow \Pi.\mathcal{E}_2(K, N, A, M)$
23 ret $C \parallel T^*$	33 if $T \neq H(K, N, A, T')$ then ret \perp
	34 ret M

Fig. 2. A CAE_{xx} -secure cAE scheme $\text{CTX}[\Pi, H]$ built from a tag-based nAE scheme Π and a collision-resistant hash function H . The nAE encryption and decryption algorithms can be broken down into \mathcal{E}_1 , \mathcal{E}_2 , and \mathcal{D}_1 . These create the ciphertext core, create the authentication tag, and recover the message from the core respectively.

a ciphertext core followed by a modified tag. The ‘X’ in the name suggests the scheme’s XX -security level. The scheme is presented in Fig. 2.

We claim that CTX is CAE_{xx} -secure as long as H is collision-resistant.

Theorem 2. *Let $\Pi = (\mathcal{E}, \mathcal{D})$ be a tag-based nAE scheme and let H be a collision-resistant hash function. Let $\mathcal{E}_1, \mathcal{E}_2, \mathcal{D}_1$ be the algorithms used by Π to encrypt messages into ciphertext cores, create authentication tags, and decrypt cores into messages respectively. Let $\text{CTX}[\Pi, H]$ be an nAE scheme constructed from Π and H as described in Fig. 2. Then, for any adversary \mathcal{A} attacking the CAE_{xx} -security of Π' , there exists an adversary \mathcal{B} , explicitly given in the proof of this theorem and depending only on \mathcal{A} as a black-box, such that*

$$\text{Adv}_{\text{CTX}}^{\text{cae-xx}}(\mathcal{A}) \leq \text{Adv}_H^{\text{col}}(\mathcal{B}).$$

We give the proof for Theorem 2 in the full paper.

DISCUSSION. From Theorem 2, we see that the CAE_{xx} -security of CTX is bounded by the collision-resistance of the hash function it employs. One can break this with about $2^{\mu/2}$ operations doing a birthday-attack, which is why we recommend having CTX tag length be 160-bits over, say 128-bits. This raises a question: Can one lower the security requirement (something weaker than CAE_{xx}) and avoid the birthday bound?

The answer is that with CTX , you cannot unless there are further assumptions made of the nAE scheme Π that it uses. There exists an attack on the CAE_{ox} -security of CTX using a birthday attack under standard assumptions on Π . We explicitly describe this attack in the full paper.

It remains to show that CTX remains nAE-secure after its transform. We do so in the random oracle model (ROM), denoting CTX as $\text{CTX}[\Pi]$ (as opposed to $\text{CTX}[\Pi, H]$) when it is in the ROM. The privacy and authenticity notions in Theorem 3 can be attained by breaking the nAE security notion in 2 apart and can be found in [18]. We prove Theorem 3 in the full paper.

Theorem 3. Let $\Pi = (\mathcal{E}, \mathcal{D})$ be a tag-based nAE scheme with an expansion of τ . Let $\text{CTX}[\Pi]$ be the scheme described in Fig. 2. Fix an integer $\delta \geq 0$. Then, in the random oracle model, for any adversary \mathcal{A}_1 attacking the privacy of CTX , we can construct nonce-respecting (explicitly given) adversaries \mathcal{B}_1 and \mathcal{B}_2 attacking the privacy of Π such that

$$\text{Adv}_{\text{CTX}[\Pi]}^{\text{priv}}(\mathcal{A}_1^H) \leq \text{Adv}_{\Pi}^{\text{priv}}(\mathcal{B}_1) + \text{Adv}_{\Pi}^{\text{priv}}(\mathcal{B}_2) + \frac{q_H}{2^{\delta+\tau}}$$

where q_H is the number of random oracle queries made by \mathcal{A}_1 . Let q_e be the number of encryption queries made by \mathcal{A}_1 . Then \mathcal{B}_1 also makes q_e queries to its own encryption oracle and \mathcal{B}_2 makes $q_e + 1$ such queries.

Furthermore, for any adversary \mathcal{A}_2 attacking the authenticity of CTX , there exists adversary \mathcal{B}_3 attacking the authenticity of Π with advantage

$$\text{Adv}_{\text{CTX}[\Pi]}^{\text{auth}}(\mathcal{A}_2^H) \leq \text{Adv}_{\Pi}^{\text{auth}}(\mathcal{B}_3) + \frac{1}{2^\mu}$$

where μ is the output length of the random oracle. We give \mathcal{B}_3 explicitly in the proof. If \mathcal{A}_2 makes q_e encryption oracle queries, then \mathcal{B}_3 makes $q_e + 1$ queries to its own oracle.

Note that the last term in the privacy bound $\frac{q_H}{2^{\delta+\tau}}$ can be made small by choice of δ , so it will not result in much loss.

5 Commitment Security of GCM and OCB

Prior work from GLR17 and DGRW18 has shown that it is possible to construct a colliding ciphertext with GCM when the attacker has control of both keys [7, 11]. DGRW18 mentions that with their attack, control over the keys is not necessary, only knowledge of the keys is. Here, we show that it is possible for an attacker to create a colliding ciphertext with knowledge of only one key. That is, there exists an attack that violates the CAE_{01} -security of GCM. As GCM is nae-secure, this attack means that nae-security does not imply CAE_{01} -security.

A SIMPLE NAE SCHEME. Before we present the attack, consider a simple nAE scheme $\text{NAE}[G, H]$ built on a PRG G and a MAC H . The definition of $\text{NAE}[G, H]$ is given in Fig. 3. In our pseudocode, we write $S[0..n]$ to denote a substring of the bitstring S starting from the 0th bit to the n th bit.

However, NAE is vulnerable to a variety of CAE attacks given that the MAC H is *targetable*. Suppose that the key K used for computing H is known and there exists an arbitrary target tag T that an adversary is interested in producing. We call H targetable if there exists a target function target that takes in K and T and outputs a message M such that $H(K, M) = T$. We say that H is *prefix-targetable* if there exists a prefixed target function target that may also take in an additional argument C , a prefix, such that $H(K, C \parallel M) = T$. GHASH, the MAC used by GCM, is prefix-targetable, and we will show how shortly.

$\text{NAE.E}(K, N, A, M)$	$\text{NAE.D}(K, N, A, C)$
00 $K_1 \parallel K_2 \leftarrow K; P \leftarrow G(K_1, N)$	10 $K_1 \parallel K_2 \leftarrow K; C \parallel T \leftarrow C$
01 $C \leftarrow M \oplus P[0.. M - 1]$	11 if $H(K_2, C \parallel A) \neq T$ then ret \perp
02 $T \leftarrow H(K_2, C \parallel A)$	12 $P \leftarrow G(K_1, N)$
03 ret $C \parallel T$	13 $M \leftarrow C \oplus P[0.. M - 1];$ ret M

Fig. 3. A simple nAE scheme given a PRG G and a MAC H . GCM has a comparable structure if one considers the counter-mode operations as the PRG and GHASH as the MAC.

ATTACK ON GCM. The simple nAE scheme described has structure similar to GCM. For concreteness, we assume the blockcipher GCM employs E has a block size of 128 bits. GCM uses E in counter mode with the nonce as part of the initial counter in order to generate a one-time pad. This acts like the PRG that the simple scheme uses. For a key K , GCM uses $K' = E_K(0^{128})$ when computing its MAC, GHASH. For a ciphertext C , the tag $T = H(K', A, C) \oplus E_K(N \parallel 0^{31}1)$ where A is the AD, C is the ciphertext, N is the nonce and H is the GHASH function. We follow the GCM specification of [14, 15].

GHASH works by computing a polynomial over the field $\text{GF}(2^{128})$ using $E_{K'}(0^{128})$ as the variable and the ciphertext and AD blocks as coefficients. By block, we mean blocks of b bits that can be used as input into a blockcipher. If the last block isn't a full 128 bits, GCM pads it with zeroes until it is. Let there be c ciphertext blocks and a AD blocks in the ciphertext and AD. Let len be a function where given some input, it outputs a 64-bit representation of said input. Let $P = E_{K'}(0^{128})$. Then GHASH is computed as follows (addition and multiplication done over $\text{GF}(2^{128})$):

$$\text{GHASH}(K', A, C) = \left[\sum_{i=1}^a A_i \cdot P^{a+c+2-i} \right] + \left[\sum_{i=1}^c C_i \cdot P^{c+2-i} \right] + (\text{len}(A) \parallel \text{len}(C)) \cdot P \quad (1)$$

And the tag T is finalized as:

$$T = \text{GHASH}(K', A, C) \oplus E_{K'}(N \parallel 0^{31}1) \quad (2)$$

where N is the nonce.

Observe that the entire MAC is prefix-targetable as if one knows K' and T , one can compute $A = \text{ptarget}(K', T, C)$ for a ciphertext C by evaluating the polynomial. Explicitly, we can solve for a single block AD A as follows:

$$A = \left[T \oplus E_{K'}(N \parallel 0^{31}1) + (\text{len}(A) \parallel \text{len}(C)) \cdot P + \left[\sum_{i=1}^c C_i \cdot P^{c+2-i} \right] \right] \cdot (P^{c+2})^{-1} \quad (3)$$

Once we can compute A with the prefix-targeting function, we have an CAE_{01} attack, call it adversary \mathcal{A} , as follows: \mathcal{A} selects an arbitrary nonce N and AD A ; \mathcal{A} queries its encryption oracle, asking for the encryption of a string of 0's of length m under K_0 , N , and A , and receives $C \parallel T$ back; \mathcal{A} uses its reveal oracle to learn K_1 ; \mathcal{A} computes a one-time pad P in the style of GCM using K_1 and the nonce N ; \mathcal{A} computes a message M' as the xor of P and C ; \mathcal{A} uses the prefix-target function $\text{ptarget}(K', T, C)$ as shown in Eq. 3 where K' is the blockcipher E applied to 0^{128} with K_1 (how GHASH is keyed) and acquires an AD A' ; \mathcal{A} queries its encryption oracle with K_1 , N , A' , M' and receives a winning collision on $C \parallel T$. This attack on GCM prove that nae-security does not imply even CAE_{01} -security as GCM is nae-secure.

While Eq. 3 computes a single AD block that allows us to obtain a target tag, this is actually not restrictive. An attack can use an arbitrary AD A , perhaps with actually relevant header information, and search for a single block A' that they can add to that AD to satisfy the equation. To capture this in terms of prefix-targeting, one would compute $A' = \text{ptarget}(K', T, A \parallel C)$ instead of $\text{ptarget}(K', T, C)$. Keep in mind that the dummy block can be placed anywhere in A , but we limit our description to prepending for simplicity.

ATTACK ON OCB. We now turn to performing a CAE_{01} attack on OCB. We follow the specification of OCB as described in [12].

During encryption, OCB computes an offset Δ for each message block using the key and the nonce. This offset is xor-ed with each message block before being processed by a blockcipher under the key. The output of the blockcipher is then xor-ed with the offset again, finalizing a ciphertext block. Since the adversary has a revealed key K_j and a nonce of its choice N , it can freely compute the offsets for each block. This allows it to decrypt the target colliding ciphertext $C \parallel T$, where C is the ciphertext core and T is the authentication tag, into some message M . The next step requires the adversary to ensure that T verifies for M under K_j and N . In OCB, tag T is generated first with a checksum that consists of an xor over all message blocks. The adversary can do this over the bogus message M it got from decryption. This checksum is then xor-ed with a special offset, again computable with knowledge of key and nonce, before being processed by the blockcipher. This output, F , is xor-ed with a block called “auth” which finalizes the tag T .

The adversary then needs $\text{auth} = F \oplus T$ for T to that remain valid with K_j , N , M , C . To do so, it has a choice of AD A . OCB computes auth by computing offsets for each block of AD, xor-ing the offsets and blocks together, and applying the blockcipher on the results (a process identical to how the message blocks are processed with the exception of how the offsets are initialized). Each of these blocks are then xor-ed with each other, finalizing a single block auth . To acquire an A that finishes the attack, the adversary deciphers $F \oplus T$ with the blockcipher, xors the result with the appropriate offset, and uses that for its final query $\text{ENC}(j, N, A, M)$. The attack is described in code in Fig. 4.

	$\mathcal{A}_{\text{OCB},01}^{\text{CAE}}$
20	$N \leftarrow \mathcal{N}; M \leftarrow \{0, 1\}^m; C \parallel T \leftarrow \text{ENC}(0, N, \varepsilon, M)$
21	$C_1 \parallel C_2 \parallel \dots \parallel C_n \leftarrow C$ where $ C_i = b$ for all $i \in [1..n]$
22	$K \leftarrow \text{REV}(1); \Delta \leftarrow \text{init}(K, N)$
23	for $i = 1$ to n do
24	$\Delta \leftarrow \text{incr}_i(\Delta)$
25	$P \leftarrow C_i \oplus \Delta; M'_i \leftarrow E_K^{-1}(P) \oplus \Delta$
26	$\text{chk} \leftarrow M'_1 \oplus M'_2 \oplus \dots \oplus M'_n$
27	$F \leftarrow \text{chk} \oplus \text{incr}_s(\Delta); F \leftarrow E_K(F)$
28	$\Delta \leftarrow \text{incr}_1(0^{128})$
29	$\text{auth} \leftarrow T \oplus F; A \leftarrow E_K^{-1}(\text{auth}) \oplus \Delta$
2A	$\text{ENC}(1, N, A, M'_1 \parallel \dots \parallel M'_n)$

Fig. 4. An CAE_{01} attack on OCB. For simplicity, this attack assumes that the length of the ciphertext is a multiple of the blockcipher E 's block size b .

Like the attack on GCM, it should be noted that the attack is not limited to a single AD block. An adversary may select an arbitrary AD A' that it wants to use to mount the attack. It can then compute a single dummy block B' to append to the end of A' to create an AD that validates decryption. Specifically, the adversary first computes F as described above. Then it computes a value auth' over the blocks of A' . The value $B = \text{auth}' \oplus F \oplus T$ will then be the “enciphered” AD block corresponding to B' . So the adversary only needs to decipher B and apply the appropriate offset to compute B' . The final AD it uses is $A = A' \parallel B'$.

6 Other Committing AE Notions

Here we describe the committing AE notions of previous authors and highlight their differences. A summary of each of their definitions can be found in Table 1. The first to study committing encryption in the AE setting was Farshim, Orlandi, and Roşie (FOR17) [9]. Calling the property *key-robustness*, FOR17 give a set of definitions capturing different adversarial behaviors that can result in the misattribution of a ciphertext. Their strongest notion, *full robustness*, requires an adversary to produce two keys and a ciphertext (K, K', C) such that C decrypts validly under both keys. It needs to be noted that FOR17 study randomized AE without AD support.

Table 1. A comparison of the subtly different definitions in CAE literature.

Paper	AE Variant	Committing AE Definition
FOR17 [9]	Probabilistic AE, No AD support	<i>Full robustness</i> - \mathcal{A} finds (K, K', C) s.t. decryption of C with both keys is successful
GLR17 [11]	Probabilistic and deterministic AEAD	<i>Receiver-binding</i> - \mathcal{A} finds $((K, A, M), (K', A', M'), C)$ s.t. decryption of C with both sub-tuples is successful and $(A, M) \neq (A', M')$
DGRW18 [7]	Probabilistic AEAD	<i>Strong Receiver-binding</i> - Same as receiver-binding except $(K, A, M) \neq (K', A', M')$
ADGKLS20 [2]	Deterministic AEAD	<i>Key-commitment</i> - \mathcal{A} finds $((K, N, A, M), (K', N', A', M'), C)$ through ENC, DEC queries s.t. $K \neq K'$ and $M, M' \neq \perp$
BH22 [5]	Deterministic AEAD + misuse-resistant AE	<i>CMT(D)_s-ℓ security</i> - \mathcal{A} finds $(K_1, N_1, A_1, M_1), \dots, (K_s, N_s, A_s, M_s)$ such that what is committed from each tuple is distinct. The parameter ℓ specifies “what is committed.”
This Paper	Deterministic AEAD	<i>CAE_t-security</i> - \mathcal{A} finds $((K, N, A, M), (K', N', A', M'), C)$ through ENC, DEC queries s.t. $(K, N, A, M) \neq (K', N', A', M')$ and $M, M' \neq \perp$. The parameter t specifies how \mathcal{A} interacts with the keys

An interesting result from FOR17 is that security for such AE schemes implies *semi-full robustness*. In this notion, two keys are generated uniformly at random and one of them is shown to the adversary. With the help of encryption and decryption oracles for the hidden key, the adversary must find a ciphertext that validly decrypts under both keys. This definition is comparable to our CAE₀₁ notion, where the adversary must find a misattribution with a revealed and an honest key. One of our results is the existence of CAE₀₁-attacks against AES-GCM and OCB, which implies that nAE security does not grant 01-security. This seemingly contradicts FOR17’s result of semi-full robustness because their analysis is for AE schemes without AD support.

In the same year as FOR17, Grubbs, Lu, and Ristenpart (GLR17) study message franking, which as they describe it, is the *verifiable* reporting of abusive messages in encrypted messaging systems [11]. To accomplish this goal, they use committing AE, focusing on randomized AE with AD support (AEAD) as it is more applicable to current encrypted messaging systems. In their model, there is a sender, a receiver, and a third party that verifies abuse reports. Every ciphertext comes with a commitment tag that serves as a commitment to the message and AD. Decryption produces an opening for the commitment alongside recovering the message. Their committing AE notion adds an additional verification algorithm as it is the third party’s role to verify the commitment using that opening. We conflate their decryption and verification algorithms for ease of discussion and comparison to other notions.

There are several parts of their committing AE notion that make it difficult to compare as they tend to other things besides preventing misattributions. The part that attends to misattributions is their notion of *receiver binding*. This notion asks the adversary to find a ciphertext C and two tuples

$(K, A, M), (K', A', M')$ such that decrypting C with those keys and ADs results in those (valid) messages. The adversary must do so in a way such that $(A, M) \neq (A', M')$. This definition does not prevent the possibility of an adversary finding two keys that can validly decrypt C into M using A .

Dodis, Grubbs, Ristenpart, and Woodage (DGRW18) [7] extend GLR17's receiver binding to *strong receiver binding*. This notion accounts for the key to address the way receiver binding does not. One can argue that strong receiver binding commits to all encryption inputs for randomized AEAD. As a building block for cAE, DGRW18 introduce a new primitive *encryption* that serves as a one-time use, deterministic encryption and commitment of a message.

One goal that GLR17 and DGRW18 consider that other works do not (including ours) is that of *multiple-opening security*. This security notion allows different ciphertexts encrypted under the same key to be “opened” and verified without jeopardizing the security of unopened ciphertexts. This is particularly useful in the message franking context as it allows a receiver to report a ciphertext to the verifying party without having to reveal the secret key, which would ruin the security of all other ciphertexts sent under that key.

Working with deterministic AEAD, Albertini, Duong, Gueron, Kölbl, Luykx, and Schmiege (ADGKLS20) [2] define their security goal as *key-commitment*. The adversary, in this notion, is tasked with finding a ciphertext C and two “explanations” $(K, N, A, M), (K', N, A', M')$ such that the messages are valid and $K \neq K'$.

Bellare and Hoang (BH22), in a contemporary work, target fully committing schemes [5]. They attend to deterministic AEAD as well as misuse-resistant AE with encryption inputs K, N, A, M . Their committing security notion is $\text{CMT}(\mathcal{D})_{s-\ell}$ where s is an integer and $\ell \in \{1, 3, 4\}$. The presence or absence “D” denotes whether adversary is tasked with finding multiple decryption inputs that validly decrypt the same ciphertext or multiple encryption inputs that encrypt to the same ciphertext. The parameter ℓ determines what is committed: $\ell = 1$ denotes just the key, $\ell = 3$ denotes everything but the plaintext, and $\ell = 4$ denotes full commitment. Comparatively, our cAE definition presented in Sect. 3 does not allow for tweaking for commitments of sub-tuples of inputs, but the alternative framework described in the full paper does. The s parameter generalizes their definition to capture misattributions with more than two valid explanations—what they call *multi-input committing security*. That is, s is the number of distinct (K, N, A, M) tuples the adversary needs to find that encrypt to the same C . While $s = 2$ implies all $s \geq 2$, Bellare and Hoang motivate this dimension of their definition by giving schemes where bounds on adversarial advantage improve as s grows. They are the first to study misuse-resistant AE and multi-input committing security in the cAE space.

CONSTRUCTIONS. We describe a number of selected constructions from the above works. These constructions, each satisfying the committing AE notion defined in the work of their origin, are presented in Table 2.

Recall that FOR17 are in the probabilistic AE setting without associated data. Their construction $\text{EtM}[\mathcal{E}, H]$ creates a tag that provides authenticity while

Table 2. A comparison of selected constructions targeting their respective cAE security goals. *Not a committing AE scheme, but closely related.

Construction	Description
EtM[\mathcal{E}, H] [9]	\mathcal{E} is AE scheme, H is CR MAC. Encrypt M w/ \mathcal{E} under key K_e to get C . Get T by using MAC w/ key K_h on (C, K_e) . Output $C \parallel T$
CEP[G, F, F^{cr}] [11]	G is PRG. F, F^{cr} are PRFs. F^{cr} is CR Use G w/ K and N to get K_0, K_1, P . Use $P \oplus M$ to get C_1 . Use F^{cr} w/ K_0 on A, M for C_2 . Use F w/ K_1 on C_2 to get T . Output $C_1 \parallel T \parallel C_2$
HFC* [7]	HFC is an <i>encryption</i> scheme built from a compression function and a padding scheme. DGRW18 show a simple transform that promotes an encryption scheme into a cAE scheme
CommitKey _{IV} [2]	\mathcal{E} is nAE scheme. F_0, F_1 are independent CR PRFs. Get K_e from using F_0 w/ K on nonce N . Get K_c from using F_1 w/ K on nonce N' . Use \mathcal{E} on N, A, M to get C . Output $C \parallel K_c$
UtC[\mathcal{E}, F] [5]	\mathcal{E} is nAE scheme. F is <i>committing</i> PRF. Get (P, L) from $F(K, N)$. Get C from $\mathcal{E}(L, N, A, M)$. Output $P \parallel C$
HtE[\mathcal{E}, H] [5]	\mathcal{E} is a <u>CMT-1</u> nAE scheme. H is a CR function. Get L from $H(K, (N, A))$. Output $\mathcal{E}(L, N, \varepsilon, M)$

serving as a commitment to the encryption key as well. This is comparable to how CTX's tag provides authenticity while committing to all of K, N, A, T .

The scheme CEP[G, F, F^{cr}] is the deterministic AEAD construction from GLR17. It makes two passes over the message—one to encrypt it one-time-pad-style using output from the PRG G and the other to commit to the message and AD using the collision-resistant PRF F^{cr} . The ciphertext output is expanded by both a tag for authenticity and a commitment—the output lengths of the two PRFs. Comparatively, our CTX construction requires no passes over the message and would typically expand ciphertexts from a 128-bit authentication tag to a 160-bit hash function output that gives both cAE security and nAE authenticity. An advantage of CEP is that one can verify the commitment without revealing the encryption key. One only needs to reveal K_0 to do so. This is in line with GLR17's additional goal of multiple opening security.

DGRW18 had similar goals to GLR17 as they both investigated committing AE for the purpose of message franking. They propose a new primitive, *encryptment*, that we do not describe in detail here. Encryptment is a primitive that simultaneously encrypts and commits a message and is one-time use. They give

a concrete encryption scheme HFC that uses a compression function and a padding scheme. They give a simple transform that builds a cAE scheme out of an encryption scheme and a probabilistic nAE scheme. We note that HFC requires a pass over the message to apply encrypt and commit it.

The CommitKey scheme from ADGKLS20 comes in four flavors. We describe the variant CommitKey_{IV} here. It consists of an nAE scheme and two independent collision-resistant PRFs. On encryption, the PRFs are used on the nonce to generate an encryption key and a “key-commitment.” The encryption key is then used to perform routine nAE encryption on the message, producing a ciphertext. Encryption returns both the ciphertext and the key-commitment. It commits only to the key and as such, does not require any passes over the plaintext. With this scheme, it is possible to find a misattribution where different AD lead to valid decryptions.

One can argue that this kind of misattribution may not be impactful to real-world systems. But CTX protects against these misattributions as well and without giving up efficiency. In fact, CTX enjoys the efficiency benefit of not having the re-key with each message encrypted.

That argument is specious, in any case. It is difficult for designers of systems to know exactly what needs to be committed to achieve their security goals. GLR17 and DGRW18 showed message franking requires the commitment of the header and message. ADGKLS20 found that various real-world systems (key management services, envelope encryption, and Subscribe with Google [3]) had potential vulnerabilities from lack of key commitment. It is not always clear what exactly needs to be committed, so a scheme that can inexpensively commit to everything would provide a way to cover all bases for application designers.

Bellare and Hoang give a fully committing cAE construction that builds off of one that only commits to a key. Their UtC construction only commits to the key (CMT-1 secure going by their notions). It uses a primitive they call a *committing PRF* which informally outputs a commitment to the key and the PRF input along with the conventional PRF output. They describe an efficient committing PRF in their paper.

To promote a CMT-1 secure scheme to a fully committing (CMT-4) one, BH22 give the HtE transform. Like our CTX construction, the application of HtE to UtC commits to everything without having to make a pass over the plaintext beyond encrypting it. The ciphertext expansion of BH22’s transform however is expected to be at least 128-bits—the block length of the blockcipher that their committing PRF employs. On the other hand, CTX is expected to replace a conventional nAE tag, say 128-bits, to a 160-bit tag that provides both nAE authenticity and the commitment to all encryption inputs. This would be a 32-bit expansion compared to the expansion by a full block.

Acknowledgements. The authors gratefully acknowledge insightful input from Mihir Bellare and Tom Shrimpton. Thanks to Mihir specifically for the discussion on CTX’s security level with notions weaker than CAE_{xx} -security as well as CTX’s privacy and authenticity.

This work was funded by NSF grant CNS 1717542. Many thanks to the NSF for their years of support.

References

1. Abdalla, M., Bellare, M., Neven, G.: Robust encryption. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 480–497. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_28
2. Albertini, A., Duong, T., Gueron, S., Kölbl, S., Luykx, A., Schmiege, S.: How to abuse and fix authenticated encryption without key commitment. Cryptology ePrint Archive, Report 2020/1456 (2020). <https://eprint.iacr.org/2020/1456>
3. Albrecht, J.: Introducing subscribe with google, March 2018. <https://blog.google/outreach-initiatives/google-news-initiative/introducing-subscribe-google/>
4. Bellare, M.: A concrete-security analysis of the apple PSI protocol, July 2021. https://www.apple.com/child-safety/pdf/Alternative_Security_Proof_of_Apple_PSI_System_Mihir_Bellare.pdf
5. Bellare, M., Hoang, V.T.: Efficient schemes for committing authenticated encryption. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 845–875. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07085-3_29
6. Bellare, M., Tackmann, B.: The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 247–276. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_10
7. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: from invisible salamanders to encryption. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 155–186. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_6
8. Farshim, P., Libert, B., Paterson, K.G., Quaglia, E.A.: Robust encryption, revisited. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 352–368. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_22
9. Farshim, P., Orlandi, C., Rosie, R.: Security of symmetric primitives under incorrect usage of keys. Cryptology ePrint Archive, Report 2017/288 (2017). <https://eprint.iacr.org/2017/288>
10. Gertner, Y., Herzberg, A.: Committing encryption and publicly-verifiable signcryption. Cryptology ePrint Archive, Report 2003/254 (2003). <https://eprint.iacr.org/2003/254>
11. Grubbs, P., Lu, J., Ristenpart, T.: Message franking via committing authenticated encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 66–97. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_3
12. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21702-9_18
13. Len, J., Grubbs, P., Ristenpart, T.: Partitioning oracle attacks. Cryptology ePrint Archive, Report 2020/1491 (2020). <https://eprint.iacr.org/2020/1491>
14. McGrew, D., Viega, J.: The Galois/counter mode of operation (GCM). submission to NIST Modes of Operation Process, vol. 20, pp. 0278–0070 (2004)

15. McGrew, D.A., Viega, J.: The security and performance of the Galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30556-9_27
16. Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering generic composition. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 257–274. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_15
17. Rogaway, P.: Formalizing human ignorance. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 211–228. Springer, Heidelberg (2006). https://doi.org/10.1007/11958239_14
18. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) ACM CCS 2001: 8th Conference on Computer and Communications Security, Philadelphia, PA, USA, 5–8 November 2001, pp. 196–205. ACM Press