REGARDING ASSUMPTIONS MADE OF AUTHENTICATED ENCRYPTION

By

JOHN M. CHAN
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Phillip Rogaway, Chair

Mihir Bellare

Sam King

Committee in Charge

2022

For
Anh, Yin, and Eric.

Contents

Regarding Assumptions Made of Authenticated Encryption

Abstract

Authenticated encryption (AE) is a cryptographic primitive providing message privacy and authenticity simultaneously. From a standard definition of AE, several natural assumptions may arise. (1) An observer of an encrypted message learns nothing about its origin. (2) If one attempts to decrypt a ciphertext using a different key (or any other decryption input) from what was used at encryption, then decryption should fail. One might conclude the former from promises of privacy and the latter from promises of authenticity.

We observe that the validity of those assumptions do not follow from standard AE definitions of privacy and authenticity. For (1), when sending a ciphertext, there is typically other information sent along with it. This information, commonly referred to as *metadata*, can be message numbers that mark the message's position in a sequence or information that identifies the sender among other possibilities. For (2), it is possible to validly decrypt a ciphertext with the "wrong" arguments.

This dissertation's main contribution consists of definitions and constructions that address these assumptions. With respect to the first, it offers an AE variant, anonymous AE (anAE)—a primitive that folds all cryptographically relevant metadata directly into its ciphertexts. For the second, it furthers the study of committing AE (cAE)—a variant of AE that ensures that decryption of a ciphertext is only possible with the "correct" inputs. These two primitives provide security that users may have mistakenly assumed AE satisfied. Lastly, we give concrete constructions for these primitives along with their proofs of security.

Acknowledgments

First and foremost, I want to thank my mentor and friend, Phil Rogaway. Thank you not only for the years of technical guidance, but for shaping my world view. Perhaps the most valuable lesson you taught me over all those chats in your office (or in those Zoom calls and Wire calls riddled with connectivity issues in the latter half of this journey) is to remember to approach technical problems with a heart.

Many thanks to Mihir Bellare who guided me on my first steps into the world of cryptography. The early mornings in your classroom were among my most memorable and enjoyable moments at UCSD. Thank you also for your insights and input on the committing AE portion of this dissertation.

Thanks to Joël Porquet who encouraged me to keep pressing forward when I was met with uncertainty during this journey. Also thank you for the hellish first experience in teaching. It was a lot of fun.

And of course I need to thank the countless other people in my life who provided unending support. Many thanks to my mom and dad, Anh Huynh and Yin Chan; your support was especially fierce and relentless. Special thanks to my brother, Eric Chan; our conversations never fail to challenge and amuse me. Thank yous to Mark Celones, Mark Nguyen, Yvie Tsui, and Josh Kohlbecker; all of whom I could write dissertation-length letters of gratitude for as our friendships have brought me nothing but joy. Lastly, maybe the real doctorate is the friends we made along the way; special thanks to Keshav Dasu, Sandra Bae, and Meghann Ma—thank you all for helping me find myself on this arduous journey.

CHAPTER 1

Introduction

As a primitive that provides both privacy and authenticity, *authenticated encryption* (AE) is a crucial component in securing many modern communication systems. Take, for example, the transport layer security (TLS) protocol [**54**]. This protocol is used to facilitate communications between clients and servers on the Internet in a secure fashion. In its latest version, the record protocol of TLS strictly employs *authenticated encryption with associated data* (AEAD) algorithms to protect records sent between servers and clients. We say *strictly* as all other symmetric encryption algorithms have since been deprecated. As such protocols are widely used, it is important to understand what exactly it means for a protocol to use AEAD. What are the limits to standard definitions that we use to capture the security of these AE schemes?

In this dissertation, we focus on nonce-based authenticated encryption with associated data (nAE) schemes. A conventional way to define nAE schemes consists of an *encryption* algorithm and a *decryption* algorithm. Occasionally, they are defined with a *key generation* algorithm as well. For inputs, nAE encryption takes in a *key*, a *nonce*, some *associated data* (AD), and a *message.* It outputs a *ciphertext.* For nAE decryption, such a ciphertext is used as input along with a key, nonce, and AD to produce either a message or some error indicating decryption failure.

When one says that such an nAE scheme is secure, one means that it has both *privacy* and *authenticity.* Informally, privacy means that the ciphertexts produced by the scheme are indistinguishable from random bits. This implies that an adversary that acquires a ciphertext learns nothing from it. Authenticity, on the other hand, means that the ciphertexts are authentic—the message received is truly from the claimed sender. To paraphrase, the adversary cannot produce a ciphertext or modify one in any way such that the ciphertext decrypts without failure.

Although these descriptions are informal and thus imprecise, they capture ideas that demand precision. When one discusses the privacy and authenticity of AE, one should understand the limits of these properties. Any leaps of logic one might make can lead to incorrect assumptions about

AE. This dissertation looks at two such assumptions, one pertaining to privacy and the other to authenticity.

1.1. Assumptions on AE

One may assume, from privacy, that the phrasing "an adversary that acquires a ciphertext learns nothing from it" is absolute—that there are no cases in which the adversary might learn something. A problem with this assumption stems from something typically considered out of scope for AE. To elaborate, when a ciphertext C is made through the encryption of message M using key K, nonce N, and AD A, it is expected that decryption uses the same K, N, and A to recover M from C. How, then, is the decrypting party supposed to know what K, N, and A to use? The answer to this question is usually that it's not AE's job to answer. So, since it is now up to the higher level protocol, information related to K, N, and A are often sent alongside the ciphertext to the decrypting party. Using the datagram variant of TLS (DTLS) as an example, nonces used by the underlying nAE schemes are derived from sequence numbers, which are sent in the clear alongside the encrypted record [**55**]. Any observer of this protocol or a protocol handling nonces similarly would then learn a particular message's position in a sequence of messages.

This is then contrary to the assumption that AE privacy means absolutely no information is granted to adversaries. Certainly, one cannot hope to learn anything from communications that one cannot distinguish from random bits. But in order for the decrypting party to decrypt the ciphertext, other information is often sent along with the ciphertext. The ciphertext alone does not encompass the full communications. From the other information sent, the adversary can learn things about the communications. This is both a usability and a security concern. In the usability sense, why does the decrypting party need more than a ciphertext (and perhaps the knowledge of some potential keys for it) to decrypt? If we consider the *full ciphertext* to be everything that the decrypting party needs to decrypt (so the full communication), then we have a security concern as we may be sending the nonce or AD in the clear. If we do so, then the full ciphertext would likely not actually satisfy nAE privacy definitions.

To address these concerns, this work proposes a new primitive: *anonymous nonce-based authenticated encryption* (anAE). Encryption is unchanged from standard nAE—the user supplies a key, nonce, AD, and message to produce a ciphertext. However, decryption takes in the full ciphertext but only the full ciphertext. Furthermore, not only is decryption expected to recover the message from the full ciphertext, but the nonce, AD, and a handle representing the key that was used to produce it as well. To assist decryption in this tall task, we consider decryption to be stateful (maintaining at the very least a vector of keys that it shares with other communicants) and give it a handful of state manipulation algorithms. By only allowing decryption the singular input of a ciphertext, any necessary transmission of the nonce, AD, and the like falls within the scope of anAE. As a result, anAE provides the level of privacy that one may fallaciously expect from the standard nAE privacy definition.

The other assumption is one that may be drawn from standard nAE authenticity definitions. Suppose ciphertext C is the result of encrypting message M with some K, N, A. Now suppose that one tries to decrypt C using some K', N', A' where at least one of K', N', A' are different from their respective arguments used for encryption. If it was established that the scheme used satisfied conventional nAE authenticity, then one may wrongfully assume that decrypting in such a way would almost certainly fail. However, this is not the case. One can, for example, find two valid and distinct decryptions for an AES-GCM (a secure nAE scheme [**44**]) ciphertext so long as one has knowledge of two different keys to do so with [**29**].

Why does this assumption fall flat when nAE authenticity seemingly ought to prevent something like this from occurring? It is important to recognize that nAE security definitions themselves have their own assumptions. That is, nAE security definitions assume that the key the scheme uses is chosen uniformly at random and remains a secret to the adversary. Any security guarantees are null if the adversary learns the key or has control over the key. To execute the attack on AES-GCM described in [**29**], the attacker only needs knowledge of the keys.

Within the scope of nAE, there is nothing that prevents the construction of ciphertexts with multiple easily found decryptions. Generally, this is not seen as an undesirable thing. In fact, the property of *deniability* is sometimes viewed as desirable. An encryption scheme is deniable if an encrypting party is able to make a ciphertext appear as if it is the encryption of a different plaintext than it actually is [**25**].

The opposite, then, are schemes where given a ciphertext and a plaintext, one can be confident that the ciphertext emerged from that plaintext. Such schemes are called *committing encryption schemes*, wherein ciphertexts serve as commitments to plaintexts [**32**]. When casting this to the context of nAE, it is natural to ask if a ciphertext can serve as a commitment to all encryption inputs: the key, the nonce, and the AD as well as the plaintext. From that, a follow-up question: can one do so efficiently? This dissertation answers both of these questions (with a yes) by contextualizing committing encryption in the nAE setting as *committing authenticated encryption* (cAE). It merges prior cAE definitions that targeted different goals into a unifying framework then demonstrates how to transform a plain old nAE scheme into a cAE one simply and efficiently. Our definition of cAE covers the initially discussed assumption of AE authenticity. That is, by our definition, a secure cAE scheme produces ciphertexts where it is difficult to decrypt validly unless one uses the exact same key, nonce, and AD inputs that were used for encryption.

1.2. Dissertation Objectives and Overview

When using cryptographic primitives, it is important that one understands the limits to their security guarantees. This motivates our study of the aforementioned assumptions. We liken this to the study of *misuse-resistant AE* (mrAE) [**63**]. The privacy and authenticity properties of nAE are only guaranteed assuming that the user of the secure nAE scheme never repeats a nonce across their encryption calls. Users do not always realize this is the case, unfortunately. This gap between user comprehension and nAE definitions led to the studying of mrAE, a primitive that is forgiving of such user error.

Following this perspective, we take two misguided assumptions that users may have of nAE. We present a new primitive (anAE) and build upon an emerging one (cAE) wherein these primitives are AE variants that account for such assumptions. We hold the (somewhat obvious) opinion that when people make assumptions of a tool, they expect that tool to satisfy those assumptions when they use it. AE is no different. Users of AE may assume things of its privacy and authenticity, then proceed to build applications using it with those assumptions in mind. When it turns out that those assumptions aren't completely true, the results can be disastrous for the security of the application. Instead, users should turn to different primitives that satisfy the assumptions and needs they have for their application.

The above summed up leads to the thesis of this dissertation: Motivated by some incorrect assumptions users may have on the privacy and authenticity of authenticated encryption, we offer two primitives, anonymous nonce-based AE and committing AE, that capture the properties users initially may have thought AE satisfied. Moving forward, the dissertation is organized in such a way that the discussions of anAE and cAE are mostly disjoint. They will appear as follows:

Nonce-based AE with Associated Data. We first recall various definitions for nonce-based authenticated encryption with associated data (nAE) as it serves as both the intellectual and motivational starting point for both anAE and cAE. This primitive simultaneously provides message privacy and authenticity while allowing the provision of associated data (AD). The AD is historically understood to be data that requires authenticity, but not privacy. As an extensively studied primitive, nAE has a number of different definitions and we review several that will be useful in our analysis of anAE and cAE.

Anonymous Nonce-based AE. The customary formulation of authenticated encryption (AE) requires the decrypting party to supply the correct nonce with each ciphertext it decrypts. To enable this, the nonce is often sent in the clear alongside the ciphertext. But doing this can forfeit anonymity and degrade usability. Anonymity can also be lost by transmitting associated data (AD) or a session-ID (used to identify the operative key). To address these issues, we introduce anonymous AE, wherein ciphertexts must conceal their origin even when they are understood to encompass everything needed to decrypt (apart from the receiver's secret state). We formalize a type of anonymous AE we call anAE, *anonymous nonce-based AE*, which generalizes and strengthens conventional nonce-based AE, nAE. We provide an efficient construction for anAE, NonceWrap, from an nAE scheme and a blockcipher. We prove NonceWrap secure. While anAE does not address privacy loss through traffic-flow analysis, it does ensure that ciphertexts, now more expansively construed, do not by themselves compromise privacy.

Committing AE. We provide a strong definition for *committing* authenticated-encryption (cAE), as well as a framework that encompasses earlier, 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.

CONCLUDING REMARKS. We summarize the major contributions of this dissertation, then discuss some future work. Specifically, we discuss how one might strengthen our anAE security definition and whether NonceWrap would satisfy this definition. Our initial motivation for studying committing AE was the observation that the anAE definition could be strengthened.

1.3. Related Work

WORKS FROM AUTHOR. This dissertation is based on two papers from the author, referred to as CR19 and CR22:

- CR19 [**26**] appeared as "Anonymous AE" in *Advances in Cryptology - Asiacrypt 2019, Part II,* volume 11922 of *Lecture Notes in Computer Science.*
- CR22 [**27**] "On Committing Authenticated Encryption" appeared at ESORICS 2022

The dissertation also contains some discussion about a prototype implementation of some anonymous AE schemes. This discussion does not appear in the publication of CR19.

While anAE and cAE stem from works on nonce-based AE, we do not discuss those works in this chapter. Instead, we go over various nAE definitions from these works in greater detail in Chapter 2.

ANONYMOUS AE RELATED WORKS. In the CAESAR call for AE algorithms, Bernstein introduced the notion of a *secret message number* (SMN) as a possible alternative to a nonce, which he renamed the *public message number* (PMN) [**21**]. When the party encrypting a message specifies an SMN, the decrypting party doesn't need to know it. It was an innovative idea, but few CAESAR submissions supported it [**3**], and none became finalists. Namprempre, Rogaway, and Shrimpton formalized Bernstein's idea by adjusting the nAE syntax and security notion [**46**]. Their definition didn't capture any privacy properties or advantages of SMNs.

It was also Bernstein who asked (personal communication, 2017) if one could quickly identify which session an AE-encrypted ciphertext belonged to if one was unwilling to explicitly annotate it. NonceWrap does this, assuming a stateful receiver using what we would call a constant-breadth nonce policy.

Coming to the problem from a different angle, Bellare, Ng, and Tackmann contemporaneously investigated the danger of flowing nonces, and recast decryption so that a nonce needn't be provided [**15**]. Their concern lies in the fact that an encrypting party can't select *any* non-repeating nonce (it shouldn't depend on the plaintext or key), and emphasize that the nAE definition fails to specify which choices are fine.

Our approach to parameterizing anAE's goal using a nonce policy Nx benefits from the evolution of treatments on stateful AE [$\mathbf{13}, \mathbf{23}, \mathbf{38}, \mathbf{64}$]. The introduction of Lx (likely nonces) as something distinct from Nx (permissible nonces) is new.

A privacy goal for semantically secure encryption has been formalized as *key privacy* [**10**] in the public-key setting and as *which-key concealing encryption* [**1**] in the shared-key one. But the intent there was narrow: probabilistic encryption (not AE), when the correct key is known, out of band, by the decrypting party.

For nAE, one justification for moving from ind-style privacy [**11**,**33**] to ind\$-style privacy [**58**,**61**] was to achieve anonymity. But this benefit evaporates if a cleartext nonce or session-ID accompanies a ciphertext.

Banfi and Maurer also study anonymity for authenticated encryption [**8**]. Their starting point differs from ours as they start from probabilistic authenticated encryption without associated data.

Committing AE Related Works. 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 [**32**], 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 term *robustness* [**2**]. The work is in the asymmetric setting and requires an adversary to produce a ciphertext that validly decrypts under two different keys. Their notion encompasses keys that are honestly generated. Later, Farshim, Libert, Paterson, and Quaglia point out that, for some applications, robustness against adversarially-chosen keys is critical [**30**]. They strengthen Abdalla et al.'s notion to address this observation.

Farshim, Orlandi, and Roşie (FOR17) [**31**] 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) [**36**] defined a variant of committing AE with the goal of constructing schemes that support *message franking.* Dodis, Grubbs, Ristenpart, and Woodage (DGRW18) [**29**] 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 Schmieg (ADGKLS20) [**4**] 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 [**12**]. 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 [**40**]. Their attacks are built on efficiently creating ciphertexts that successfully decrypt under many keys.

As one of the goals of our cAE work is to merge committing AE definitions into a unifying framework, we give a more detailed comparison of most of these related works in Chapter 4.4, looking at both their definitions and some of their constructions.

CHAPTER 2

Nonce-Based Authenticated Encryption with Associated Data

In this chapter, we recall various definitions of *nonce-based authenticated encryption with associated data* (nAE). These definitions are the foundational starting points for anonymous AE and committing AE and will be useful in the analysis of the schemes presented in this dissertation.

2.1. Nonce-based AE Definitions

STANDARD nAE. To start, we present a standard nAE definition close to that from [**58**], which formalizes an nAE scheme as a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$—key generation, encryption, and decryption. Some formalizations may omit the key generation algorithm. The *key generation algorithm* $\mathcal{K}$ returns a string from some non-empty finite set of strings call the *key space*. The *Encryption algorithm* $\mathcal{E}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ takes as input a key, nonce, AD, and message and outputs a ciphertext. Some formulations may also allow encryption to output an error symbol $\bot$, insisting that if given invalid inputs (a key, nonce, AD, and message that are not elements of their appropriate sets) then encryption outputs $\bot$. Otherwise encryption always outputs non-$\bot$ values. The *Decryption algorithm* $\mathcal{D}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\bot\}$ takes as input a key, nonce, AD, and a ciphertext and either outputs a message or the symbol $\bot$ indicating failure. Occasionally, $\mathcal{E}(K, N, A, M)$ and $\mathcal{D}(K, N, A, C)$ are written as $\mathcal{E}_K^{N,A}(M)$ and $\mathcal{D}_K^{N,A}(C)$ respectively. We may also write $\Pi.\mathcal{E}(K, N, A, M)$ to specify that we are calling the scheme Π's encryption algorithm, but will sometimes omit Π if it is obvious from context.

An nAE scheme is said to be *correct* if decryption reverses encryption. That is, if $C \leftarrow \mathcal{E}(K, N, A, M)$ then $\mathcal{D}(K, N, A, C) = M$. Otherwise $\mathcal{D}$ returns $\bot$. For a correct scheme, one can observe that $\mathcal{E}(K, N, A, \cdot)$ is injective for any K, N, A. In our usage of nAE, we will require that the message space $\mathcal{M} \subseteq \{0,1\}^*$ be a set of strings for which $M \in \mathcal{M}$ implies that $\{0,1\}^{|M|} \subseteq \mathcal{M}$. Furthermore, we assume that $|\mathcal{E}_K^{N,A}(M)| = |M| + \tau$ where τ is a constant that we refer to as the nAE scheme's *expansion*.

In our anAE chapter (Chapter 3), we treat nAE encryption schemes as just the encryption function $\mathcal{E}$ itself instead of the standard triple. We can do so since the injectivity of $\mathcal{E}(K, N, A, \cdot)$ for correct schemes naturally gives rise to the decryption function $\mathcal{D} = \mathcal{E}^{-1}$.

Let Π be an nAE encryption scheme with an expansion of τ. One customary way to define nAE security [**47**, **63**] associates to an adversary $\mathcal{A}$ the real number, its *advantage*, $\mathbf{Adv}^{\mathrm{nae}}_{\mathcal{E}}(\mathcal{A}) = \Pr[K \twoheadleftarrow \mathcal{K} \colon \mathcal{A}^{E_K(\cdot,\cdot,\cdot),\, D_K(\cdot,\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot,\cdot),\, \perp(\cdot,\cdot,\cdot)} \Rightarrow 1]$ where the four oracles have the following behavior: the oracle $E_K(\cdot,\cdot,\cdot)$ on input N, A, M returns $\Pi.\mathcal{E}^{N,A}_K(M)$; the oracle $D_K(\cdot,\cdot,\cdot)$ on input N, A, C returns $\Pi.\mathcal{D}^{N,A}_K(C)$; the oracle $\$(\cdot,\cdot,\cdot)$ on input N, A, M returns a string of sampled uniformly at random from $\{0,1\}^{|M|+\tau}$; the oracle $\perp(\cdot,\cdot,\cdot)$ on input N, A, C always returns $\perp$. The oracles are sometimes referred to as the *real* encryption, *real* decryption, *ideal* or *fake* encryption, and *ideal* or *fake* decryption oracles respectively. The paired real oracles are referred to as the *real game* and the paired ideal oracles are the *ideal game*. The advantage quantifies how well the adversary $\mathcal{A}$ does in distinguishing the real oracles from the ideal oracles. The adversary $\mathcal{A}$ is forbidden from asking its first oracle (either E_K or \$) a query (N, A, M) if it previously asked a query (N, A', M'). Adversaries that adhere to this restriction are said to be *nonce-respecting.* It may also not ask its second oracle (N, A, C) if it previously asked its first oracle (N, A, M) and got back an answer C. These two restrictions are in place to prevent $\mathcal{A}$ from trivially distinguishing the two games.

This security definition is sometimes referred to as the *all-in-one* definition for nAE security as it captures both privacy and authenticity "all-in-one" package. Privacy comes from the adversary having to discern between real ciphertexts and uniformly random bits of the appropriate length. Authenticity comes from the adversary having to produce a ciphertext that decrypts without error. These are the two ways in which an adversary can distinguish between the real and ideal games.

Privacy and Authenticity. While the all-in-one definition is convenient in that it captures a strong notion of security in one simple definition, it is sometimes useful to think of privacy and authenticity as separate notions. We start with the games capturing these two notions from [**61**] albeit the notions from this paper are for nonce-based symmetric encryption—meaning they do not include AD. Quick adjustments can bring them to the nAE setting.

Let Π be an nAE scheme. Let $\mathcal{A}$ be a nonce-respecting adversary attacking the *privacy* of Π. Adversary $\mathcal{A}$ is asked to distinguish between a "real" and "ideal" encryption oracle. The real oracle E is initialized with a key K sampled uniformly at random and uses it to output $\Pi.\mathcal{E}(K, N, A, M)$ for any queries (N, A, M) that $\mathcal{A}$ makes. The ideal oracle \$ always outputs a uniformly random string of length $|M| + \tau$ where τ is the expansion of Π. The advantage of $\mathcal{A}$ in this privacy game is then quantified as:

$$\mathbf{Adv}_{\Pi}^{\mathrm{priv}}(\mathcal{A}) = \Pr[K \twoheadleftarrow \mathcal{K}; \mathcal{A}^{E_K(\cdot,\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1].$$

The authenticity game asks that the nonce-respecting adversary $\mathcal{A}$ *forges.* That is, $\mathcal{A}$ is asked to output a tuple (N, A, C) such that $\Pi.\mathcal{D}(K, N, A, C) \neq \perp$ for some key K sampled uniformly at random upon initialization of the game. Adversary $\mathcal{A}$ has access to an encryption oracle that performs encryption under K. Its authenticity advantage is defined as:

$$\mathbf{Adv}_{\Pi}^{\mathrm{auth}}(\mathcal{A}) = \Pr[K \twoheadleftarrow \mathcal{K}; \mathcal{A}^{E_K(\cdot,\cdot,\cdot)} \text{ forges }].$$

To prevent trivial wins, if $\mathcal{A}$ made an encryption query (N, A, M) that returned C, it may not output (N, A, C) as its forgery.

Multi-user Security. Classic nAE definitions like those described above are in the single user setting, meaning they deal with an adversary attacking a scheme using a single hidden key. Both of the main contributions of this dissertation, anonymous AE and committing AE, naturally give rise to a setting that concerns multiple keys. As such, the multi-user nAE security definition of [**18**] will be useful. There will be a couple differences in its presentation here: (1) The definition we use will be in the standard model as opposed to the ideal-cipher model. (2) Instead of having a verification oracle, the definition here will use standard decryption oracles. (3) In place of an oracle that allows the adversary to ask for a new secret key to be generated, the definition here will have all the keys pre-generated at initialization for the real game. In regards to (2) and (3), [**18**] asserts that the two can be shown to be equivalent using standard hybrid arguments and techniques from [**14**].

Let Π be an nAE scheme. The games in Fig. 2.1 capture the multi-user nAE security of Π. In the pseudocode, we write $X \twoheadleftarrow \mathcal{S}$ to mean that an element of $\mathcal{S}$ is picked uniformly at random and

```
Real$_\Pi^{\mathrm{naem}}$

procedure Initialize()
00 for $\ell \in \mathbb{N}$ do $K_\ell \twoheadleftarrow \mathcal{K}$; NE[$\ell$] $\leftarrow \emptyset$

procedure ENC($i, N, A, M$)
10 if $N \in$ NE[$\ell$] then ret $\bot$
11 NE[$\ell$] $\leftarrow N$; ret $\Pi.\mathcal{E}(K_i, N, A, M)$

procedure DEC($i, N, A, C$)
20 ret $\Pi.\mathcal{D}(K_i, N, A, C)$
```

```
Ideal$_\Pi^{\mathrm{naem}}$

procedure ENC($i, N, A, M$)
30 if $N \in$ NE[$\ell$] then ret $\bot$
31 NE[$\ell$] $\leftarrow N$
32 $C \twoheadleftarrow \{0,1\}^{|M|+\tau}$; ret $C$

procedure DEC($i, N, A, C$)
40 ret $\bot$
```

FIGURE 2.1. Games defining multi-user security for nAE schemes.

assigned to X. Like single user nAE security, the adversary interacts with either the real game or the ideal game and has to guess which. The advantage of an adversary $\mathcal{A}$ attacking Π is then quantified as

$$\mathbf{Adv}_{\Pi}^{\mathrm{naem}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathrm{Real}_\Pi^{\mathrm{naem}}} \to 1] - \Pr[\mathcal{A}^{\mathrm{Ideal}_\Pi^{\mathrm{naem}}} \to 1].$$

Lastly, a nonce-respecting adversary in the multi-user setting is one that does not repeat a nonce for a particular key for its ENC queries. This is enforced by the check on table NE in lines 10 and 30 in the game code. The letters N and E are intended to evoke *nonces* used for *encryption.*

PSEUDORANDOM INJECTION VARIANT. There is yet another nAE definition variant that we will find useful to recall. This variant asks that for an adversary to distinguish between $\mathcal{E}(K, N, A, \cdot)$ from a random injective function with the appropriate domain and range. This treatment of nAE comes from the PRI (pseudorandom injection) characterization of misuse-resistant nAE schemes in [**62**]. We adapt it to the multi-user setting following [**18**].

Recall that the expansion of an nAE scheme $\mathcal{E}$ is a constant τ such that $|\mathcal{E}_K^{N,A}(M)| = |M| + \tau$. Let $\mathcal{E}$ and τ be an nAE scheme and its expansion. Let $\mathcal{T}$ be an arbitrary nonempty set. Let $\mathrm{Inj}_\tau^{\mathcal{T}}(\mathcal{M})$ be the set of all functions $f\colon \mathcal{T} \times \mathcal{M} \to \{0,1\}^*$ such that $|f(T, M)| = |M| + \tau$ for all $M \in \{0,1\}^*$ and $f(T, \cdot)$ is an injection for all $T \in \mathcal{T}$. For $f \in \mathrm{Inj}_\tau^{\mathcal{T}}$ define $f^{-1}\colon \mathcal{T} \times \{0,1\}^* \to \mathcal{M} \cup \{\bot\}$ by

$f^{-1}(T, Y) = X$ when $f(T, X) = Y$ for some (unique) $X \in \mathcal{M}$, and $f^{-1}(T, Y) = \perp$ otherwise. Now given an adversary $\mathcal{A}$, define its advantage in attacking the nae$*$-security of $\mathcal{E}$ as the real number

$$\begin{aligned}\mathbf{Adv}^{\mathrm{nae}*}_{\mathcal{E}}(\mathcal{A}) \quad = \quad & \Pr[\text{for } i \in \mathbb{N} \text{ do } K_i \twoheadleftarrow \mathcal{K}\text{: } \mathcal{A}^{E_{\boldsymbol{K}}(\cdot,\cdot,\cdot,\cdot), D_{\boldsymbol{K}}(\cdot,\cdot,\cdot,\cdot)} \Rightarrow 1] - \\ & \Pr[f \twoheadleftarrow \mathrm{Inj}_{\tau}^{\mathbb{N}\times\mathcal{N}\times\mathcal{A}}(\mathcal{M})\text{: } \mathcal{A}^{f(\cdot,\cdot,\cdot,\cdot), f^{-1}(\cdot\cdot,\cdot,\cdot)} \Rightarrow 1]\end{aligned}$$

where the oracles behave as follows: oracle $E_{\boldsymbol{K}}$, on query (i, N, A, M), returns $\mathcal{E}(K_i, N, A, M)$; oracle $D_{\boldsymbol{K}}$, on query (i, N, A, C), returns $\mathcal{E}^{-1}(K_i, N, A, C)$; oracle f, on query (i, N, A, M), returns $f((i, N, A), M)$; and oracle f^{-1}, on query (i, N, A, C), returns $f^{-1}((i, N, A), C)$. The adversary $\mathcal{A}$ is forbidden from asking its first oracle any query (i, N, A, M) if it previously asked a query (i, N, A', M').

It is a standard exercise to show the equivalence of standard nae defined at the beginning of this chapter, and this formalization nae$*$ following the PRI treatment and multi-key treatment of [**62**] and [**18**].

2.2. Privacy and Usability-Violating Assumptions of Nonce-based AE

With those definitions established, we now discuss the privacy and usability-violating nuances of nAE in greater detail. Beginning with the syntax, decryption $\mathcal{D}$ is understood to be performed directly by the function. However, this function requires K, N, and A as inputs. This implies that the decrypting party knows what key to use. Suppose this decrypting party is communicating with multiple senders as a server might have connections with multiple clients. How, then, is the receiver supposed to know which of its keys to use? This dilemma suggests that the ciphertext will be delivered within some context that explicitly identifies the session associated with the ciphertext, which is synonymous to identifying the sender.

This is evident in practice in our most common standard secure Internet communications, HTTPS [**53**]. This protocol is HTTP secured by TLS [**19, 54**]. Hence, like its base protocol, HTTPS relies on TCP/IP as its transport and Internet protocols [**24, 56, 57**]. When a receiver such as a server gets a packet from a client over HTTP(S), it typically arrives on port 80—the official port number reserved for HTTP communications. Since all such packets arrive at the same destination, the receiver needs to know what key to use to decrypt them, which contextualizes the problem described above. To identify the sender and thus the associated key, the receiver relies on TCP/IP. Every TCP/IP connection or “session” can be uniquely identified by a four-tuple that consists of the source IP address, the destination IP address, the source port number, and the destination port number. This information is all sent in the clear and provides no anonymity.

It is important to note that this example described here can seem “out of scope” in some sense as a large part of the above pertains to packet routing and not so much nAE. After all, without the IP addresses and port numbers, how would one know where the packet should go and where to send a response to that packet? We admit that without the destination parts of the four-tuple, significant changes to the architecture of the Internet would need to be made to deliver a packet to its correct destination. However, we’d like to point out that a more privacy-minded routing protocol could put source information in a packet’s encrypted payload without affecting routing (source addresses can be spoofed after all). This would, at the very least, obfuscate the identity of the sender. For such a protocol, nAE would be a poor fit as it expects to be given an explicit key, which would naturally lead to tagging the ciphertext with sender identity in some way.

Moreover, nAE’s privacy issues do not end at key identification. For decryption, nAE also requires a nonce and AD. Associated data, historically, has been understood as data that needs to be authenticated but need not be confidential such as the header for a packet. But like a packet header, as previously described, flowing such information with the ciphertext can be damaging to privacy.

As for the nonce, popular nAE schemes such as AES-GCM [**45**] use a counter-based one. When flowed explicitly with the ciphertext, such a nonce then reveals a message’s *ordinality*—its position in a sequence of messages within a session. Counter-based nonces can also expose the identities of senders based on their volume of messages. Such nonces may, for example, be all that is needed to distinguish between a high-frequency stock trader (large counters) from a low-frequency stock trader (small counters). Furthermore, streams of messages from multiple sessions at different points of their counter can be sorted by their origin. Conventional nAE definitions effectively define the leakage of such information through the nonce as harmless, but perhaps it is nothing but tradition that has led us to accept this to be the case.

Regarding the privacy definition of nAE, *indistinguishability from random bits* is conventionally understood to provide anonymity. This is postulated by [**1**] and later proved by [**8**]. After all, it fits natural intuition: if the encryption of M under keys K and K' are both indistinguishable from random bits, then they are indistinguishable from each other. But the scoping of nAE overlooks the basic problem that the thing—the ciphertext—that is indistinguishable from random bits isn't everything the adversary will see—what we refer to as the *full ciphertext*.

While we focused primarily on the privacy-violating implications in our discussion here, we note that there are usability issues embedded in the nAE definition as well. After all, there is no direction as to how a user ought to supply a nonce and AD to the decryption algorithm–and if the user is in a context where identities are hidden, the key as well.

We turn to the design choices of several cryptographic libraries for some examples. In particular, we look at Google's Tink and Bernstein, Lange, and Schwabe's (BLS) NaCl [**20**, **22**, **35**]. When BLS published [**22**] detailing their design choices in 2012, they emphasized that they wanted to "avoid various types of cryptographic disasters suffered by previous libraries such as OpenSSL [**49**]," effectively putting usability as a priority. To encrypt and decrypt with NaCl, a user must supply a key, nonce, and message (the base library does not support AD). Regarding the nonce in the API, BLS states that they do not believe that the nonce is the user's responsibility. They stress that they "believe that cryptographers should take responsibility not just for nonces but also for other security aspects of high-level network protocols" [**22**]. It is not difficult to see why; developers with little to no cryptographic familiarity might not even know what a nonce *is*, let alone that the security of nAE is contingent on them not repeating one. The nonce surfacing in the NaCl interface is an artifact of how nonces are used by higher level protocols in different ways, which ultimately led them to leave it up to the user. Nonetheless, to take away the burden of nonce handling from the user was a consideration to improve usability at the time of NaCl's design.

Google's Tink is a more recent library [**35**] and they execute on that consideration. The Tink API supports a number of a languages including Python, Java, and C++. Regardless of which language, for key handling, a cipher object is instantiated with a key and this object is used subsequently for any encryption and decryption calls with respect to that key. (This is not uncommon among cryptographic libraries [**51**], but not every popular library does it [**42**].) Tink offers an nAE interface where the user only provides an AD and plaintext to the keyed cipher object per encryption call. Internally, a fresh nonce is generated every time encryption is called. This prevents any potential nonce misuse by users.

One can argue that since AD is traditionally understood to be data sent in the clear that requires authenticity and not privacy, having to supply the AD is not a usability issue for the decrypting party. This argument is fine for standard nAE, but if one wants a stronger notion of privacy—one including anonymity, then one needs to hide the AD as it can have identity revealing information. It is not clear how the decrypting party obtains the AD if it is hidden since this goes against conventional usage of AD. This is one of the issues we address in anonymous AE.

But usability issues don't end at just necessary provision of nonce and AD. Conventional nAE is in the single key setting; a sender and receiver share a secret key and use it to encrypt and decrypt messages. But in practice, systems using nAE do not always have just one active key in use. Some systems store a number of secret keys and need to identify which key to use before decrypting a ciphertext. It is not always obvious which key to use for a given ciphertext. Real world applications for which this is applicable include key management services (Amazon Web Services KMS, Oracle Key Vault, Microsoft Azure Key Vault to name a few [**6**,**7**,**50**]) and the Shadowsocks protocol. For cryptographic libraries, Google's Tink offers key labeling as a means to identify the appropriate key among a set of candidate keys [**35**].

The need for key identification can extend beyond a usability issue and become a security issue. For example, Grubbs, Len, and Ristenpart exploit ciphertexts that can decrypt under multiple keys to mount password recovery attacks on the Shadowsocks protocol in [**41**]. Dodis, Grubbs, Ristenpart, and Woodage show how to create AES-GCM ciphertexts to decrypt into an innocuous image under one key and a malicious one under a different key [**29**]. These issues would not arise if given a ciphertext, the decrypting party knows which of its keys to use. Appropriate means of key identification may allow just this. Alternatively, *Key robustness* is a property of an nAE scheme that ensures that decryption for a ciphertext fails unless it is decrypted with the "correct" key. This property is closely tied to committing AE, which we define and compare with key robustness in Chapter 4.

But in remedying all of the issues discussed here, we first present *anonymous AE*—a variant of nAE that will include anonymity in its notion of privacy. To move in this direction, we expand the scope of nAE's understanding of its ciphertext to what we called the full ciphertext, then the nonce, AD, and session identifier sent alongside the encrypted message is part of the ciphertext as well. This "other" data flowed with the encrypted message is what we will refer to as *metadata*. Anonymous AE has the goal of protecting exactly this metadata from adversarial parties.

CHAPTER 3

Anonymous Authenticated Encryption

This chapter presents *anonymous nonce-based authenticated encryption* (anAE), one of two major primitives and contributions of this dissertation. More precisely, this chapter formalizes anAE, provides an efficient anAE construction called NonceWrap, and proves NonceWrap secure.

Traditional formulations of authenticated encryption (AE) implicitly assume that auxiliary information is flowed alongside the ciphertext. This information, necessary to decrypt but not normally regarded as part of the ciphertext, may include a nonce, a session-ID (SID), and associated data (AD). But flowing these values in the clear may reveal the sender's identity.

To realize a more private form of encryption, we introduce a primitive we call *anonymous nonce-based AE*, or anAE. Unlike traditional AE [**16**, **37**, **58**, **61**, **63**], anAE treats privacy as a first-class goal. When we speak of privacy here and for this entire chapter, we mean in the sense of anonymity along with the conventional understanding of privacy. We insist that ciphertexts contain everything the receiver needs to decrypt other than its secret state (including its keys), and ask for indistinguishability from random bits even then. We show how to achieve anAE, providing a transform, NonceWrap, that turns a conventional nonce-based AE (nAE) scheme into an anAE scheme. We claim that anAE can improve not only on privacy, but on usability, too.

Background. The customary formulation for AE, nAE, requires the user to provide a nonce not only to encrypt a plaintext, but also to decrypt a ciphertext [**47**, **58**, **63**]. Decryption fails if the wrong nonce is provided.

How is the decrypting party supposed to know the right nonce to use? Sometimes it will know it *a priori*, as when communicants speak over a reliable channel and maintain matching counters. But at least as often the nonce is flowed, in the clear, alongside the ciphertext. The *full* ciphertext should be understood as including that nonce, as the decrypting party needs it to decrypt.

Yet transmitting a nonce along with the ciphertext raises both usability and security concerns. Usability is harmed because the ciphertext is no longer self-contained: information beyond it and the operative key are needed to decrypt. At the same time, confidentiality and privacy are harmed because the transmitted nonce *is* information, and information likely correlated to identity. Sending a counter-based nonce, which is the norm, will reveal a message's *ordinality*—its position is the sequence of messages that comprise a session. While the usual definition for nAE effectively *defines* this leakage as harmless, is it always so? A counter-based nonce may be all that is needed to distinguish, say, a high-frequency stock trader (large counters) from a low-frequency stock trader (small counters). With a counter-based nonce, multiple sessions at different points in the sequence can be sorted by point of origin. Perhaps it is nothing but tradition that has led us to accept that nAE schemes, conventionally used, may leak a message's ordinality and the sender's identify.

This chapter is about defining and constructing nonce-based AE schemes that are more protective of such metadata. We imagine multiple senders simultaneously communicating with a receiver, as though by broadcast, each session protected by its own key. When a ciphertext arrives, the receiver must decide which session it belongs to. But ciphertexts shouldn't get packaged with a nonce, or even an SID (session identifier) or AD (associated data), any of which would destroy anonymity. Instead, decryption should return these values, along with the underlying plaintext.

A LOUSY APPROACH. One way to conceal the operative nonce and SID would be to encrypt those things under a public key belonging to the receiver. The resulting ciphertext would flow along with an ind$-secure nAE-encrypted ciphertext (where ind$ refers to indistinguishability from uniform random bits [**61**]). While this approach can work, moving to the public-key setting would decimate the trust model, lengthen each ciphertext, and substantially slow each encryption and decryption, augmenting every symmetric-key operation with a public-key one. We prefer an approach that preserves the symmetric trust model and has minimal impact on message lengths and computation time.

CONTRIBUTIONS: DEFINITIONS. We provide a formalization of anonymous AE that we call anAE, *anonymous nonce-based AE.* Our treatment makes anAE encryption identical to encryption under nAE. Either way, encryption is accomplished with a deterministic algorithm $C = \mathcal{E}_K^{N,A}(M)$ operating on the key K, nonce N, associated data A, and plaintext M. As usual, ciphertexts so produced can be decrypted by an algorithm $M = \mathcal{D}_K^{N,A}(C)$. But the receiver employing a privacy-conscious protocol might not *know* what K, N, or A to use, as flowing N or A, or identifying K in any direct way, would damage privacy. So an anAE scheme supplements the decryption algorithm $\mathcal{D}$ with a constellation of alternative algorithms. They let the receiver: initialize a session (`Init`); terminate a session (`Term`); associate an AD with a session, or with all sessions (`Asso`); disassociate an AD with a session, or with all sessions (`Disa`); and decrypt a ciphertext, given nothing else (`Dec`). The last returns not only the plaintext but, also, the nonce, SID, and AD.

After formalizing the syntax for anAE we define security, doing this in the concrete-security, game-based tradition. A single game formalizes confidentiality, privacy, and authenticity, unified as a single notion. It is parameterized by a *nonce policy*, $N\mathrm{x}$, which defines what nonces a receiver should consider permissible at some point in time. We distinguish this from the nonce or nonces that are *anticipated*, or *likely*, at some point in time, formalized by a different function, $L\mathrm{x}$. Our treatment of permissible nonces vs. likely nonces may be useful beyond anonymity, and can be used to speed up decryption.

Anonymous AE can be formalized without a user-supplied nonce as an input to encryption, going back to a probabilistic or stateful definition of AE. For this reason, anAE should be understood as one way to treat anonymous AE, not the only way possible. That said, our choice to build on nAE was carefully considered. Maintaining nAE-style encryption, right down to the API, should facilitate backward compatibility and a cleaner migration path from something now quite standard. Beyond this, the reasons for a nonce-based treatment of AE remain valid after privacy (and anonymity) become a concern. These include minimizing requirements on user-supplied randomness/IVs.

CONTRIBUTIONS: CONSTRUCTIONS. We next investigate how to achieve anAE. Ignoring the AD, an obvious construction is to encipher the nonce using a blockcipher, creating a header $\mathrm{Head} = E_{K_1}(N)$. This is sent along with an nAE-encrypted $\mathrm{Body} = \mathcal{E}_{K_2}^{N,A}(M)$. But the ciphertext $C = \mathrm{Head} \parallel \mathrm{Body}$ so produced would be slow to decrypt, as one would need to trial-decrypt Body under each receiver-known key K_2' until the (apparently) right one is found (according to the nAE scheme's authenticity-check). If the receiver has s active sessions and the message has $|M| = m$ bits, one can expect a decryption time of $\Theta(ms)$.

To do better we put redundancy in the header, replacing it with $\mathrm{Head} = E_{K_1}(N \,\|\, 0^{\rho} \,\|\, H(AD))$. Look ahead to Fig. 3.2 for our scheme, NonceWrap. As a concrete example, if the nonce N is 12 bytes [**43**] and we use the degenerate hash $H(x) = \varepsilon$ (the empty string), then one could encrypt a plaintext M as $C = \mathrm{AES}_{K_1}(N \,\|\, 0^{32}) \,\|\, \mathrm{GCM}_{K_2}^{N,A}(M)$. Using the header Head to screen candidate keys (only those that produce the right redundancy) and assuming $\rho \geq \lg s$ we can now expect a decryption time of $\Theta(m + s)$ for s blockcipher calls and a single nAE decryption.

In many situations, we can do better, as the receiver will be able to anticipate each nonce for each session. If the receiver is stateful and maintains a dictionary ADT (abstract data type) of all anticipated headers expected to arrive, then a single `lookup` operation replaces the trial decryptions of Head under each prospective key. Using standard data-structure techniques based on hashing or balanced binary trees, the expected run time drops to $\Theta(m + \lg(s))$ for decrypting a length-m string. And one can always fall back to the $\Theta(m + s)$-time process if an unanticipated nonce was used.

Finally, in some situations one can do better still, when all permissible nonces can be anticipated. In such a case the decrypting party need never invert the blockcipher E and the header can be truncated, or some other PRF can be used. In practice, the header could be reduced from 16 bytes to one or two bytes—a savings over a conventional nAE scheme that transmits the nonce.

While NonceWrap encryption is simple, decryption is not; look ahead to Figs. 3.3 and 3.4. Even on the encryption side, there are multiple approaches for handling the AD. Among them we have chosen the one that is most bandwidth-efficient and that seems to make the least fuss over the AD.

3.1. Anonymous Nonce-based AE (anAE) Definition

A foundational principle of secure encryption is that parties that do not possess the key should not learn any information from just the ciphertext (although typically leakage of the message length is allowed). If we exclude metadata, nAE notions follow this principle just fine. But once we expand our understanding of the typical nAE ciphertext to a full ciphertext—everything needed to decrypt other than the key itself—then this falls apart for nAE. On the other hand, our anAE notion follows this principle and adheres to it strictly by including identity in that which should not be leaked, hence the name *anonymous AE*. Any metadata attached to a ciphertext can be identity-compromising, so anAE seeks to protect it. We rewrite this principle in a stricter fashion explicitly here:

> **Privacy Principle.** A ciphertext should not by itself compromise the identity of its sender. This should hold even when the term "ciphertext" is understood as the *full ciphertext*—everything the receiver needs to decrypt and that the adversary might see.

This assertion of the principle forbids our exclusion of metadata from our understanding of the word *ciphertext*. It emphasizes that although metadata is necessary to decrypt, we cannot ignore the fact that data is a privacy-violating part of transmitting a ciphertext.

Stated as above, the privacy principle may seem obvious. But the fact that nAE blatantly violates this principle, despite being understood as an extremely strong notion of security, suggests otherwise. As such, in moving towards a formalization of anAE, attending to the full ciphertext seems appropriate. Our anAE notion captures the privacy that this principle demands while attending to conventional notions of confidentiality and authenticity. We now turn to demonstrating that.

Syntax. An anAE scheme is an extension of an nAE scheme with five additional algorithms. Formally, an anAE scheme is a six-tuple of deterministic algorithms $\Pi = (\texttt{Init}, \texttt{Term}, \texttt{Asso}, \texttt{Disa}, \texttt{Enc}, \texttt{Dec})$ These algorithms create a session, tear down a session, register an associated data, deregister an AD, encrypt a message, and decrypt a ciphertext respectively. Their interfaces are explicitly as follows:

- `Init`, the receiver's *session initialization algorithm*. It takes as input a key $K \in \mathcal{K}$ and outputs a session identifier (SID) $\ell \in \mathcal{L}$ that will be subsequently used as the name for the key's session. We will assume that all session-IDs outputted are unique.
- `Term`, the receiver's *session termination algorithm*. It takes as input an SID $\ell \in \mathcal{L}$ and returns nothing.

- Asso, the receiver's *AD-association algorithm.* It takes as input either an AD $A \in \mathcal{A}$ or a pair $(A, \ell) \in \mathcal{A} \times \mathcal{L}$. It outputs nothing.
- Disa, the receiver's *AD-disassociation algorithm.* On an input of either an AD $A \in \mathcal{A}$ or a pair $(A, \ell) \in \mathcal{A} \times \mathcal{L}$, it returns nothing.
- Enc, the sender's *encryption algorithm.* By itself, this algorithm is an nAE scheme, meaning $\texttt{Enc} = \mathcal{E}$. Recall that nAE encryption takes as input a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, an AD $A \in \mathcal{A}$, and a plaintext $M \in \mathcal{M}$, and outputs a ciphertext $C \in \mathcal{C}$.
- Dec, the receiver's *decryption algorithm.* This algorithm takes as input *only* a ciphertext $C \in \mathcal{C}$. It outputs either a four-tuple $(\ell, N, A, M) \in \mathcal{L} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ or the symbol $\perp$, indicating rejection. Observe that $\texttt{Dec} \neq \mathcal{D} = \texttt{Enc}^{-1}$ as is the case with conventional nAE encryption.

The sets, all non-empty, referred to above are as follows:

- $\mathcal{A}$, the *AD space*, is an arbitrary set.
- $\mathcal{C}$, the *ciphertext space*, is a set of strings.
- $\mathcal{K}$, the *key space*, is a finite set of strings.
- $\mathcal{L}$, the *SID space*, is an arbitrary set. However, for the remainder of this text, we will assume $\mathcal{L} = \mathbb{N}$, treating all SIDs as natural numbers for simplicity.
- $\mathcal{M}$, the *message Space*, is a set of strings. Since anAE encryption is in itself an nAE scheme, the same assumption on the message space applies. That is, if $M \in \mathcal{M}$ then $\{0,1\}^{|M|} \subseteq \mathcal{M}$.
- $\mathcal{N}$, the *nonce space*, is a finite set of strings.

We stress that, while it follows from nAE injectivity that Enc has some inverse $\mathcal{D} = \texttt{Enc}^{-1}$, this is distinct from Dec. The Dec algorithm is asked to recover not just the message, but all of its relevant metadata when given just a ciphertext. In accomplishing this task, the decrypting party is equipped with a persistent state $\boldsymbol{K} \in \boldsymbol{\mathcal{K}}$ where $\boldsymbol{K}$ is some arbitrary set. The four other algorithms Init, Term, Asso, and Disa are algorithms for the decryptor to manipulate its state. One can think of these four algorithms and Dec as an alternative to $\mathcal{D}$. After all, for any ciphertext C made by Enc, one can explicitly supply a key K, nonce N, and AD A to perform $\mathcal{D}_K^{N,A}(C)$ since Enc is an nAE scheme. The four algorithms and Dec simply offer an option to preserve anonymity.

Nonce Policy. In an AE scheme with stateful decryption [**13**, **23**, **38**, **64**], at any given point in time, the receiver will have some set of nonces that it deems acceptable. Typically, this set depends on the nonces already received and nothing else. We formalize this as a function we call a *nonce policy* $N\mathrm{x}\colon \mathcal{N}^{\leq d} \to \mathcal{P}(\mathcal{N})$ where $d \geq 0$. The notation $\mathcal{P}(\mathcal{S})$ denotes the set of all subsets of the set $\mathcal{S}$.

The set $N\mathrm{x}(\boldsymbol{N})$ is the set of *permissible nonces* given the history of nonces $\boldsymbol{N}$. The history is an ordered list of previously received nonces. The value $d = \mathrm{depth}(Nx) \in \mathbb{N} \cup \{\infty\}$ is the *depth* of the policy. This value captures the number of nonces that needs to be recorded in the history for the policy to give an appropriate set of valid nonces. The value $b = \mathrm{breadth}(Nx) = \max_{\boldsymbol{N} \in \mathrm{dom}(Nx)} |N\mathrm{x}(\boldsymbol{N})|$ is the *breadth* of the policy. This value is the maximum number of nonces that are deemed possible across all possible histories for the policy. For a function $F\colon \mathcal{A} \to \mathcal{B}$, we write $\mathrm{dom}(F) = \mathcal{A}$ to indicate the domain of F and $\mathrm{range}(F) = \mathcal{B}$ to indicate its range. Lastly, the naming of the function *Nx* is intended to suggest the words *next* and *nonce*.

We highlight that the input of the nonce policy is bounded by its depth as opposed to allowing an input of $\mathcal{N}^*$. This is because any practical nonce policy would have a bounded depth, as it would otherwise require the receiver to maintain an unlimited state and slow down decryption as connections grow old.

It is worth mentioning two different nonce policy extremes. There is what we refer to as the *permissive* policy $N\mathrm{x}(\Lambda) = \mathcal{N}$, which allows all nonces and needs not keep track of history. This policy captures a stateless nAE scheme where repetitions, omissions, and out-of-order deliveries are all permitted. We use the symbol Λ to denote the empty list. The permissive policy has a depth $d = 0$ and a breadth $b = |\mathcal{N}|$. Note that while a decryption algorithm employing this policy treats all nonces as permissible, a higher-level process using the algorithm could still restrict nonces. On the other end of the spectrum is the *strict* policy. For a nonce space of $\mathcal{N} = \{0..N_{\max}\}$, the strict policy is defined by $N\mathrm{x}(\Lambda) = 0$, $N\mathrm{x}((N)) = \{N+1\}$ (for $N < N_{\max}$), and $N\mathrm{x}(N_{\max}) = \emptyset$. It demands the absence of repetitions, omissions, and out-of-order deliveries. With a breadth and dept of $b = d = 1$, the nonces for this policy must start at zero and strictly increment until all nonces are exhausted. On a reliable channel, the strict policy is a natural choice. There is a rich set of policies between these two extremes [**13**, **23**, **38**, **64**].

Associated Data. In the context of AE, associated data has historically been understood as additional information that needs to be authenticated, but need not be private [**58**]. This can, for example, be the header for a packet. It is not difficult to see that usage of AD can be identity-compromising. So, anAE targets the privacy of AD as well as its authenticity.

Now, one might interject and claim that it is unreasonable to require both privacy and authenticity for AD. Such a requirement would not only contradict how AD is traditionally defined, but also blurs the distinction between it and the message. After all, the whole point of encrypting the message with AE is to give it privacy and authenticity. Since AD now requires the same security, this defeats the purpose of having the AD. Why not just treat it the same as the message and do away with the concept of AD altogether?

We argue that this reasoning is specious—despite all there is distinction between the AD and the message. First put aside the usage of AE where one encrypts an empty message solely to authenticate the AD. Then, the sender of a message turns to encryption in the first place because they possess that they want to communicate to a receiver. The sender does not typically expect the receiver to know the contents of the message *a priori* to decryption. This is not the case with AD. Take the usage of AD in TLS for example. As of TLS 1.3, the AD is the header of the encrypted record, which usually consists of two protocol-related constants and the length of the ciphertext. The constants are known to the receiver as they are defined by the protocol specification and the length of the ciphertext can be gleaned simply by receiving the encrypted record. All this information is known to the receiver a priori to decryption.

Our treatment of AD in the anAE context will be limited to exactly this kind of AD—data that is known a priori to the receiver, but AD will be supported nonetheless. Recall that anAE is only an extension of nAE and anAE only offers a more privacy-ambitious decryption. Should one need to use AD that the receiver cannot know a priori to decryption, then one can simply fall back to using the anAE scheme as an nAE one. Obviously, one loses the privacy benefits of anAE from doing so. One can also reconsider the distinction between message and AD. That is, the original argument against a private and authentic AD is that it lumps AD too closely with the message. So, one can consider instead lumping all information that cannot be learned before decryption with the message and anything that can be known by the receiver a priori with the AD. Pivoting one's understanding of AD in this way may also allow one to enjoy the privacy benefits of using anAE.

With the inclusion of AD in the anAE definition justified, we now turn to the peculiar interface that supports it. The two algorithms responsible for managing AD are the AD registration algorithm `Asso` and the AD disassociation algorithm `Disa`. These algorithms take in either an SID and AD pair, or a standalone AD. The former follows the intuition that for a specific session, one might have specific AD values that are expected from the sender. Therefore, per session, the receiver can maintain a set of AD values that it expects to receive. The latter addresses the use case where some AD values would not be specific to a session. In reference to the above example of TLS 1.3's usage of AD, one can observe that AD values like the two protocol-related constants can apply to multiple or all sessions that a receiver has open. To support such a use case, we allow the registration of AD values that apply across all active sessions. We call such AD values *global ADs* as opposed to the *session-specific ADs*. For a received ciphertext, only AD values registered globally or to the session specific to the ciphertext, which decryption is responsible for identifying, can match it. Registration and de-registration of ADs with this interface is possible because the decrypting party can determine *a priori* what ADs may be applicable to the ciphertexts it expects to receive.

We anticipate that for most settings, users will only need to associate a single AD to a session at any given time. This single AD may be session-specific or global, but once that AD is identified, it is understood as *the* AD for it. If this is the case for the decrypting party, we say that the receiver abides by the *one-AD-per-session restriction*, which we write as 1AD/session. Later in our anAE construction, such receivers will enjoy efficiency benefits.

STATELESS SCHEMES. Our formalization of anAE schemes treats the decrypting party as stateful. Even if there is a only single key K and a single AD A to be associated with it, the decrypting party would still initialize the session for K with an `Init` call, register A with an `Asso` call, and then call `Dec` on any incoming ciphertexts. This seemingly roundabout usage of state is an artifact of the generality of our formulation. For distinction between statefulness and statelessness in the anAE context, we say that a scheme is *stateless* if the decryption algorith `Dec` does not modify the state.

For stateless anAE, one might provide an alternative API in which keys and ADs are provided on each call—an interface like $\texttt{Dec}'(K, A, C)$ for example. One may also initialize a read-only data structure managing operative key and AD values that is given to decryption each call. Many popular crypto libraries today do something similar to this. That is, they do not pass in string-valued keys to encryption or decryption algorithms, but rather an opaque data structure is created during a key-preprocessing step and that structure is then used for subsequent encryption and decryption calls [**34**, **49**, **51**].

Defining Security. Let $\Pi = (\texttt{Init}, \texttt{Term}, \texttt{Asso}, \texttt{Disa}, \texttt{Enc}, \texttt{Dec})$ be an anAE scheme and let $N\mathrm{x}$ be a nonce policy. The anAE security of Π with respect to $N\mathrm{x}$ is captured by the pair of games in Fig. 3.1. The adversary interacts with either the real game $\mathrm{Real}^{\mathrm{anae}}_{\Pi,N\mathrm{x}}$ or the ideal game $\mathrm{Ideal}^{\mathrm{anae}}_{\Pi,N\mathrm{x}}$ and tries to guess which of the two its interacting with. The advantage of $\mathcal{A}$ attacking Π with respect to $N\mathrm{x}$ is defined as

$$\mathbf{Adv}^{\mathrm{anae}}_{\Pi,N\mathrm{x}}(\mathcal{A}) = \mathbf{Pr}[\ \mathcal{A}^{\mathrm{Real}^{\mathrm{anae}}_{\Pi,N\mathrm{x}}} \rightarrow 1\ ] - \mathbf{Pr}[\ \mathcal{A}^{\mathrm{Ideal}^{\mathrm{anae}}_{\Pi,N\mathrm{x}}} \rightarrow 1\ ]\,,$$

the difference in probability that the adversary outputs "1" in each of the games.

In our pseudocode, integers, strings, lists, and associative arrays are silently intialized to $0, \varepsilon, \Lambda$, and $\emptyset$ respectively. For a nonempty list $\boldsymbol{x} = (x_1, \ldots, x_n)$ we let $\mathrm{tail}(\boldsymbol{x}) = (x_2, \ldots, x_n)$. We write $A \overset{\cup}{\leftarrow} B$, $A \overset{\setminus}{\leftarrow} B$, and $A \overset{\|}{\leftarrow} B$ to denote $A \leftarrow A \cup B$, $A \leftarrow A \setminus B$, and $A \leftarrow A \parallel B$. When iterating through a string-valued set, we do so in lexicographic order.

Our security games (and later our anAE scheme itself) makes extensive use of *associative arrays* (also called *maps* or *dictionaries*). These are collections of (key, value) pairs with at most one value per key. For an array A and a key K, we write $A[K]$ for the value in A associated with the key K. To add a new value, or modify an existing one, we write $A[K] getsV$ to (re)assign the value V to the key K in A. We write A.keys to denote the set of all keys in A Similarly, we use A.values to denote the set of all values in A These last two operations are not always mentioned in abstract treatments of dictionaries, but programming languages like Python do support these methods and realizations of dictionaries invariably enable them.

```
$\underline{\mathrm{Real}^{\mathrm{anae}}_{\Pi,Nx}}$

procedure INIT()
  $K \twoheadleftarrow \mathcal{K}$
  $\ell \leftarrow \Pi.\mathtt{Init}(K)$    Guaranteed new
  $\mathrm{K}[\ell] \leftarrow K$; $\mathrm{L} \xleftarrow{\cup} \{\ell\}$; $\mathrm{NE}[\ell] \leftarrow \emptyset$
  ret $\ell$

procedure TERM($\ell$)
  $\Pi.\mathtt{Term}(\ell)$; $\mathrm{L} \xleftarrow{-} \{\ell\}$

procedure ASSO($A$)
  $\Pi.\mathtt{Asso}(A)$
procedure ASSO($A, \ell$)
  $\Pi.\mathtt{Asso}(A, \ell)$

procedure DISA($A$)
  $\Pi.\mathtt{Disa}(A)$
procedure DISA($A, \ell$)
  $\Pi.\mathtt{Disa}(A, \ell)$

procedure ENC($\ell, N, A, M$)
  if $\ell \notin \mathrm{L}$ or $N \in \mathrm{NE}[\ell]$ then
    ret $\perp$
  $\mathrm{NE}[\ell] \xleftarrow{\cup} \{N\}$
  ret $\Pi.\mathtt{Enc}(\mathrm{K}[\ell], N, A, M)$

procedure DEC($C$)
  ret $\Pi.\mathtt{Dec}(C)$
```

```
$\underline{\mathrm{Ideal}^{\mathrm{anae}}_{\Pi,Nx}}$

procedure INIT()
  $K \twoheadleftarrow \mathcal{K}$
  $\ell \leftarrow \Pi.\mathtt{Init}(K)$
  $\mathrm{A}[\ell] \leftarrow \emptyset$; $\mathrm{L} \xleftarrow{\cup} \{\ell\}$
  $\mathrm{NE}[\ell] \leftarrow \emptyset$; $\mathrm{ND}[\ell] \leftarrow \Lambda$
  ret $\ell$

procedure TERM($\ell$)
  $\mathrm{L} \xleftarrow{-} \{\ell\}$

procedure ASSO($A$)
  $\mathrm{AD} \xleftarrow{\cup} \{A\}$
procedure ASSO($A, \ell$)
  $\mathrm{A}[\ell] \xleftarrow{\cup} \{A\}$

procedure DISA($A$)
  $\mathrm{AD} \xleftarrow{-} \{A\}$
procedure DISA($A, \ell$)
  $\mathrm{A}[\ell] \xleftarrow{-} \{A\}$

procedure ENC($\ell, N, A, M$)
  if $\ell \notin \mathrm{L}$ or $N \in \mathrm{NE}[\ell]$ then
    ret $\perp$
  $C \twoheadleftarrow \{0,1\}^{|M|+\tau}$
  $\mathrm{NE}[\ell] \xleftarrow{\cup} \{N\}$
  $\mathrm{H}[C] \xleftarrow{\cup} \{(\ell, N, A, M)\}$
  ret $C$

procedure DEC($C$)
  if $H[C] = \emptyset$ then ret $\perp$
  if $\exists$ unique $(\ell, N, A, M) \in \mathrm{H}[C]$ s.t.
    $\ell \in \mathrm{L}$ and $N \in Nx(\mathrm{ND}[\ell])$ and
      $A \in \mathrm{AD} \cup \mathrm{A}[\ell]$ then
      $\mathrm{ND}[\ell] \xleftarrow{\|} N$
      if $|\mathrm{ND}[\ell]| > d$ then
        $\mathrm{ND}[\ell] \leftarrow \mathrm{tail}(\mathrm{ND}[\ell])$
      ret $(\ell, N, A, M)$
  ret $\perp$
```

FIGURE 3.1. **Defining anAE security.** The games depend on an anAE scheme Π and a nonce policy Nx. The adversary must distinguish the game on the left from the one on the right. Privacy, confidentiality, and authenticity are simultaneously captured.

EXPLANATION. The "real" anAE game surfaces to the adversary six procedures, each corresponding to the algorithms of an anAE scheme. Modeling correct use, the `Init` algorithm generates random keys, while calls `Enc` may not repeat nonces within a given session. Further restrictions on requesting encryption for uninitialized SIDs or the SIDs of a terminated session apply (which is a restriction in-line with previous multi-user AE definitions [**18**]). Some bookkeeping is necessary for the game to enforce these restrictions, namely, K_ℓ being the key associated to session ℓ and L being the set of all active session labels. The set NE[ℓ] is the set of nonces that have already been used for session ℓ (nested within the associative array NE). Enforcing these restrictions aside, the real game allows the adversary to interact with each of the algorithms of Π through its procedures in a straightforward manner.

Similarly, the "ideal" game provides the same six entry points as the "real" game, but only employs the scheme Π insofar as ensuring the procedure INIT returns the same sequence of session labels as the scheme's session initialization algorithm `Init`. By ensuring that the labels of the INIT oracle match that of the `Init` algorithm, we prevent an adversary from trivially distinguishing the two games simply by observing a mismatch in session labels. One other aspect of Π used in the ideal game is the expansion constant τ of its encryption algorithm. Recall that the *expansion* of an nAE scheme is the length increase that a message undergoes when it is encrypted into a ciphertext. Using the constant, ideal encryption oracle ENC can return uniformly random bits of the appropriate length regardless of SID, nonce, AD, or plaintext (line 242 in the code). This captures both confidentiality and anonymity in a strong sense. The same idea is used by the ind$-form of the nonce-based symmetric encryption definition except that does not attend to the full ciphertext [**59**].

Following the all-in-one definition for nAE security [**62**], authenticity is ensured by having the ideal counterpart of the real decryption oracle routinely return $\perp$. When should ideal decryption *not* return $\perp$? Just like nAE, we want ideal decryption to return $\perp$ if the ciphertext C was not previously returned from an ENC query (line 250). (Common nAE definitions like the one presented in Chapter 2.2 forbid the adversary from making a decryption query with encryption output and always returning $\perp$. But one could write an identical ideal decryption oracle where such a query is allowed and is the only time a non-$\perp$ response is given.) We also want ideal DEC to return $\perp$

if the relevant session had been torn down, if the relevant nonce is out-of-policy, or if the relevant AD is unregistered as these would be impermissible decryptions. By *relevant* we mean associated with the ciphertext that is being queried for decryption. These are captured by lines 252 -253 .

There is one more case where ideal decryption returns $\perp$. This occurs when a query can result in more than one valid explanation for the ciphertext (captured by the word "unique" on line 251). Ambiguity arises when multiple valid decryptions are possible. Instead of dictating what secure behavior is when this occurs, we choose to give the adversary a win when it does. This effectively demands that it must be difficult for an adversary to create colliding ciphertexts using secure anAE schemes *when keys are sampled uniformly at random.* It is important to note that the definition does not address the possibility of adversaries that can learn the keys of communicants or even install their own keys. As in-depth discussion surrounding colliding ciphertexts naturally moves towards committing AE, we save further discussion on this for a later chapter.

It is worth touching on the bookkeeping used in the ideal game as well. There is a dictionary H (for "history") that is used in recording which (ℓ, N, A, M) values gave rise to a ciphertext C during encryption (line 244). This is later used in decryption to correctly respond to the adversary (line 251). The list ND[ℓ] records the sequence of already-observed nonces for session ℓ, which is used to compute the acceptable nonces using the policy $N\mathrm{x}$. The lines 255 -256 truncate this nonce history to only that which is needed to compute $N\mathrm{x}$. Lastly, the associative arrays AD and A[ℓ] keep track of globally registered ADs and ADs registered specific to session ℓ respectively.

3.2. The NonceWrap Scheme

Ciphertext Structure. Encryption under NonceWrap is illustrated in Fig. 3.2. The construction uses two main primitives: an n-bit blockcipher E and an nAE scheme $\mathcal{E}$. The blockcipher is invoked once per encryption call, while the nAE scheme does the majority of the work. NonceWrap also employs a hash function H, but it is used only for processing the AD. The hash function only outputs a few bits as we do not seek collision-resistance from it. In fact, we do not depend on any security property of H whatsoever. A poor choice of H (like the constant function) would only slow down decryption (in the case where there are multiple AD values for each session), but would

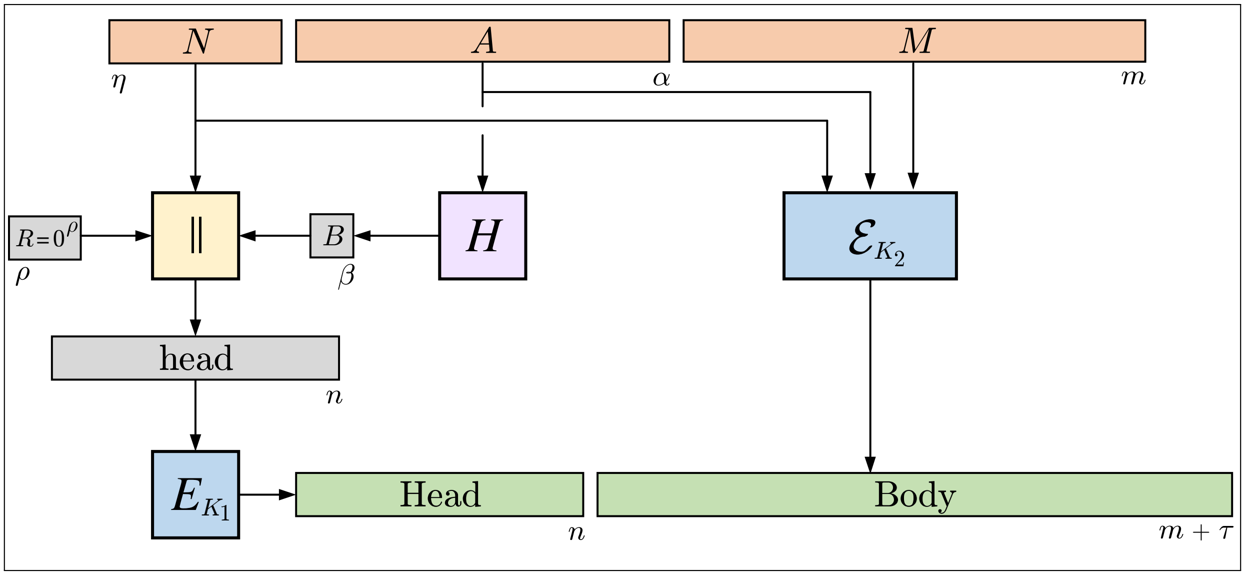

FIGURE 3.2. **Scheme illustration.** NonceWrap encryption outputs a ciphertext that consists of two parts: a header Head, which is produced from a blockcipher E, and a body Body, which is produced from an nAE scheme $\mathcal{E}$. The hashed AD in the header can be omitted in the customary case where there is one AD per session at any time.

otherwise have no adverse effect. If the 1AD/session restriction is being followed, then NonceWrap does not need to use a hash function at all.

There are two parts to a NonceWrap-produced ciphertext: a header and a body. The header Head would usually be sixteen bytes. It not only encodes the nonce N, which would typically be twelve bytes [**43**], but also some redundancy (a length ρ string of zero bits) and a hash of the AD. The body Body is created by standard nAE encryption using the given key, nonce, AD, and message. To get the *full* ciphertext, NonceWrap simply prepends the header to the body. We stress that this is the *full* ciphertext as this is all the encrypting party needs to send to the decrypting one. In total, the length of the ciphertext for a message M is $|M| + \lambda + \tau$ where λ is the length of the header and τ is the expansion of the nAE scheme. For now, the header length $\lambda = n$ is the same as the blocksize of E.

NonceWrap employs distinct keys for the blockcipher and the nAE scheme. The two keys together make up a single NonceWrap session key.

Upon receiving a ciphertext $C = \text{Head} \parallel \text{Body}$, a receiver will often be able to determine that the ciphertext does not belong to a candidate session just by examining the prefix Head. The header is deciphered using the candidate session key and if the mandated string of redundant zero bits is not present, then the receiver immediately knows that this ciphertext cannot be for the candidate key. If the zero bits are present, it can check whether the deciphered nonce is within the session's nonce policy, or whether the deciphered AD hash is the hash of an AD registered for the session. If any of these checks fail, then it knows that the ciphertext as a whole is not associated with the candidate session. If all of them pass, then the body Body itself still needs to be decrypted under the session's key, which can very well still reject the ciphertext. But, the point that we want to emphasize is that the receiver may skip trial-decryption for the body if these checks on the header do not pass. This is valuable as trial-decryption may be expensive, especially if the ciphertext is long.

If one is employing the 1AD/session assumption, then the hash of the AD can be omitted (which is equivalent to treating the hash function as a constant function that always returns the empty string). With a 16-byte header encrypting a 12-byte nonce, there would then be a 4-byte string of zeroes. From this, there is roughly a 2^{-32} chance that a header for one session would be considered a plausible candidate for another. When that does happen, trial-decryption of the ciphertext body Body is performed, not the attribution of the ciphertext to an incorrect session. For misattribution to occur, the ciphertext body would also have to verify as authentic when decrypted under the inccorect key. For an nAE-secure encryption scheme and two honestly chosen random keys, it is difficult to find two distinct explanations of the same encrypted message. This is a result that we show in our later chapter on committing AE.

Precomputing Headers. Since the header is computed from the nonce and the AD, it is possible for the receiver to precompute a header before it actually arrives. This is possible because the receiver knows what nonces are within the policy its employing and it knows what AD values are registered to particular sessions and globally across sessions. Recall that AD values in anAE schemes are known to the receiver a priori to decryption, so it would be able to register the AD values it is willing to accept. For efficiency purposes, NonceWrap will find it useful to precompute headers in this way.

Suppose that the protocol is using the 1AD/session assumption, and thus the AD component of header generation is omitted. Even in this case, the number of potential headers to precompute would be large if the breadth of the nonce policy is large– for example, under the permissive policy $Nx(\Lambda) = \mathcal{N}$. It is not practical to precompute headers under such a policy. To get around this, we introduce another function that yields the *anticipated* (or *likely*) nonces, Lx. The function, given the last few nonces received so far (some history), returns a set of nonces that are likely to come next. Like the nonce policy function Nx, the signature of the anticipated-nonce function is $Lx\colon \mathcal{N}^{\leq d} \to \mathcal{P}(\mathcal{N})$. While its signature is identical to that of Nx, it is distinct from Nx, which names the set of nonces that are *permissible* given a history of received nonces as opposed to naming what is *likely.* We keep in mind that anything outside of Nx's returned set should be considered inauthentic. As such, we require that anything that is likely is also permissible: $Lx(\boldsymbol{n}) \subseteq Nx(\boldsymbol{n})$ for all $\boldsymbol{n} \in \mathcal{N}^{\leq d}$.

We call an anAE scheme *sharp* if it employs nonce policy and anticipated nonce functions such that $Lx = Nx$. For a sharp scheme, any nonce that it does not anticipate is considered out of policy. Through precomputing headers by anticipating incoming nonces and leveraging a priori knowledge of ADs, NonceWrap will enjoy some efficiency benefits in the general case. Sharpness can improve efficiency even more.

NonceWrap Data Structures. We now delve more deeply into the structure of NonceWrap. The construction is defined in Fig. 3.3 and a list of the data structures it employs is given in Fig. 3.4. The scheme in full NonceWrap$[E, H, \mathcal{E}, Lx, Nx]$ is built from a blockcipher $E\colon \{0,1\}^{k_1} \times \{0,1\}^n \to \{0,1\}^n$, a hash function $H\colon \{0,1\}^* \to \{0,1\}^\beta$, an nAE scheme $\mathcal{E}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$, a nonce policy $Nx\colon \{0,1\}^{\leq d} \to \mathcal{P}(\mathcal{N})$, and an anticipated nonce function Lx (with the same signature as that of Nx), which always outputs a subset of what Nx permits.

NonceWrap heavily utilizes dictionaries for efficiency. The most significant of these dictionaries is $\boldsymbol{K}$.LNA, which maps headers to sets of (SID, nonce, AD) triples. (Recall that $\boldsymbol{K}$ is the state of NonceWrap). The set contains the triples that "explain" the header. When a session is initialized, the dictionary is populated with headers based on anticipated nonces from an empty nonce history and the set of globally registered ADs. When a session ℓ is torn down, all the triples associated with the terminated session are expunged from the dictionary. When a new global AD is registered, new headers are precomputed and entered into $\boldsymbol{K}$.LNA for each of the sessions and their currently anticipated nonces. When a session-specific AD is registered, new headers based on the registered

```
$\Pi$.Init($K$)
00 $\ell \leftarrow \boldsymbol{K}$.ctr++
01 $\boldsymbol{K}$.K1[$\ell$] $\|$ $\boldsymbol{K}$.K2[$\ell$] $\leftarrow K$
02   where $|\boldsymbol{K}.\mathrm{K1}[\ell]| = k_1$
03 $\boldsymbol{K}$.L $\xleftarrow{\cup} \{\ell\}$; $\boldsymbol{K}$.A[$\ell$] $\leftarrow \emptyset$
04 $\boldsymbol{K}$.N[$\ell$] $\leftarrow \Lambda$; $K1 \leftarrow \boldsymbol{K}$.K1[$\ell$]
05 for $N \in Lx(\Lambda)$ do
06   for ADs $\in \boldsymbol{K}$.AD.values do
07     for $A \in$ ADs do
08       head $\leftarrow N \| 0^\rho \| H(A)$
09       Head $\leftarrow E_{K1}$(head))
0A       $\boldsymbol{K}$.LNA[Head] $\xleftarrow{\cup} \{(\ell, N, A)\}$
0B ret $\ell$

$\Pi$.Term($\ell$)
10 for $S \in \boldsymbol{K}$.LNA.values do
11   $S \xleftarrow{\setminus} \{\ell\} \times \mathcal{N} \times \mathcal{A}$
12 $\boldsymbol{K}$.L $\xleftarrow{\setminus} \{\ell\}$

$\Pi$.Asso($A$)
20 $B \leftarrow H(A)$; $\boldsymbol{K}$.AD[$B$] $\xleftarrow{\cup} \{A\}$
21 for $\ell \in \boldsymbol{K}$.L do
22   for $N \in Lx(\boldsymbol{K}.\mathrm{N}[\ell])$ do
23     Head $\leftarrow E_{\boldsymbol{K}.\mathrm{K1}[\ell]}(N \| 0^\rho \| B)$
24     $\boldsymbol{K}$.LNA[Head] $\xleftarrow{\cup} \{(\ell, N, A)\}$
$\Pi$.Asso($A, \ell$)
25 $B \leftarrow H(A)$; $\boldsymbol{K}$.A[$\ell$][$B$] $\xleftarrow{\cup} \{A\}$
26 for $N \in Lx(\boldsymbol{K}.\mathrm{N}[\ell])$ do
27   Head $\leftarrow E_{\boldsymbol{K}.\mathrm{K1}[\ell]}(N \| 0^\rho \| B)$
28   $\boldsymbol{K}$.LNA[Head] $\xleftarrow{\cup} \{(\ell, N, A)\}$

$\Pi$.Disa($A$)
30 $B \leftarrow H(A)$; $\boldsymbol{K}$.AD[$B$] $\xleftarrow{\setminus} \{A\}$
31 for $S \in \boldsymbol{K}$.LNA.values do
32   $S \xleftarrow{\setminus} \mathcal{L} \times \mathcal{N} \times \{A\}$
$\Pi$.Disa($A, \ell$)
33 $B \leftarrow H(A)$; $\boldsymbol{K}$.A[$\ell$][$B$] $\xleftarrow{\setminus} \{A\}$
34 for $S \in \boldsymbol{K}$.LNA.values do
35   $S \xleftarrow{\setminus} \{\ell\} \times \mathcal{N} \times \{A\}$

$\Pi$.Enc($K, N, A, M$)
40 $K_1 \| K_2 \leftarrow K$ where $|K_1| = k_1$
41 Head $\leftarrow E_{K_1}(N \| 0^\rho \| H(A))$
42 Body $\leftarrow \mathcal{E}(K_2, N, A, M)$
43 ret $C \leftarrow$ Head $\|$ Body

$\Pi$.Dec($C$)          Phase-1 (starting at 51)
50 Head $\|$ Body $\leftarrow C$ where $|\mathrm{Head}| = n$
51 for $(\ell, N, A) \in \boldsymbol{K}$.LNA[Head] do
52   $K_1 \leftarrow \boldsymbol{K}$.K1[$\ell$]; $K_2 \leftarrow \boldsymbol{K}$.K2[$\ell$]
53   $M \leftarrow \mathcal{D}(K_2, N, A, \mathrm{Body})$
54   if $M \neq \bot$ then goto 5F

55 for $\ell \in \boldsymbol{K}$.L do          Phase-2
56   $K_1 \leftarrow \boldsymbol{K}$.K1[$\ell$]; $K_2 \leftarrow \boldsymbol{K}$.K2[$\ell$]
57   $N \| R \| B \leftarrow E^{-1}_{K_1}$(Head)
58     where $|N| = \eta$ and $|R| = \rho$
59   if $R \neq 0^\rho$ or $N \notin Nx(\boldsymbol{K}.\mathrm{N}[\ell])$ then
5A       continue
5B   for $A \in \boldsymbol{K}.\mathrm{A}[\ell][B] \cup \boldsymbol{K}.\mathrm{AD}[B]$ do
5C     $M \leftarrow \mathcal{D}(K_2, N, A, \mathrm{Body})$
5D     if $M \neq \bot$ then goto 5F
5E ret $\bot$

5F Old $\leftarrow Lx(\boldsymbol{K}.\mathrm{N}[\ell])$          Phase-3
5G $\boldsymbol{K}$.N[$\ell$] $\xleftarrow{\|} N$
5H if $|\boldsymbol{K}.\mathrm{N}[\ell]| > d$ then
5I   $\boldsymbol{K}$.N[$\ell$] $\leftarrow$ tail($\boldsymbol{K}$.N[$\ell$])
5J New $\leftarrow Lx(\boldsymbol{K}.\mathrm{N}[\ell])$
5K for $N' \in$ Old $\setminus$ New do
5L   for $S \in \boldsymbol{K}$.LNA.values do
5M     $S \xleftarrow{\setminus} \{\ell\} \times \{N'\} \times \mathcal{A}$
5N for $N' \in$ New $\setminus$ Old do
5O   for $B \in \boldsymbol{K}.\mathrm{A}[\ell].\mathrm{keys} \cup \boldsymbol{K}.\mathrm{AD}.\mathrm{keys}$ do
5P     Head $\leftarrow E_{K_1}(N' \| 0^\rho \| B)$
5Q     for $A' \in \boldsymbol{K}.\mathrm{A}[\ell][B] \cup \boldsymbol{K}.\mathrm{AD}[B]$ do
5R       $\boldsymbol{K}$.LNA[Head] $\xleftarrow{\cup} \{(\ell, N', A')\}$
5S ret $(\ell, N, A, M)$
```

FIGURE 3.3. **NonceWrap.** The scheme $\Pi = \mathsf{NonceWrap}[E, H, \mathcal{E}, Lx, Nx]$. Data structures employed are described in Fig. 3.4.

$\boldsymbol{K}$.L	Set of SIDs
$\boldsymbol{K}$.K1	Dictionary mapping an SID to a key for the blockcipher E
$\boldsymbol{K}$.K2	Dictionary mapping an SID to a key for the nAE scheme $\mathcal{E}$
$\boldsymbol{K}$.N	Dictionary mapping an SID to a list of nonces
$\boldsymbol{K}$.A	Dictionary mapping an SID to a dict. mapping a hashed AD to a set of ADs
$\boldsymbol{K}$.AD	Dictionary mapping a hashed AD to a set of ADs
$\boldsymbol{K}$.LNA	Dictionary mapping a header to a set of (SID, nonce, AD) triples

FIGURE 3.4. **Data structures employed for NonceWrap.** To achieve good decryption-time efficiency, NonceWrap employs a set ADT and multiple dictionaries, one of which has dictionary-valued entries. Some simplifications are possible for the customary case of 1AD/session.

AD are computed just for that session and its anticipated nonces. Removing an AD through `Disa` will remove all the triples associated with that AD from $\boldsymbol{K}$.LNA. We note that the nonces in the triples recorded in $\boldsymbol{K}$.LNA: all the nonces must belong to an active session's *anticipated* nonce set. From this and the way headers and tuples are entered and removed from $\boldsymbol{K}$.LNA, an invariant property of the dictionary arises: for any tuple (ℓ, N, A) extracted from $\boldsymbol{K}$.LNA using any header as a key, it must be the case that ℓ is an active session, N is within the nonce policy of ℓ by Nx, and A is registered either globally or specifically to ℓ.

There are two dictionaries used for managing AD values. The straightforward one is the dictionary $\boldsymbol{K}$.AD, which maps hashes of AD values to sets of AD values. As it does not manage anything related to SIDs, this dictionary is reserved specifically for global ADs The other dictionary, $\boldsymbol{K}$.A maps an SID to *another dictionary* that maps hashed ADs to sets of AD values.

There is some complexity to these dictionaries as there is some data structure nesting. The reason that the dictionaries use sets as their values (in terms of their key and value pairs) is that it is possible for there to be multiple explanations of the same header or AD hash. For the header, after q headers are inserted into $\boldsymbol{K}$.LNA, a rough upper bound of there being a collision is $q^2/2^{n+1}$. Since headers are made with the blockcipher E, multiple headers for the same session will never collide because they are enciphered under the same key. The only effect of header collisions is that the receiver may have to perform more nAE decryptions as it must do a trial-decryption for each collided header. Any collisions in AD hashing depends on the hash function used. We make no assumptions of the hash function and any collisions resulting from it only impact efficiency (the receiver might have to execute multiple trial-decryptions in case of collision). There is also the dictionary $\boldsymbol{K}.\mathrm{A}$ that maps SIDs to other dictionaries. Since SIDs are assumed to be unique, one can think of this as creating a dictionary like the global AD dictionary AD for each session that keeps track of session-specific ADs.

The remaining data structures are straightforward. The dictionary $\boldsymbol{K}.\mathrm{N}$ keeps track of this history of received nonces for each session. The dictionaries $\boldsymbol{K}.\mathrm{K}_1$ and $\boldsymbol{K}.\mathrm{K}_2$ map SIDs to blockcipher and nAE scheme keys respectively. These do not have to worry about collisions due to the uniqueness of SIDs. Finally, $\boldsymbol{K}.\mathrm{L}$ is a simple set of all active SIDs.

NonceWrap Decryption. Decryption is the most complicated of NonceWrap's algorithm. Expectedly so, as the anAE definition requires schemes to recover the SID, the nonce, the AD, and the plaintext from just the ciphertext.

NonceWrap decryption can be broken into three phases. Phase-1 attempts to decrypt the ciphertext in the most efficient way. Upon receiving a ciphertext $C = \mathrm{Head} \parallel \mathrm{Body}$, decryption looks up the header Head in its dictionary $\boldsymbol{K}.\mathrm{LNA}$, which would contain the precomputed headers with anticipated nonces. If the header is there, then decryption immediately knows the SID, nonce, and AD triples that are associated with the header. There is an entire set of these triples, but we do not expect the set to be large. Decryption then iterates over this small set and trial-decrypts the ciphertext body using the SID, nonce, and AD the triples provide. If one of the nAE decryption attempts succeeds, then the message has been recovered and the procedure skips to phase-3.

If all the trial-decryptions fail, or if Head is not in $\boldsymbol{K}.\mathrm{LNA}$ (perhaps the nonce used for Head was not anticipated), then NonceWrap will try to decrypt with phase-2. This phase iterates over every session, using each session key to trial-decipher the header. For a candidate deciphered header, it checks to see whether there are ρ 0-bits present and whether the nonce found is within the candidate session's policy. If not, NonceWrap can quickly reject this session for this ciphertext without having to process the ciphertext body. If so, NonceWrap tries all the ADs, both global and specific to the candidate session, associated with the hash value found in the deciphered header and decrypts the body. If no valid decryption is found, then decryption outputs $\perp$, rejecting this ciphertext for all of NonceWrap's active sessions. If one of the attempts is successful, then the message has been recovered and decryption goes into phase-3.

Phase-3 is where precomputation of the next anticipated headers is done. Entering phase-3 means that decryption knows the (ℓ, N, A, M) for the queried ciphertext regardless of which of the previous phases it came in from. After updating the nonce history for session ℓ using N, $\mathsf{NonceWrap}$ can then compute the next set of anticipated nonces using Lx, a set called New in the code. With the newly anticipated nonces, the $\mathsf{NonceWrap}$ can compute new headers and add them to $\boldsymbol{K}$.LNA as the next expected headers. It also removes any headers that have to do with nonces that are no longer anticipated, which is the set Old $\setminus$ New where Old is the set of nonces anticipated with the nonce history before the update. Because of this, the only headers that populate $\boldsymbol{K}$.LNA at any point in time are headers that the receiver expects to receive next.

Efficiency. Let s denote the maximum number of active sessions. Let t be the time it takes to compute the E or E^{-1}. Assume an anticipated-nonce policy Lx whose breadth is a small constant. Assume the maximum number of AD values registered either globally or to any one session is a small constant. Assume an amount of redundancy $\rho \in O(\lg s)$ used to create headers. Assume the nAE scheme $\mathcal{E}$ uses time $O(m + a)$ to decrypt a length $m + \tau$ ciphertext with AD A. Assume a nonce can be checked as being in-policy, according to Nx, in constant time. Assume dictionaries are implemented in some customary way, with expected log-time operations. Then the expected time to decrypt a valid ciphertext that used an anticipated nonce will be $O(m + a + t + \lg s)$. The expected time to decrypt an invalid ciphertext, or a valid ciphertext that used an unanticipated nonce, will be $O(m + a + st)$.

For any sharp scheme, phase-2 is never executed as any nonce not anticipated is not within policy. Hence, the headers recorded in $\boldsymbol{K}$.LNA consist of everything that is acceptable. This results in sharp schemes having a decryption time of $O(m + a + t + \lg s)$ for any ciphertext. This does not just improve run-time efficiency; we can leverage sharpness to improve bandwidth efficiency as well. Notice that without running phase-2, the blockcipher's inverse E^{-1} is never computed. As such, the blockcipher $E_{K_1}\colon \{0,1\}^n \to \{0,1\}^n$ can be replaced by a PRF $F_{K_1}\colon \{0,1\}^n \to \{0,1\}^\lambda$ where λ is considerably smaller than n. Only one or two bytes is necessary for λ as the header collisions only cause the receiver to have to do more than one trial-decryptions. Compare this to plain old nAE where a sender may be required to transmit the 12-byte nonce in the clear. By encoding the nonce with the PRF, we can actually save significant bandwidth while hiding it from observers of the transmission.

3.3. NonceWrap Security

Multi-key strong-PRP security. We will find it convenient to define a notion of multi-key strong PRP security, which we denote as prp$*$-security. In customary strong PRP security, like conventional PRP security, the adversary has access to a forward direction oracle that computes a real or ideal permutation. Strong PRP security adds a backward direction oracle that computes the inverse. To adapt this to the multi-key setting, we treat the PRP as a length-preserving PRI. Define $\mathrm{Inj}^{\mathcal{T}}(\{0,1\}^n) = \mathrm{Inj}_0^{\mathcal{T}}(\{0,1\}^n)$. For an adversary $\mathcal{A}$, we define its advantage in attacking the prp$*$-security of an n-bit PRP E as the real number

$$\begin{aligned} \mathbf{Adv}_E^{\mathrm{prp}*}(\mathcal{A}) \;\; = \;\; & \Pr[\text{for } i \in \mathbb{N} \text{ do } K_i \twoheadleftarrow \mathcal{K}\text{: } \mathcal{A}^{F_{\boldsymbol{K}}(\cdot,\cdot), G_{\boldsymbol{K}}(\cdot,\cdot)} \Rightarrow 1] - \\ & \Pr[p \twoheadleftarrow \mathrm{Inj}^{\mathbb{N}}(\{0,1\}^n)\text{: } \mathcal{A}^{F_p(\cdot,\cdot), G_p(\cdot,\cdot)} \Rightarrow 1] \end{aligned}$$

where the oracles behave as follows: oracle $F_{\boldsymbol{K}}$, on query (i, X), returns $E(K_i, X)$; oracle $G_{\boldsymbol{K}}$, on query (i, X), returns $E^{-1}(K_i, X)$; oracle F_p, on query (i, X), returns $p(i, X)$; and oracle G_p, on query (i, X), returns $p^{-1}(i, X)$.

NonceWrap security. To show the security of NonceWrap, we establish that its anae-security is good if E is prp$*$-secure and $\mathcal{E}$ is nae$*$-secure.

Theorem 3.3.1. *There exists a reduction* R*, explicitly given in the proof of this theorem, as follows: Let* $E\colon \{0,1\}^{k_1} \times \{0,1\}^n \to \{0,1\}^n$ *be a blockcipher, let* $H\colon \{0,1\}^* \to \{0,1\}^\beta$ *be a hash function, let* $\mathcal{E} : \mathcal{K}_{\mathcal{E}} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}_{\mathcal{E}}$ *be an nAE scheme, and let* $\mathrm{Nx} : \mathcal{N}^{\leq d} \to \mathcal{P}(\mathcal{N})$ *be a nonce policy with depth* d*. Let* Lx *be an anticipated-nonce function with the same signature as* Nx *such that* $\mathrm{Lx}(\boldsymbol{n}) \subseteq \mathrm{Nx}(\boldsymbol{n})$ *for all* $\boldsymbol{n} \in \mathcal{N}^{\leq d}$*. Let* $\Pi = \mathsf{NonceWrap}[E, H, \mathcal{E}, \mathrm{Lx}, \mathrm{Nx}]$ *be a* NonceWrap *scheme. Let* σ *be the expansion of* Π *and* τ *be the expansion of* $\mathcal{E}$*. Let* $\mathcal{A}$ *be an adversary that attacks* Π*. Then* R *transforms* $\mathcal{A}$ *into a pair of adversaries* ($\mathcal{B}_1$,$\mathcal{B}_2$) *such that*

$$\mathbf{Adv}^{\mathrm{anae}}_{\Pi,Nx}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prp*}}_{E}(\mathcal{B}_1) + \mathbf{Adv}^{\mathrm{nae*}}_{\mathcal{E}}(\mathcal{B}_2) + \frac{q_e^2}{2^{n+1}} + \frac{q_e^2}{2^{\tau+1}} + \frac{q_e^2 + q_d^2}{2^{\sigma+1}} + \frac{q_e^4}{2^{n+\tau+2}}$$

where q_e and q_d are the number of encryption and decryption queries that $\mathcal{A}$ makes. The resource usage of $\mathcal{B}_1$ and $\mathcal{B}_2$ are similar to that of $\mathcal{A}$.

Proof. We define a sequence of hybrid games that transition the real anae game to the ideal anae game, where the games are using Π and Nx. The first of these hybrids, G_1 replaces the blockcipher E with a random function P from $\mathrm{Inj}^{\mathbb{N}}(\{0,1\}^n)$. Note that $P(i,\cdot)$ is an injection for all $i \in \mathbb{N}$ and is length-preserving, so it is a permutation. We construct an adversary $\mathcal{B}_1$ that attacks the blockcipher E by having it simulate these two games. Whenever $\mathcal{A}$ makes a query, $\mathcal{B}_1$ follows the protocol defined in the real anae game. If the query requires a blockcipher operation, $\mathcal{B}_1$ would query its own forward direction oracle and use that output for the operation instead. It can use its backward direction oracle for inverting the blockcipher. At the end, $\mathcal{B}_1$ outputs the same bit $\mathcal{A}$ returns. The advantage of $\mathcal{B}_1$ is equivalent to $\mathcal{A}$'s advantage in distinguishing the games it simulates as the ciphertexts that the simulated encryption oracles would produce would be identical with the exception of the header, which depends on whether $\mathcal{B}_1$'s oracle is using P or the real blockcipher E. With that, we have:

$$\mathbf{Pr}[\ \mathcal{A}^{\mathrm{Real}^{\mathrm{anae}}_{\Pi}}\ ] - \mathbf{Pr}[\ \mathcal{A}^{\mathsf{G}_1}\ ] \leq \mathbf{Adv}^{\mathrm{prp*}}_{E}(\mathcal{B}_1)$$

The next hybrid G_2 replaces $\mathsf{NonceWrap}$'s underlying nAE scheme $\mathcal{E}$ with a random function F from $\mathrm{Inj}^{\mathbb{N}\times\mathcal{N}\times\mathcal{A}}_{\tau}(\mathcal{M})$. We construct an adversary $\mathcal{B}_2$ that attacks the nAE scheme by simulating the two hybrid games. Like $\mathcal{B}_1$, adversary $\mathcal{B}_2$ will just follow protocol except it replaces any nAE operations with its oracles. For any blockcipher operations, it simulates P as described in the previous step. It returns the same bit that $\mathcal{A}$ returns. The advantage of $\mathcal{B}_2$ is equivalent to $\mathcal{A}$'s advantage in distinguishing the games it simulates as the only difference between the simulated games is how the ciphertext body is produced, which depends on whether $\mathcal{B}_2$'s oracle is using F or the real nAE scheme $\mathcal{E}$. With that, we have:

$$\mathbf{Pr}[\, \mathcal{A}^{\mathsf{G}_2}\,] - \mathbf{Pr}[\, \mathcal{A}^{\mathsf{G}_3}\,] \leq \mathbf{Adv}_{\mathcal{E}}^{\text{nae}*}(\mathcal{B}_2)$$

At this point we have a real anae game using a $\mathsf{NonceWrap}$ scheme built on ideal primitives and we want to measure how well $\mathcal{A}$ can distinguish it from the ideal anae game. For the upcoming parts, we modify the ideal game step-by-step until it is indistinguishable from the real game.

The first hybrid, G_7, makes a simple change to the decryption oracle. Referring to the code in Fig. 3.1, on line 251, there is a condition that the tuple in the history must be unique. This hybrid simply removes the "unique" condition. Instead, if there are multiple valid tuples that map to a queried ciphertext, the oracle will return the lexicographically first tuple instead of returning $\perp$. Clearly, to distinguish between G_7 and the ideal game, $\mathcal{A}$ would need to call decryption on a ciphertext with multiple valid tuples as the former would return a tuple and the latter would return $\perp$. The probability that this occurs is upper-bounded by the probability that two ciphertexts from encryption are the same as multiple tuples need to be mapped to the same ciphertext in H for there to be multiple valid tuples. Hence, the advantage $\mathcal{A}$ has for distinguishing between these two games is

$$\mathbf{Pr}[\, \mathcal{A}^{\text{Ideal}_{\Pi}^{\text{anae}}}\,] - \mathbf{Pr}[\, \mathcal{A}^{\mathsf{G}_7}\,] \leq \frac{q_e^2}{2^{\sigma+1}}$$

The next modification only changes how ciphertexts are generated. Instead of randomly sampling from $\{0,1\}^{|M|+\tau}$ on an encryption query, the encryption oracle will instead use a pair of PRIs to generate a "header" and "body" to create the ciphertext. To do this, we modify the code for the ENC oracle to use the procedure F defined in the top half of Fig. 3.5. The bottom half of the figure shows the modified encryption oracle. The procedure captures the lazy-sampling of the forward direction of a random function or injection depending on whether the code in grey is executed. Without the grey, the code simulates a function for each tweak T; With the grey, it simulates an injection for each T. Having that, we can use F to capture the pair of PRIs: one from $\text{Inj}_{\tau}^{\mathbb{N}\times\mathcal{N}\times\mathcal{A}}(\mathcal{M})$ for creating the body and one from $\text{Inj}^{\mathbb{N}}(\{0,1\}^n)$ for creating the header.

We can think of G_7 as using two different instances of F, which we label as F_E and $F_{\mathcal{E}}$, without the grey to generate a header and body and concatenating the two results. This is the same as

```
procedure $F(T, X)$
if $X \parallel 0^{w-u} \in \mathrm{dom}(f(t, \cdot))$ then
    ret $f(T, X)$
$Y \twoheadleftarrow \{0,1\}^v$
if $Y \in \mathrm{range}(f(T, \cdot))$ then
    bad ← true
    $Y \twoheadleftarrow \{0,1\}^v \setminus \mathrm{range}(f(T, \cdot))$
$f(T, X \parallel 0^{w-u}) \leftarrow Y$
ret $Y$
```

```
procedure $F^{-1}(T, Y)$
if $Y \in \mathrm{range}(f(T, \cdot))$
    $X' \parallel R \leftarrow f^{-1}(T, Y)$
        where $|X'| = u$
    if $R = 0^{w-u}$ then ret $X'$
    ret $\perp$
$X \twoheadleftarrow \{0,1\}^w$
if $X \in \mathrm{dom}(f(T, \cdot))$ then
    bad ← true
    $X \twoheadleftarrow \{0,1\}^v \setminus \mathrm{dom}(f(T, \cdot))$
$f(T, X) \leftarrow Y$
$X' \parallel R \leftarrow X$ where $|X'| = u$
if $R = 0^{w-u}$ then ret $X'$
ret $\perp$
```

```
procedure $\mathsf{G}_6.\mathrm{ENC}(\ell, N, A, M)$
if $\ell \notin \mathrm{L}$ or $N \in \mathrm{NE}[\ell]$ then ret $\perp$
$\mathrm{NE}[\ell] \overset{\cup}{\leftarrow} \{N\}$
$\mathrm{Head} \leftarrow F_E(\ell, N \parallel 0^\rho \parallel H(A))$
$\mathrm{Body} \leftarrow F_{\mathcal{E}}((\ell, N, A), M)$
$C \leftarrow \mathrm{Head} \parallel \mathrm{Body}$
$\mathrm{H}[C] \overset{\cup}{\leftarrow} \{(\ell, N, A, M)\}$; ret $C$
```

FIGURE 3.5. **Top.** Lazy-sampling of random functions or injections in the multi-key setting. With the code in grey, the procedures simulate a random injection for each T from u bits to w bits. Without the code in grey, the procedures simulate a random function for each T. **Bottom.** Modified encryption oracle that uses either random functions or random injections to generate the ciphertext. Here, $\rho = n - \eta - \beta$ where η is the length of the nonce. The game using injections is called G_6.

generating a random string of the same length since queries to the encryption oracle can't be repeated, so a random header and body is sampled each time. When we replace the random ciphertext generation with the pair of PRIs, we use F_E and $F_{\mathcal{E}}$ with the grey code. We refer to the game using F for the PRIs as G_6.

To distinguish between G_6 and G_7, $\mathcal{A}$ would need to distinguish the difference between F with and without the grey code. This is the probability that bad gets set to true in F. For now, we don't need to worry about F^{-1} as the adversary has no way of accessing it. On the ith encryption query, the probability that bad gets set to true is at most $(i-1)/2^w$. It follows that the probability

```
procedure $\mathsf{G}_5$.DEC($C$)
550 Head || Body $\leftarrow C$ where $|\text{Head}| = n$
551 for $\ell \in \text{L}$ do
552     for $N \in N\text{x}(\text{ND}[\ell])$ do
553         for $A \in \text{A}[\ell] \cup \text{AD}$ do
554             $M \leftarrow F_{\mathcal{E}}^{-1}((\ell, N, A), \text{Body})$
555             if $(\ell, N, A, M) \in \text{H}[C]$ then
556                 $\text{ND}[\ell] \overset{\|}{\leftarrow} N$
557                 if $\text{ND}[\ell] \notin \text{dom}(N\text{x})$ then $\text{ND}[\ell] \leftarrow \text{tail}(\text{ND}[\ell])$
558                 ret $(\ell, N, A, M)$
559 ret $\perp$
```

Figure 3.6. **G_5's decryption oracle.** This decryption oracle searches for a (ℓ, N, A) triple to use to recover M. It then validates the resulting quadruple by making sure that it maps to the ciphertext in the history H.

bad gets set to true is at most $q_e^2/2^{w+1}$ for q_e encryption queries. The adversary may observe this event in either F_E or $F_{\mathcal{E}}$. Thus, $\mathcal{A}$'s advantage here is

$$\mathbf{Pr}[\ \mathcal{A}^{\mathsf{G}_7}\ ] - \mathbf{Pr}[\ \mathcal{A}^{\mathsf{G}_6}\ ] \le \frac{q_e^2}{2^{n+1}} + \frac{q_e^2}{2^{\tau+1}}$$

Our next hybrid G_5 changes the decryption oracle and is shown in Fig. 3.6. The other oracles remain the same. Instead of identifying the SID, nonce, and AD using $\text{H}[C]$ right away, the oracle will search for the tuple by going through all $\ell \in \text{L}$, $N \in N\text{x}(\text{ND}[\ell])$, and $A \in \text{A}[\ell] \cup \text{AD}$. For each of those tuples, it will try to invert the injection on Body to recover M. Now it's possible that the inversion results in an M that wasn't recorded in H since F^{-1} as defined in Fig. 3.5 can return values that weren't given by the forward oracle. However, we check on line 555 to make sure that the (ℓ, N, A, M) we found is actually mapped to C, which is something required to return a valid tuple in G_6's decryption. The other validity conditions on ℓ, N, and A are already accounted for since we iterate through the sets that validate them. We also iterate through them in lexicographic order, which guarantees that if there are multiple valid tuples, we return the lexicographically first one. Essentially, G_5 does the same as G_6's decryption; it just does it in a roundabout way by searching for the tuple. Hence, G_5 and G_6 are indistinguishable from each other to $\mathcal{A}$.

Instead of looping through the permitted nonces and ADs, we can use the header to figure out the nonce and AD. The header as generated in the previous hybrid's encryption contains the nonce

```
procedure $\mathsf{G}_4$.ASSO($A$)
  $B \leftarrow H(A)$; $\mathrm{AD}[B] \xleftarrow{\cup} \{A\}$
procedure $\mathsf{G}_4$.ASSO($A, \ell$)
  $B \leftarrow H(A)$; $\mathrm{A}[\ell][B] \xleftarrow{\cup} \{A\}$

procedure $\mathsf{G}_4$.DISA($A$)
  $B \leftarrow H(A)$; $\mathrm{AD}[B] \xleftarrow{\setminus} \{A\}$
procedure $\mathsf{G}_4$.DISA($A, \ell$)
  $B \leftarrow H(A)$; $\mathrm{A}[\ell][B] \xleftarrow{\setminus} \{A\}$

procedure $\mathsf{G}_4$.DEC($C$)    Resembles phase-2
  $\mathrm{Head} \parallel \mathrm{Body} \leftarrow C$ where $|\mathrm{Head}| = n$
  for $\ell \in \mathrm{L}$ do
    $N \parallel R \parallel B \leftarrow F_E^{-1}(\ell, \mathrm{Head})$
      where $|N| = \eta$ and $|R| = r$
    if $R \neq 0^\rho$ or $N \notin N\mathrm{x}(\mathrm{ND}[\ell])$
      then continue
    for $A \in \mathrm{A}[\ell][B] \cup \mathrm{AD}[B]$ do
      $M \leftarrow f_{\mathcal{E}}^{-1}((\ell, N, A), \mathrm{Body})$
      if $(\ell, N, A, M) \in \mathrm{H}[C]$ then
        $\mathrm{ND}[\ell] \xleftarrow{\|} N$
        if $\mathrm{ND}[\ell] \notin \mathrm{dom}(N\mathrm{x})$ then
          $\mathrm{ND}[\ell] \leftarrow \mathrm{tail}(\mathrm{ND}[\ell])$
        ret $(\ell, N, A, M)$
  ret $\perp$
```

FIGURE 3.7. G_4**'s decryption oracle.** This decryption oracle resembles phase-2 of NonceWrap. Functionally, it does what the ideal decryption oracle does except instead of looking up a valid tuple in the ciphertext history it iterates through every possibility to search for one.

and a hash of the AD. This is just like in NonceWrap encryption. We make modifications to the decryption oracle to do just this. For us to use the AD hash, we also need to modify the ASSO and DISA oracles. The result of these modifications leaves us with hybrid G_4, which is presented in Fig. 3.7.

Note that decryption now resembles phase-2 of NonceWrap decryption. It's clear that any session it returns is active and any nonce it returns is within the policy as the former is found through iteration and there is an explicit check of the latter. It's also clear that any AD that it returns is registered as $\mathrm{A}[\ell][B] \cup \mathrm{AD}[B]$ is a subset of all the ℓ's ADs and all the global ADs.

But does G_4 decryption always behave like G_5's decryption? If queried with a C that did not come from the encryption oracle, then both of them return $\perp$ as they both check to make sure $(\ell, N, A, M) \in \mathrm{H}[C]$ before returning a tuple. If queried with a C that did, assuming that C was made with an active session key, a nonce under the session's policy, and a properly registered AD, then both decryptions return the same tuple. It's clear that G_5 will find the first lexicographic tuple due to its iteration. If there's only one valid tuple explaining C, then, trivially, the first tuple is returned.

But if there are multiple valid tuples, what happens? If the tuples are under different SIDs, then we arrive at the lexicographically first SID by iteration. If the SIDs are the same, then the header is deciphered and the nonce and AD hash are found. This SID can only have one valid nonce mapped to this header since the header was generated by an injection. Even though G_5 doesn't decipher the header, it still checks the association between nonce and header since it checks whether the tuple is in $\mathrm{H}[C]$. This means that G_5, for a fixed session, can only find one nonce—the same nonce as G_4—that is in $\mathrm{H}[C]$ even if it iterated through the entirety of the policy. Similarly, the SID can only have one AD hash mapped to this header for the same reason. Even though G_5 iterates through all registered ADs, the ones that it finds that are in $\mathrm{H}[C]$ would have their hashes associated to the header. Since G_4 lexicographically iterates through the $\mathrm{A}[\ell][B] \cup \mathrm{AD}[B]$ subset of registered ADs, it would arrive at the same AD as G_5. Hence, G_5 and G_4 always arrive at the same result for a given ciphertext, making the two indistinguishable.

The next modification adds dictionary LNA from $\mathsf{NonceWrap}$ into the game. To start, suppose that we add LNA into the ideal game without actually using it for decryption yet. All other data structures that are needed to support LNA are already exist in our hybrids up to this point; we already manage the active SIDs in the set L and the nonce history of a session in $\mathrm{ND}[\ell]$. The structures for ADs were modified from sets into dictionaries in G_4, but we can still derive the set of all valid ADs for a session ℓ from them. The union of all sets in $\mathrm{A}[\ell].\mathrm{values} \cup \mathrm{AD}.\mathrm{values}$ is just that. We'll denote this set as $\mathcal{A}_\ell$. All of these data structures are needed to add or remove tuples from LNA. The code for this hybrid G_3 is presented in Fig. 3.8, but disregard the phase-1 decryption block for now. First, we want to assert a property of LNA.

Lemma 3.3.1. *Let* L, ND*,* A*,* AD*, and* LNA *be the data structures used in hybrid game* G_3*. Let* $\mathcal{X}$ *be the union of all sets in* LNA.values*. For any SID* ℓ*, let* $\mathcal{A}_\ell$ *be the union of all sets in* $\mathrm{A}[\ell].\mathrm{values} \cup \mathrm{AD}.\mathrm{values}$*. If* $(\ell, N, A) \in \mathcal{X}$ *then* $\ell \in \mathrm{L}$*,* $N \in N\mathrm{x}(\mathrm{ND}[\ell])$*, and* $A \in \mathcal{A}$*.*

Proof. Suppose there exists some $(\ell, N, A) \in \mathcal{X}$ such that one of the conditions described in the lemma is false. There are two ways that this can happen: either a value was added into LNA that violated one of the conditions or the condition itself was modified, but LNA was not modified accordingly. We exhaustively check for a case in which this can occur, specifically looking at when we add a tuple or modify the condition.

- Case: $(\ell, N, A) \in \mathcal{X}$ and $\ell \notin \mathrm{L}$.
    - When tuple is added in INIT, $\ell \in \mathrm{L}$ since INIT adds it to L.
    - When tuple is added in ASSO(A), $\ell \in \mathrm{L}$ since the procedure iterates through ℓ to add it.
    - When tuple is added in ASSO(A, ℓ), $\ell \in \mathrm{L}$ by assumption.
    - When tuple is added in DEC, $\ell \in \mathrm{L}$ since the tuple is added on successful decryption, which happens by iterating through L and finding ℓ.
    - When ℓ is removed from L, all tuples with ℓ as an element are removed from LNA.
- Case: $(\ell, N, A) \in \mathcal{X}$ and $A \notin \mathcal{A}_\ell$.
    - When tuple is added in INIT, $A \in \mathcal{A}_\ell$ since the procedure iterates through AD to get A.
    - When tuple is added in ASSO(A), $A \in \mathcal{A}_\ell$ since the procedure adds A to AD before adding the tuple to LNA.
    - When tuple added in ASSO(A, ℓ), $A \in \mathcal{A}_\ell$ since the procedure adds A to A[ℓ] before adding the tuple to LNA.
    - When tuple is added in DEC, $A \in \mathcal{A}_\ell$ since the procedure iterates through $\mathcal{A}_\ell$ to add each A.
    - When A is removed in DISA(A), all tuples with A as an element are removed from LNA.
    - When A is removed in DISA(A, ℓ), all tuples with both ℓ and A are removed from LNA. If a tuple containing A is still in $\mathcal{X}$, then it must have a different SID from ℓ.
- Case: $(\ell, N, A) \in \mathcal{X}$ and $N \notin Nx(\mathrm{ND}[\ell])$.
    - When tuple is added in INIT, $N \in Nx(\mathrm{ND}[\ell])$ since ND[ℓ] is initialized to the empty list and the procedure iterates over $Lx(\Lambda)$, which is a subset of $Nx(\Lambda)$.
    - When tuple is added in either ASSO, $N \in Nx(\mathrm{ND}[\ell])$ since the procedure iterates through each nonce in $Lx(\mathrm{ND}[\ell])$, which is a subset of $Nx(\mathrm{ND}[\ell])$.

– When tuple is added in DEC, ND[ℓ] is appended with a new nonce N' first. Two sets are generated here: $Lx(\text{ND}[\ell])$ and $Lx(\text{ND}[\ell] \parallel N')$. The former is Old and the latter is New in the pseudocode. The procedure iterates over New \ Old, which is a subset of $Nx(\text{ND}[\ell] \parallel N')$ when adding new tuples.

– When tuple is removed in DEC, the sets Old and New are used again. The procedure iterates over Old\New and removes tuples containing those nonces from LNA. Hence, any tuple with a nonce not in $Lx(\text{ND}[\ell] \parallel N')$ is removed.

None of these cases provide a situation where $(\ell, N, A) \in \mathcal{X}$ such that $\ell \notin \text{L}$, $N \notin Nx(\text{ND}[\ell])$, or $A \notin \mathcal{A}_\ell$. The lemma follows. □

As per lemma 3.3.1, we have that all tuples recorded in LNA satisfy the validity conditions in ideal decryption. Now when phase-1 decryption is accounted for in G_3 we observe that any successful decryption that occurs must have happened on a tuple in LNA, meeting the validity conditions. Here, success is defined as executing the **goto** instruction on line 354, which instructs the procedure to enter phase-3. The third phase does not modify the tuple being returned in any way; it only does bookkeeping to update the data structures, making sure that they are compliant to the validity conditions. So, whatever tuple was acquired in phase-1 would be returned. If no tuple was found in phase-1, the procedure will enter phase-2 where it iterates through every session as done in G_4's decryption. Whether the valid tuple (ℓ, N, A, M) being returned is found in phase-1 or phase-2, the conditions placed on each component of the tuple remains the same: ℓ must be in L, N must be in $Nx(\text{ND}[\ell])$, A must be in $\text{A}[\ell] \cup \text{AD}$, and the entire tuple must be in H[C]. Thus, G_3 decryption always returns a valid tuple under the same conditions as G_4.

However, in some cases, G_3 does not return the lexicographically first tuple. Suppose that the adversary makes two encryption queries with tuples T_1 and T_2 such that the tuples are different and their parameters are valid for decryption. Suppose it gets back the same ciphertext C both times. Let's say T_1 is the lexicographically first tuple, but its nonce is not within $Lx(\cdot)$. Let's say T_2's nonce *is* within $Lx(\cdot)$. When the adversary queries decryption with C, in G_4, it gets back T_1. On the other hand, it gets back T_2 in G_3 since phase-1 decryption would find T_2 first. The probability this occurs is upper-bounded by the probability of getting the same ciphertext from the encryption oracle, which occurs if the same header and body are outputted by their respective

```
procedure $\mathsf{G}_3$.INIT()
300 $K \twoheadleftarrow \mathcal{K}$
301 $\ell \leftarrow \Pi.\mathtt{Init}(K)$
302 $\mathrm{K}[\ell] \leftarrow K$; $\mathrm{L} \xleftarrow{\cup} \{\ell\}$; $\mathrm{NE}[\ell] \leftarrow \emptyset$
303 for $N \in Lx(\Lambda)$ do
304   for ADs $\in$ AD.values do
305     for $A \in$ ADs do
306       head $\leftarrow N \,\|\, 0^\rho \,\|\, H(A)$
307       Head $\leftarrow F_E(\ell, \text{head})$
308       LNA[Head] $\xleftarrow{\cup} \{(\ell, N, A)\}$
309 ret $\ell$

procedure $\mathsf{G}_3$.TERM($\ell$)
310 for $S \in$ LNA.values do
311   $S \xleftarrow{\setminus} \{\ell\} \times \mathcal{N} \times \mathcal{A}$
312 $\mathrm{L} \xleftarrow{\setminus} \{\ell\}$

procedure $\mathsf{G}_3$.ASSO($A$)
320 $B \leftarrow H(A)$; $\mathrm{AD}[B] \xleftarrow{\cup} \{A\}$
321 for $\ell \in \mathrm{L}$ do
322   for $N \in Lx(\mathrm{ND}[\ell])$ do
323     Head $\leftarrow F_E(\ell, N \,\|\, 0^\rho \,\|\, B)$
324     LNA[Head] $\xleftarrow{\cup} \{(\ell, N, A)\}$
procedure $\mathsf{G}_3$.ASSO($A, \ell$)
325 $B \leftarrow H(A)$; $\mathrm{A}[\ell][B] \xleftarrow{\cup} \{A\}$
326 for $N \in Lx(\mathrm{ND}[\ell])$ do
323   Head $\leftarrow F_E(\ell, N \,\|\, 0^\rho \,\|\, B)$
324   LNA[Head] $\xleftarrow{\cup} \{(\ell, N, A)\}$

procedure $\mathsf{G}_3$.DISA($A$)
330 $B \leftarrow H(A)$; $\mathrm{AD}[B] \xleftarrow{\setminus} \{A\}$
331 for $S \in$ LNA.values do
332   $S \xleftarrow{\setminus} \mathcal{L} \times \mathcal{N} \times \{A\}$
procedure $\mathsf{G}_3$.DISA($A, \ell$)
333 $B \leftarrow H(A)$; $\mathrm{A}[\ell][B] \xleftarrow{\setminus} \{A\}$
334 for $S \in$ LNA.values do
335   $S \xleftarrow{\setminus} \{\ell\} \times \mathcal{N} \times \{A\}$

procedure $\mathsf{G}_3$.DEC($C$)    Phase-1
350 Head $\|$ Body $\leftarrow C$ where $|\text{Head}| = n$
351 for $(\ell, N, A) \in$ LNA[Head] do
352   $M \leftarrow F_{\mathcal{E}}^{-1}((\ell, N, A), \text{Body})$
353   if $(\ell, N, A, M) \in \mathrm{H}[C]$ then
354     goto 35F

355 for $\ell \in \mathrm{L}$ do    P-2, same as $\mathsf{G}_4$'s
356   $N \,\|\, R \,\|\, B \leftarrow F_E^{-1}(\ell, \text{Head})$
357     where $|N| = \eta$ and $|R| = r$
358   if $R \neq 0^r$ or $N \notin Nx(\mathrm{ND}[\ell])$
359     then continue
35A   for $A \in \mathrm{A}[\ell][B] \cup \mathrm{AD}[B]$ do
35B     $M \leftarrow F_{\mathcal{E}}^{-1}((\ell, N, A), \text{Body})$
35C     if $(\ell, N, A, M) \in \mathrm{H}[C]$ then
35D       goto 35F
35E ret $\perp$

35F Old $\leftarrow Lx(\mathrm{ND}[\ell])$    Phase-3
35G $\mathrm{ND}[\ell] \xleftarrow{\|} N$
35H if $|\mathrm{ND}[\ell]| > d$ then
35I   $\mathrm{ND}[\ell] \leftarrow \text{tail}(\mathrm{ND}[\ell])$
35J New $\leftarrow Lx(\mathrm{ND}[\ell])$
35K for $N' \in$ Old $\setminus$ New do
35L   for $S \in$ LNA.values do
35M     $S \xleftarrow{\setminus} \{\ell\} \times \{N'\} \times \mathcal{A}$
35N for $N' \in$ New $\setminus$ Old do
35O   for $B \in \mathrm{A}[\ell]$.keys $\cup$ AD.keys do
35P     Head $\leftarrow F_E(\ell, N' \,\|\, 0^\rho \,\|\, B)$
35Q     for $A' \in \mathrm{A}[\ell][B] \cup \mathrm{AD}[B]$ do
35R       LNA.[Head] $\xleftarrow{\cup} \{(\ell, N', A')\}$
35S ret $(\ell, N, A, M)$
```

FIGURE 3.8. **Hybrid resembling NonceWrap.** Game G_3 executes procedures similar to those of NonceWrap. For decryption to succeed, it follows the ideal game. The encryption oracle is omitted as it is the same as G_5's, which is in Fig. 3.6.

injections. In regards to just the header, the probability that any two headers is the same is $1/2^n$. After q_e encryption queries, any of those pairs of queries can have such a collision. There are about $q_e^2/2$ ways to choose such a pair. Applying the same logic to the ciphertext body, $\mathcal{A}$ gets a collision in both header and body and distinguishes the two hybrids with probability

$$\mathbf{Pr}[\ \mathcal{A}^{\mathsf{G}_4}\ ] - \mathbf{Pr}[\ \mathcal{A}^{\mathsf{G}_3}\ ] \leq \frac{q_e^2}{2^{n+1}} \cdot \frac{q_e^2}{2^{\tau+1}} = \frac{q_e^4}{2^{n+\tau+2}}$$

Observe that G_3 executes almost exactly the same as G_2, which is the real game with ideal primitives does. The only differences in code are the checks for successful decryption. On lines 353 and 35C for G_3, we verify that the tuple was actually used in encryption. On the other hand, in G_2, we move to phase-3 if $M \neq \perp$. This difference can result in the two returning different values. More precisely, if queried with a ciphertext C that was not the result of an encryption query, G_2 may return a tuple while G_3 would never return a tuple. The probability this occurs is upper-bounded by the probability that the function $F_{\mathcal{E}}^{-1}$ on query (T, Y) returns a non-$\perp$ value given that Y was not an output of $F_{\mathcal{E}}$. This is the probability that line 91B in the top half of Fig. 3.5 returns. That is, the advantage $\mathcal{A}$ has in distinguishing G_3 and G_2 is

$$\mathbf{Pr}[\ \mathcal{A}^{\mathsf{G}_3}\ ] - \mathbf{Pr}[\ \mathcal{A}^{\mathsf{G}_2}\ ] \leq \frac{q_d^2}{2^{\sigma+1}}$$

Summing up all of the bounds computed over the hybrid argument, we get the bound in the theorem statement.

□

3.4. Anonymous AE Prototype Implementations

A prototype implementation of NonceWrap from the author is available at [**48**]. The code is written in Python (2.7.14 [**52**]) as this was the version compatible with the Python Cryptography (2.3.1) at the time [**51**].

Along with NonceWrap, there are several other anAE schemes in the prototype. These are the schemes aAENaive, aAEBase, and aAEDict. They are primarily historical artifacts of the development and design of NonceWrap.

The scheme aAENaive takes the most naïve and brute force approach to decrypting an anonymous ciphertext. It simply iterates over all active keys, each of the nonces within their respective nonce policies, and all the ADs that are registered with a given key, and trial decrypts until it finds some input that successfully decrypts. It is not meant to be a good scheme, but is present for benchmarking purposes.

The schemes aAEBase and aAEDict are the phases of NonceWrap decryption broken up into individual schemes. The scheme aAEBase's decryption is phase-2 of NonceWrap decryption—the phase where decryption iterates over its keys and deciphers the header for candidate nonces and AD hash values before trial decrypting. The scheme aAEDict's decryption consists of phase-1 and phase-3 of NonceWrap decryption—the phase that uses a dictionary and anticipated nonces in order to quickly identify the inputs to use for decryption. There is no fallback phase-2 for this scheme and as such it must be a sharp scheme. That is, what is anticipated is what is in policy for the nonces of aAEDict. It should be noted that aAEDict has two headers, one for the nonce and one for the AD hash. NonceWrap is the product of merging the two headers and combining the schemes aAEBase and aAEDict.

3.5. Anonymous AE Remarks

Complexity. While we don't find the anAE definition excessively complex, NonceWrap decryption is quite complicated. One complicating factor is the rich support we have provided for AD values—despite our expectation that implementations will assume the 1AD/Session restriction. Yet we have found that building in the 1AD/Session restriction would only simplify matters modestly. It didn't seem worth it.

We suspect that, no matter what, decryption in anonymous-AE schemes is going to be complicated compared to decryption under conventional nAE. The privacy principle demands that ciphertexts contain everything the receiver needs to decrypt, yet no adversarially worthwhile metadata. The decrypting party must infer this metadata, and it should do so quite efficiently.

Timing side-channels. Our anAE definition does not address timing side-channels, and it is worth noting NonceWrap raises several concerns with leaking identity information through decryption times. Timing information might leak how many sessions a header can belong to. In phase-2, nAE decryption is likely to be the operation that takes the longest, and it is possible that an observer might learn information on the number of sessions that produced a valid-looking header. Then there is the timing side-channel that arises from the usage of Lx and Nx. Phase-1 only works on headers in Lx, and is expected to be faster than phase-2, leaking information about whether a nonce was anticipated. We leave the modeling, analysis, and elimination of timing side-channels as an open problem.

THE USAGE PUZZLE. There is an apparent paradox in the use of anonymous AE. If used in an application-layer protocol over something like TCP/IP, then anonymous AE would seem irrelevant because communicated packets already reveal identity. But if used over an anonymity layer like Tor [**28**], then use of that service would seem to obviate the need for privacy protection. It would seem as though anonymous AE is pointless if the transport provides anonymity, and that pointless if the transport does not provide anonymity.

This reasoning is specious. First, an anonymity layer like Tor only protects a packet while it traverses the Tor network; once it leaves an exit node, the Tor-associated encryption is gone, and end-to-end privacy may still be desired. Second, it simply is not the case that every low-level transport completely leaks identity. For example, while a UDP packet includes a source port, the field need not be used.

To give a concrete example for potential use, consider how NonceWrap (and anAE in general) might fit in with DTLS 1.3 over UDP [**55**]. Unlike TLS, where session information is presumptively gathered from the underlying transport, DTLS transmits with each record an explicit (sometimes partially explicit) epoch and sequence number (SN). Since UDP itself does not use SNs, the explicit SNs of DTLS are used for replay protection. While DTLS has a mechanism for SN encryption in its latest draft, NonceWrap would seem to improve upon it. The way DTLS associates a key with encrypted records is through the sender's IP and port number at the UDP level. Using NonceWrap, these identifiers could be omitted. If the receiver needs to know source IP and port in order to reply, those values can be moved to the encrypted payload.

Further features of DTLS over UDP might be facilitated by NonceWrap. It provides a mechanism in which an invalid record can often be quickly identified, a feature useful in DTLS. In DTLS, when an SN greater than the next expected one is received, there is an option to either discard the message or keep it in a queue for later. This aligns with NonceWrap's formulation of Lx and Nx.

It is rarely straightforward to deploy encryption in an efficient, privacy-preserving way, and anAE is no panacea. But who's to say how privacy protocols might evolve if one of our most basic tools, AE, is re-envisioned as something more privacy friendly?

CHAPTER 4

Committing Authenticated Encryption

This chapter presents *committing authenticated encryption* (cAE), the other major primitive and contribution of this dissertation. This chapter examines definitions of previous committing AE works and highlights their subtle differences. It presents its own definitional framework for cAE, one that requires commitment to *all* encryption inputs in contrast to previous cAE definitions. Furthermore, the *fully committing* CTX construction is presented in this chapter along with security analysis. Lastly, this chapter gives new attacks on the committing security of AES-GCM and OCB.

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 [**31**] (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 [**29**, **36**]. 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 [**4**, **29**, **40**].

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 [**4**, **31**] 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 (see Appendix 4.5). 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 $\mathtt{t}$. 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 $\mathtt{0}$), meaning they are generated uniformly at random and remain unknown to the adversary; *revealed* (represented by a $\mathtt{1}$), meaning they were honestly generated, but the adversary knows their value; or *corrupted* (an $\mathtt{X}$), 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 $\mathtt{0X}$-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 [**29, 36**]. It is mentioned by those same authors that knowledge of the keys is sufficient. Under our terminology, this would be a $\mathrm{CAE}_{\mathtt{XX}}$-attack and a $\mathrm{CAE}_{\mathtt{11}}$-attack respectively.

We present a new attack on GCM for a weaker adversary, a $\mathrm{CAE}_{\mathtt{01}}$-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 $\mathrm{CAE}_{\mathtt{01}}$-attack on OCB as well. These attacks demonstrate that nAE-security is insufficient for even $\mathrm{CAE}_{\mathtt{01}}$-security.

4.1. Committing AE Definition

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 standard nAE from Chapter 2, those elements would be the key, nonce, AD, and message. The CAE game that captures this property is presented in Fig. 4.1.

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 $\mathsf{S} \overset{\cup}{\leftarrow} \{x\}$ as shorthand for $\mathsf{S} \overset{\cup}{\leftarrow} \mathsf{S} \cup \{x\}$, 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 $\mathsf{K_r}$ and $\mathsf{K_c}$ 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

```
$\mathrm{CAE}_{\Pi,\mathtt{t}}$

procedure Initialize()
for $i \in \mathbb{N}$ do $K_i \twoheadleftarrow \mathcal{K}$; $\boldsymbol{N}_i \leftarrow \emptyset$
$\mathsf{S}, \mathsf{K_c}, \mathsf{K_r} \leftarrow \emptyset$

procedure Finalize()
ret $\exists (K_i, N, A, M, C), (K_j, N', A', M', C') \in \mathsf{S}$ s.t.
    $(M \neq \bot \wedge M' \neq \bot) \wedge$
    $(C = C') \wedge$
    $(K_i, N, A, M) \neq (K_j, N', A', M') \wedge$
    $(\mathrm{chk}(K_i, K_j) \vee \mathrm{chk}(K_j, K_i))$

chk$(K_i, K_j)$
if $\mathtt{t} = \mathtt{00} \wedge K_i \notin \mathsf{K_c} \cup \mathsf{K_r} \wedge K_j \notin \mathsf{K_c} \cup \mathsf{K_r}$ then ret $1$
if $\mathtt{t} = \mathtt{01} \wedge K_i \notin \mathsf{K_r} \cup \mathsf{K_c} \wedge K_j \notin \mathsf{K_c}$ then ret $1$
if $\mathtt{t} = \mathtt{0X} \wedge K_i \notin \mathsf{K_r} \cup \mathsf{K_c}$ then ret $1$
if $\mathtt{t} = \mathtt{11} \wedge K_i \notin \mathsf{K_c} \wedge K_j \notin \mathsf{K_c}$ then ret $1$
if $\mathtt{t} = \mathtt{1X} \wedge K_i \notin \mathsf{K_c}$ then ret $1$
if $\mathtt{t} = \mathtt{XX}$ then ret $1$
ret $0$

procedure ENC$(i, N, A, M)$
if $K_i \notin \mathsf{K_r} \cup \mathsf{K_c} \wedge N \in \boldsymbol{N}_i$
    then ret $\bot$
$C \leftarrow \Pi.\mathcal{E}(K_i, N, A, M)$
$\mathsf{S} \overset{\cup}{\leftarrow} \{(K_i, N, A, M, C)\}$
ret $C$

procedure DEC$(i, N, A, C)$
$M \leftarrow \Pi.\mathcal{D}(K_i, N, A, C)$
$\mathsf{S} \overset{\cup}{\leftarrow} \{(K_i, N, A, M, C)\}$
ret $M$

procedure REV$(i)$
$\mathsf{K_r} \overset{\cup}{\leftarrow} \{K_i\}$; ret $K_i$

procedure COR$(i, K)$
$K_i \leftarrow K$; $\mathsf{K_c} \overset{\cup}{\leftarrow} \{K_i\}$
```

FIGURE 4.1. **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 $\mathtt{t}$ of the game. This function places restrictions on the keys that the adversary may win with.

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, $\mathtt{t}$, 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 $\mathsf{K_c}$. A key is revealed when the adversary learns of it through the reveal oracle REV and thus part of the set $\mathsf{K_r}$. 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 $\mathtt{X}, \mathtt{1}$, and $\mathtt{0}$ bits respectively. Six different types of collisions arise from these types of keys. The parameter $\mathtt{t}$ 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 $\mathtt{t}$ is quantified as $\mathbf{Adv}^{\mathrm{cae}}_{\Pi,\mathtt{t}}(\mathcal{A}) = \Pr[\mathrm{CAE}^{\mathcal{A}}_{\Pi,\mathtt{t}} \to 1]$. When discussing CAE-security with a specific type of collision, we denote the collision with a subscript i.e. $\mathrm{CAE}_{\mathtt{XX}}$-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 $\mathtt{t} = \mathtt{XX}$) collision [**4**, **29**, **36**]. 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 $\mathtt{01}$-collision) [**31**]. Bellare gives another robustness notion for randomized symmetric encryption called *random-key robustness* in [**9**] that is comparable to the $\mathrm{CAE}_{\mathtt{00}}$ 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.

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 [**4**]. 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 Chapter 4.2, 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 [**12**]. 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 in Appendix 4.5. 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 Chapter 4.2 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 [**29**]. This shows that nae-secure schemes do not achieve $\mathrm{CAE}_{\mathsf{XX}}$-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 $\mathrm{CAE}_{\mathsf{11}}$-security either.

We show that nAE schemes that are nae-secure in the multi-key sense—nae$*$-secure, are already $\mathrm{CAE}_{\mathsf{00}}$-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 4.1.1. *Any authenticated encryption scheme* Π *that is* nae$*$*-secure is also* $\mathrm{CAE}_{\mathsf{00}}$*-secure. That is, for any adversary* $\mathcal{A}$ *attacking the* $\mathrm{CAE}_{\mathsf{00}}$*-security of* Π*, there exists an adversary* $\mathcal{B}$ *against the* nae$*$*-security of* Π *such that*

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

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 $\mathrm{CAE}_{\mathsf{00}}$-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. 4.1, 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 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 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, $\Pr[E_1] \leq \mathbf{Adv}^{\mathrm{nae}}_{\mathcal{B}}$.

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: $\Pr[E_2] \leq \mathbf{Adv}^{\mathrm{nae}}_{\mathcal{B}}$.

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 $\Pr[E_3] \leq \mathbf{Adv}^{\mathrm{nae}}_{\mathcal{B}} + \frac{q_e^2}{2^{\tau+1}}$.

□

4.2. The CTX Construction

Collision-resistant hash functions. We briefly recall the definition for collision-resistant (CR) hash functions as it will be useful in analyzing our CTX construction. A *hash function* $H \colon \mathcal{D} \to \{0,1\}^h$ 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 $\textsf{true}$ if $H(M) = H(M')$ and $M \neq M'$. The adversary $\mathcal{A}$'s advantage against H is then quantified as $\mathbf{Adv}^{\text{col}}_{H}(\mathcal{A}) = \Pr[\text{CR}^{\mathcal{A}}_{H} \Rightarrow \textsf{true}]$. This definition of collision-resistance of unkeyed hash functions follows the human-ignorance approach of [**60**].

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 $\text{CAE}_{\texttt{XX}}$ security. Our $\textsf{CTX}$ construction is fully committing, efficient, and strong.

Let $\Pi = (\texttt{Enc}, \texttt{Dec})$ 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 $\texttt{Enc}$ 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 $\text{CAE}_{\texttt{XX}}$-secure cAE scheme, $\textsf{CTX}[\Pi, H]$. $\textsf{CTX}$'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 $\textsf{CTX}$ captures the scheme's ciphertext structure, which is a ciphertext core followed by a modified tag. The 'X' in the name suggests the scheme's $\texttt{XX}$-security level. The scheme is presented in Fig. 4.2.

We claim that $\textsf{CTX}$ is $\text{CAE}_{\texttt{XX}}$-secure as long as H is collision-resistant.

Theorem 4.2.1. *Let* $\Pi = (\texttt{Enc}, \texttt{Dec})$ *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* $\textsf{CTX}[\Pi, H]$ *be an nAE scheme constructed from* Π *and* H *as described in Fig. 4.2. Then, for any adversary* $\mathcal{A}$ *attacking the* $\text{CAE}_{\texttt{XX}}$*-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*

```
$\mathsf{CTX.Enc}(K, N, A, M)$
$C \leftarrow \Pi.\mathcal{E}_1(K, N, A, M)$
$T \leftarrow \Pi.\mathcal{E}_2(K, N, A, M)$
$T^* \leftarrow H(K, N, A, T)$
ret $C \parallel T^*$

$\mathsf{CTX.Dec}(K, N, A, \mathcal{C})$
$C \parallel T \leftarrow \mathcal{C}$
$M \leftarrow \Pi.\mathcal{D}_1(K, N, A, C)$
$T' \leftarrow \Pi.\mathcal{E}_2(K, N, A, M)$
if $T \neq H(K, N, A, T')$ then ret $\perp$
ret $M$
```

FIGURE 4.2. A CAE$_{\texttt{XX}}$-secure cAE scheme $\mathsf{CTX}[\Pi, H]$ 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.

$$\mathbf{Adv}^{\text{cae-}\texttt{XX}}_{\mathsf{CTX}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{col}}_{H}(\mathcal{B}).$$

PROOF. We construct adversary $\mathcal{B}$ to find a winning collision for H as follows. Adversary $\mathcal{B}$ sets up the CAE game and runs $\mathcal{A}$, answering its queries appropriately. When $\mathcal{A}$ terminates, it will have produced a pair of winning tuples for the CAE game: $(K_i, N, A, M, C \parallel T^*), (K_j, N', A', M', C \parallel T^*)$. Then, $\mathcal{B}$ can compute $T = \Pi.\mathcal{E}_2(K_i, N, A, M)$ and $T' = \Pi.\mathcal{E}_2(K_j, N', A', M')$ to produce authentication tags for the winning tuples. Furthermore, it must be the case that $T^* = H(K_i, N, A, T) = H(K_j, N', A', T')$ as that is how $C \parallel T^*$ of the winning tuples was produced in the first place.

So for $(K_i, N, A, T), (K_j, N', A', T')$ to be a winning collision for $\mathcal{B}$, the two tuples must not be equivalent. Suppose for contradiction that they are equivalent. Then for $\mathcal{A}$'s tuples to have won the CAE game it must be the case that $M \neq M'$. But this is impossible as it would violate the bijectivity of $\mathcal{E}_1$. Since core C is fixed and $(K_i, N, A) = (K_j, N', A')$ are fixed from the collision, there exists only one M'' such that $\mathcal{E}_1(K_i, N, A, M'') = C = \mathcal{E}_1(K_j, N', A', M'')$. Thus $M \neq M'$ is a contradiction and $(K_i, N, A, T) \neq (K_j, N', A', T')$ follows. In conclusion, the winning collision for $\mathcal{B}$ is $H(K_i, N, A, T) = H(K_j, N', A', T') = T^*$. □

It remains to show that CTX remains nAE-secure after its transform. We do so in the random oracle model (ROM), denoting CTX as $\mathsf{CTX}[\Pi]$ (as opposed to $\mathsf{CTX}[\Pi, H]$) when it is in the ROM. The separate privacy and authenticity notions used in Theorem 4.2.2 were recalled in Chapter 2 and can be found in [**61**].

THEOREM 4.2.2. *Let $\Pi = (\mathcal{E}, \mathcal{D})$ be a tag-based nAE scheme with an expansion of τ. Let $\mathsf{CTX}[\Pi]$ be the scheme described in Fig. 4.2. 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*

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

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

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

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 $\frac{q_{\mathrm{H}}^2}{2^{\delta+\tau}}$ can be made small by choice of δ, so it will not result in much loss.

PROOF. (1) We begin with the privacy part of the theorem. Let G_0 and G_1 be the games presented in Fig. 4.3. Note G_0 uses the boxed code whereas G_1 does not. Let G_2 be the game presented in Fig. 4.4. We claim that the advantage of adversary $\mathcal{A}_1$ is

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{CTX}[\Pi]}(\mathcal{A}_1) = \Pr[G_0(\mathcal{A}_1)] - \Pr[G_2(\mathcal{A}_2)] \tag{4.1}$$

where $\Pr[G(\mathcal{A})]$ denotes the probability that running adversary $\mathcal{A}$ in game G results in the Finalize procedure of G returning true.

One can observe that G_0 is exactly the real privacy game of nAE security using CTX as the scheme. There are two tables keeping track of random oracle entries, HT and ET. The former records new random oracle (RO) entries generated by direct calls to H and the latter by calls to

```
Games $\boxed{G_0}/G_1$

procedure Initialize()
  $K \twoheadleftarrow \mathcal{K}$

procedure ENC$(N, A, M)$
  $C \parallel T \leftarrow \Pi.\mathtt{Enc}_K(N, A, M)$
  $T^* \twoheadleftarrow \{0,1\}^\mu$
  if $\mathrm{HT}[K, N, A, T] \neq \bot$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{T^* \leftarrow \mathrm{HT}[K, N, A, T]}$
  $\mathrm{ET}[K, N, A, T] \leftarrow T^*$
  ret $C \parallel T^*$

procedure H$(L, N, A, T)$
  if $\mathrm{HT}[L, N, A, T] \neq \bot$ then ret $\mathrm{HT}[L, N, A, T]$
  $\mathrm{HT}[L, N, A, T] \twoheadleftarrow \{0,1\}^\mu$
  if $\mathrm{ET}[L, N, A, T] \neq \bot$ then
     $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{\mathrm{HT}[L, N, A, T] \leftarrow \mathrm{ET}[L, N, A, T]}$
  ret $\mathrm{HT}[L, N, A, T]$

procedure Finalize$(d)$
  ret $d$
```

FIGURE 4.3. Games G_0 and G_1 for the privacy proof of Theorem 4.2.2. The two games are identical except G_0 contains the boxed code and G_1 does not.

```
Games $G_2$

procedure ENC$(N, A, M)$
  $C \twoheadleftarrow \{0,1\}^{|M|}$; $T^* \twoheadleftarrow \{0,1\}^\mu$; ret $C \parallel T^*$

procedure H$(L, N, A, T)$
  if $\mathrm{HT}[L, N, A, T] = \bot$ then $\mathrm{HT}[L, N, A, T] \twoheadleftarrow \{0,1\}^\mu$
  ret $\mathrm{HT}[L, N, A, T]$

procedure Finalize$(d)$
  ret $d$
```

FIGURE 4.4. Games G_2 for the proof of Theorem 4.2.2.

ENC. On the other hand, G_2 is the ideal privacy game for nAE security, coupled with access to a random oracle. Hence, Equation 4.1 follows by the definition of nAE privacy. Equation 4.1 is equivalent to the following:

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{CTX}[\Pi]}(\mathcal{A}_1) = \Pr[G_0(\mathcal{A}_1)] - \Pr[G_1(\mathcal{A}_1)] + \Pr[G_1(\mathcal{A}_1)] - \Pr[G_2(\mathcal{A}_1)] \tag{4.2}$$

From here, we can build adversary $\mathcal{B}_1$ such that

$$\Pr[G_1(\mathcal{A}_1)] - \Pr[G_2(\mathcal{A}_1)] \leq \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(\mathcal{B}_1). \tag{4.3}$$

Adversary $\mathcal{B}_1$ behaves as follows. First, it initializes a table HT that it will maintain during its execution. Then, it runs adversary $\mathcal{A}_1$. When $\mathcal{A}_1$ makes a query of:

- ENC(N, A, M) - Adversary $\mathcal{B}_1$ calls its own encryption oracle with the same N, A, M, getting back $C \parallel T$ as a response. It then samples $T^* \twoheadleftarrow \{0,1\}^\mu$ and returns $C \parallel T^*$ to $\mathcal{A}_1$.
- H(L, N, A, T) - Adversary $\mathcal{B}_1$ checks to see if there is an entry in HT$[L, N, A, T]$. If not, then it samples a string uniformly at random from $\{0,1\}^\mu$ and records it at HT$[L, N, A, T]$. It always responds to $\mathcal{A}_1$ with HT$[L, N, A, T]$.

When $\mathcal{A}_1$ terminates and outputs a bit, $\mathcal{B}_1$ outputs the same bit. When $\mathcal{B}_1$'s encryption oracle is fake, then it perfectly simulates G_2 as the ciphertext bodies C it returns to $\mathcal{A}_1$ will be uniformly random strings. When $\mathcal{B}_1$'s encryption oracle is real, then it perfectly simulates G_1 as C would be generated through encryption with Π under some hidden key (which is unknown to $\mathcal{B}_1$). This gives the advantage term of Equation 4.3.

Games G_0 and G_1 are identical-until-bad. By the Fundamental Lemma of Game Playing [**17**], we have that the advantage of an adversary distinguishing these two games is at most the probability of bad being set. That is, we have

$$\Pr[G_0(\mathcal{A}_1)] - \Pr[G_1(\mathcal{A}_1)] \leq \Pr[G_1(\mathcal{A}_1) \textit{ sets } \mathsf{bad}]. \tag{4.4}$$

In the games, bad gets set to true on lines 12 and 22 in Fig. 4.3 when the tables HT and ET are checked for an entry. Recall that the table ET records mappings of K, N, A, T quadruples to random T^* that are generated during encryption. The key K here is fixed to the one sampled at game initialization. The table HT, on the other hand, tracks mappings from L, N, A, T quadruples to random T^* generated during random oracle queries. The key L here is part of the adversary's query. The flag bad gets set to true if one oracle (either encryption or the random oracle) finds an entry already recorded in the other's table when trying to generate the random T^*. For example, suppose the adversary makes a query (K, N, A, T) to H that results in some T^*. If the adversary later queries encryption with N, A, M such that $C \parallel T$ is the result of Π's encryption, then the T^* returned to the adversary needs to be the T^* in the random oracle query. This is covered by the table HT. The other table ET covers the other direction– when a later RO query needs a tag from a previous encryption query.

Observe that for bad to be set true, the adversary will need to make a query to H with K, the secret key, as the argument L. It has to in order to satisfy the conditions of either 12 or 22 . Following this, game G_3 in Fig. 4.5 is set up in a way such that an adversary wins if it queries the random oracle with the secret key. It runs its encryption oracle like G_1. Hence, we have that

$$\Pr[G_1(\mathcal{A}_1) \text{ sets } \mathsf{bad}] \leq \Pr[G_3(\mathcal{A}_1)]. \tag{4.5}$$

Now, we build adversary $\mathcal{B}_2$ such that

$$\Pr[G_3(\mathcal{A}_1)] \leq \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(\mathcal{B}_2) + \frac{q_H}{2^{\delta+\tau}} \tag{4.6}$$

Adversary $\mathcal{B}_2$ initializes with a table HT and a set S. It runs $\mathcal{A}_1$ and responds to encryption queries just like adversary $\mathcal{B}_1$. For queries $\mathrm{H}(L, N, A, T)$, $\mathcal{B}_2$ answers like $\mathcal{B}_1$ except it also adds L to its set S.

When $\mathcal{A}_1$ terminates, $\mathcal{B}_2$ ignores its output. It then picks a nonce N^* that was not used by $\mathcal{A}_1$ in any of its ENC queries. It picks a message $M^* \twoheadleftarrow \{0,1\}^{\delta}$. Recall that $\delta \geq 0$ is the adjustable parameter from the theorem statement. Then it queries its own ENC with N^*, ε, M^* and gets back

```
Games $G_3$

procedure Initialize()
    $K \twoheadleftarrow \mathcal{K}$; $\mathsf{S} \leftarrow \emptyset$

procedure ENC($N, A, M$)
    $C \parallel T \leftarrow \Pi.\mathtt{Enc}_K(N, A, M)$; $T^* \twoheadleftarrow \{0,1\}^\mu$; ret $C \parallel T^*$

procedure H($L, N, A, T$)
    if $\mathrm{HT}[L, N, A, T] = \bot$ then $\mathrm{HT}[L, N, A, T] \twoheadleftarrow \{0,1\}^\mu$
    $\mathsf{S} \leftarrow \mathsf{S} \cup \{L\}$; ret $\mathrm{HT}[L, N, A, T]$

procedure Finalize($d$)
    ret $(K \in \mathsf{S})$
```

FIGURE 4.5. Games G_3 for the proof of Theorem 4.2.2.

a response C^* of the length $\delta + \tau$. Next, it sets a flag $b \leftarrow 0$ before executing a loop, iterating over every key $L \in \mathsf{S}$. Each iteration, it checks whether $\Pi.\mathtt{Enc}_L(N^*, \varepsilon, M^*)$ outputs C^*, setting $b \leftarrow 1$ if one does. Finally, $\mathcal{B}_2$ outputs b as its output.

Let E_1 be the event that $\mathcal{B}_2$ outputs 1 in its real game and E_0 be the event it does so in its ideal game. Then, we have that $\Pr[E_1] \geq \Pr[G_3(\mathcal{A}_1)]$ because if the key K (the key to $\mathcal{B}_2$'s real game) is in the set S, then b is set to 1 when $L = K$ during $\mathcal{B}_2$'s loop. For E_0, we have that $\Pr[E_0] \leq q_H/2^{\delta+\tau}$. Since S is fixed independently of the random C^* returned by the ideal encryption oracle, each iteration of $\mathcal{B}_2$'s loop sets b to 1 is at most $2^{-\delta+\tau}$. Applying the union bound across iterations, we get the bound for $\Pr[E_0]$. Finally, we get

$$\mathbf{Adv}_{\Pi}^{\mathrm{priv}}(B_2) = \Pr[E_1] - \Pr[E_0] \geq \Pr[G_3(\mathcal{A}_1] - \frac{q_H}{2^{\delta+\tau}}$$

which gives Equation 4.6. Combining Equations 4.1, 4.3, and 4.6 yields the stated privacy bound in the theorem.

(2) We now proceed with the authenticity part of Theorem 4.2.2. Let $\mathcal{A}_2$ be the adversary attacking the authenticity of $\mathsf{CTX}[\Pi]$. We can construct an adversary $\mathcal{B}_3$ attacking the authenticity of Π such that

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathsf{CTX}[\Pi]}(\mathcal{A}_2) \leq \mathbf{Adv}^{\mathrm{auth}}_{\Pi}(\mathcal{B}_3) + \frac{1}{2^{\mu}}. \tag{4.7}$$

We define $\mathcal{B}_3$ as follows. Adversary $\mathcal{B}_3$ responds to $\mathcal{A}_2$'s ENC and H queries the same way that $\mathcal{B}_1$ does. When $\mathcal{A}_2$ terminates with a forgery $(N_2, A_2, C_2 \parallel T_2)$, adversary $\mathcal{B}_3$ then picks a message M^* of length δ, and a nonce N^* that has not been used by $\mathcal{A}_2$ in its ENC queries and is not the nonce in $\mathcal{A}_2$'s forgery ($N^* \neq N_2$). It then makes a query to its own encryption oracle ENC with N^*, ε, M^*, getting back a response C^*.

Now $\mathcal{B}_3$ selects particular entries in its table HT. We write these entries as quintuples (L, N, A, T, T^*) to succinctly denote the mapping of (L, N, A, T) to T^*. Specifically, $\mathcal{B}_3$ picks entries (L, N, A, T, T^*) such that $N = N_2$, $A = A_2$, and $T^* = T_2$. It then iterates through all such entries and tests whether $\Pi.\mathcal{E}_L(N, A, M^*) = C^*$. Let S be the set of entries that satisfy this condition. Adversary $\mathcal{B}_3$ then executes the following:

```
**for** $(L, N, A, T, T^*) \in \mathsf{S}$ **do**
    $M \leftarrow \Pi.\mathcal{D}_1(L, N, A, C_2)$
    $T' \leftarrow \Pi.\mathcal{E}_2(L, N, A, M)$
    **if** $T = T'$ **then ret** $(N, A, C_2 \parallel T)$
```

Recall that $\mathcal{E}_2$ and $\mathcal{D}_1$ are the algorithms of a tag-based nAE scheme that produce the authentication tag and decrypt the ciphertext core respectively. This loop checks which of the candidate entries contains a forgery $\mathcal{B}_3$ can use for C_2 by verifying a tag (for the scheme Π) for it. If the loop finishes without successfully verifying a tag, this means that $\mathcal{A}_2$ failed its own forgery, which would cause $\mathcal{B}_3$ to fail as well. When $\mathcal{A}_2$ succeeds in forging, then $\mathcal{B}_3$ can recover a tag T appropriate for its own forgery. This gives us the authenticity advantage term for $\mathcal{B}_3$ in the bound.

It is possible for $\mathcal{A}_2$ to forge without $\mathcal{B}_3$ being able to recover the tag it needs. Suppose $\mathcal{A}_2$'s forgery is $(N, A, C \parallel T^*)$. Then, for example, $\mathcal{A}_2$ could never ask H the appropriate query to fill in HT with T^*. However, all responses to H queries are independent of one another, so one response for H tells $\mathcal{A}_2$ nothing about another response for H. This is also true for the random tags generated by any ENC queries made by $\mathcal{A}_2$. So, without querying to H the exact (L, N, A, T) used by $\mathcal{B}_3$'s encryption oracle to make C, the best $\mathcal{A}_2$ can do to come up with a valid T^* for its forgery is by guessing. For a fixed key, nonce, AD, and ciphertext core, there is exactly one tag that verifies. Following this, the probability $\mathcal{A}_2$ creates a valid tag T^* and forges by guessing is $1/2^\mu$ as it has a single guess for its forgery over 2^μ possible tags. This gives us the last term in the bound. □

4.3. 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 [**29**, **36**]. 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 NAE$[G, H]$ built on a PRG G and a MAC H. The definition of NAE$[G, H]$ is given in Fig. 4.6. In our pseudocode, we write $S[0..n]$ to denote a substring of the bitstring S starting from the 0th bit to the nth 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 may also take in an additional argument

```
NAE.Enc(K, N, A, M)
$K_1 \parallel K_2 \leftarrow K$; $P \leftarrow G(K_1, N)$
$C \leftarrow M \oplus P[0..|M|-1]$
$T \leftarrow H(K_2, C \parallel A)$
ret $C \parallel T$

NAE.Dec(K, N, A, C)
$K_1 \parallel K_2 \leftarrow K$; $C \parallel T \leftarrow C$
if $H(K_2, C \parallel A) \neq T$ then ret $\perp$
$P \leftarrow G(K_1, N)$
$M \leftarrow C \oplus P[0..|M|-1]$; ret $M$
```

Figure 4.6. 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.

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.

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 [**44**, **45**].

GHASH works by computing a polynomial over the field 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 GF(2^{128})):

$$\text{GHASH}(K', A, C) = \left[\sum_{i=1}^{a} A_i \cdot P^{a+c+2-i}\right] + \left[\sum_{i=1}^{c} C_i \cdot P^{c+2-i}\right] + (\text{len}(A) \parallel \text{len}(C)) \cdot P \tag{4.8}$$

And the tag T is finalized as:

$$T = \text{GHASH}(K', A, C) \; \oplus \; E_{K'}(N \parallel 0^{31}1) \tag{4.9}$$

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:

$$A = \left[T \oplus E_{K'}(N \parallel 0^{31}1) + (\mathrm{len}(A) \parallel \mathrm{len}(C)) \cdot P + \left[\sum_{i=1}^{c} C_i \cdot P^{c+2-i} \right] \right] \cdot (P^{c+2})^{-1} \tag{4.10}$$

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 $\mathrm{ptarget}(K', T, C)$ as shown in Equation 4.10 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 Equation 4.10 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 $A' = \mathrm{ptarget}(K', T, A \parallel C)$ instead of $\mathrm{ptarget}(K', T, C)$. 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 [**39**].

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

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

FIGURE 4.7. 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. In the case that it is not, on lines 24 and 25, one would instead compute the offset Δ for the last block, apply the blockcipher on it, then xor the result with the last ciphertext block to recover the last message block. The attack written here shows how to compute a single associated data block to get a colliding ciphertext (line 26-29). But it should be noted that it is possible to mount the attack by choosing an arbitrary AD first and computing a single block that satisfies a necessary value for *auth* to get the colliding tag.

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 $auth = F \oplus T$ 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 $\mathrm{ENC}(j, N, A, M)$. The attack is described in code in Fig. 4.7.

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 $B = auth' \oplus F \oplus T$ 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 $A = A' \parallel B'$.

4.4. 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 4.1. The first to study committing encryption in the AE setting was Farshim, Orlandi, and Roşie (FOR17) [**31**]. 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.

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 $\text{CAE}_{\texttt{01}}$ notion, where the adversary must find a misattribution with a revealed and an honest key. One of our results is the existence of $\text{CAE}_{\texttt{01}}$-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 [**36**]. 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

Paper	AE Variant	Committing AE Definition
FOR17 [**31**]	Probabilistic AE, No AD support	*Full robustness* - $\mathcal{A}$ finds (K, K', C) s.t. decryption of C with both keys is successful.
GLR17 [**36**]	Probabilistic and deterministic AEAD	*Receiver-binding* - $\mathcal{A}$ 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 [**29**]	Probabilistic AEAD	*Strong Receiver-binding* - Same as receiver-binding except $(K, A, M) \neq (K', A', M')$.
ADGKLS20 [**4**]	Deterministic AEAD	*Key-commitment* - $\mathcal{A}$ finds $((K, N, A, M), (K', N, A', M'), C)$ through ENC, DEC queries s.t. $K \neq K'$ and $M, M' \neq \bot$.
BH22 [**12**]	Deterministic AEAD + misuse-resistant AE	CMT(D)$_{\mathrm{s}}$-ℓ *security* - $\mathcal{A}$ finds $(K_1, N_1, A_1, M_1), .., (K_s, N_s, A_s, M_s)$ such that what is committed from each tuple is distinct. The parameter ℓ specifies "what is committed."
Our work	Deterministic AEAD	CAE$_{\mathtt{t}}$-*security* - $\mathcal{A}$ finds $((K, N, A, M), (K', N', A', M'), C)$ through $\mathrm{ENC}, \mathrm{DEC}$ queries s.t. $(K, N, A, M) \neq (K', N', A', M')$ and $M, M' \neq \bot$. The parameter $\mathtt{t}$ specifies how $\mathcal{A}$ interacts with the keys.

TABLE 4.1. A comparison of the subtly different definitions in CAE literature.

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) [**29**] 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 *encryptment* 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 Schmieg (ADGKLS20) [**4**] 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 [**12**]. They attend to deterministic AEAD as well as misuse-resistant AE with encryption inputs K, N, A, M. Their committing security notion is $\mathrm{CMT(D)}_{\mathrm{s}}$-$\ell$ where s is an integer and $\ell \in \{1, 3, 4\}$. The presence or absence "D" denotes whether the notion is decryption or encryption-based– adversary 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 Chapter 4.1 does not allow for tweaking for commitments of sub-tuples of inputs, but the alternative framework given in Appendix 4.5 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. All in all, 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 4.2.

Recall that FOR17 are in the probabilistic AE setting without associated data. Their construction $\mathsf{EtM}[\mathcal{E}, H]$ creates a tag that provides authenticity while 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 $\mathsf{CEP}[G, F, F^{\mathrm{cr}}]$ 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 a primitive that simultaneously encrypts and commits a message and is one-time use. They give a concrete encryptment scheme HFC that uses a compression function and a padding scheme. They give a simple transform that builds a cAE scheme out of an encryptment scheme and a probabilistic nAE scheme. We note that HFC requires a pass over the message to apply encrypt and commit it.

The $\mathsf{CommitKey}$ scheme from ADGKLS20 comes in four flavors. We describe the variant $\mathsf{CommitKey}_{IV}$ 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."

Construction	Description
$\mathsf{EtM}[\mathcal{E}, H]$ [**31**]	$\mathcal{E}$ 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 $C \parallel T$.
$\mathsf{CEP}[G, F, F^{\mathrm{cr}}]$ [**36**]	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* [**29**]	HFC is an *encryptment* scheme built from a compression function and a padding scheme. DGRW18 show a simple transform that promotes an encryptment scheme into a cAE scheme.
$\mathsf{CommitKey}_{IV}$ [**4**]	$\mathcal{E}$ 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 $C \parallel K_c$.
$\mathsf{UtC}[\mathcal{E}, \mathsf{F}]$ [**12**]	$\mathcal{E}$ is nAE scheme. F is *committing* PRF. Get (P, L) from $\mathsf{F}(K, N)$. Get C from $\mathcal{E}(L, N, A, M)$. Output $P \parallel C$.
$\mathsf{HtE}[\mathcal{E}, H]$ [**12**]	$\mathcal{E}$ is a CMT-1 nAE scheme. H is a CR function. Get L from $H(K, (N, A))$. Output $\mathcal{E}(L, N, \varepsilon, M)$.

TABLE 4.2. A comparison of selected constructions targeting their respective cAE security goals. *Not a committing AE scheme, but closely related.

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 [**5**]) 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.

4.5. CAE with Misattribution Types

Modeling weaker misattributions. The CAE definition presented in 4.1 only attends to the strongest level of misattributions. Specifically, a ciphertext C experiences the strongest level of misattribution if an adversary finds two distinct explanations $(K, N, A, M) \neq (K', N', A', M')$ for it. We call schemes that are resistant against this type of misattribution *fully committing.*

However, one may be interested in schemes that protect against weaker misattributions. After all, other works have investigated these weaker notions [**4**,**29**,**31**,**36**] and contemporary work by [**12**] give a framework that supports committing to subsets of the encryption inputs.

In addressing this, we have an alternative definition given in Fig. 4.8. In this definition, the game is parametrized with an additional four-bit string $\mathbf{u}$. This string allows the specification of which elements of the (K, N, A, M) tuple one should commit to. The first bit corresponds to the key, the second the nonce, and so on. This framework is finer-grained than the contemporary framework of [**12**] as theirs only captures three kinds of commitments: committing to just the key;

```
$\mathrm{CAE}_{\Pi,\mathtt{t},\mathtt{u}}$

procedure Finalize()
ret $\exists(K_i, N, A, M, C), (K_j, N', A', M', C') \in \mathsf{S}$ s.t.
    $(M \neq \bot \wedge M' \neq \bot) \wedge$
    $(C = C') \wedge$
    $\mathrm{tup}((K_i, N, A, M), (K_j, N', A', M')) \wedge$
    $(\mathrm{chk}(K_i, K_j) \vee \mathrm{chk}(K_j, K_i))$

$\mathrm{chk}(K_i, K_j)$
if $\mathtt{t} = \mathtt{00} \wedge K_i \notin \mathsf{K_c} \cup \mathsf{K_r} \wedge K_j \notin \mathsf{K_c} \cup \mathsf{K_r}$ then ret $1$
if $\mathtt{t} = \mathtt{01} \wedge K_i \notin \mathsf{K_r} \cup \mathsf{K_c} \wedge K_j \notin \mathsf{K_c}$ then ret $1$
if $\mathtt{t} = \mathtt{0X} \wedge K_i \notin \mathsf{K_r} \cup \mathsf{K_c}$ then ret $1$
if $\mathtt{t} = \mathtt{11} \wedge K_i \notin \mathsf{K_c} \wedge K_j \notin \mathsf{K_c}$ then ret $1$
if $\mathtt{t} = \mathtt{1X} \wedge K_i \notin \mathsf{K_c}$ then ret $1$
if $\mathtt{t} = \mathtt{XX}$ then ret $1$
ret $0$

$\mathrm{tup}((K_i, N, A, M), (K_j, N', A', M'))$
$T_i, T_j \leftarrow (\bot, \bot, \bot, \bot)$
if $\mathtt{u}[0] = 1$ then $T_i[0] \leftarrow K_i$; $T_j[0] \leftarrow K_j$
if $\mathtt{u}[1] = 1$ then $T_i[1] \leftarrow N$; $T_j[1] \leftarrow N'$
if $\mathtt{u}[2] = 1$ then $T_i[2] \leftarrow A$; $T_j[2] \leftarrow A'$
if $\mathtt{u}[3] = 1$ then $T_i[3] \leftarrow M$; $T_j[3] \leftarrow M'$
ret $T_i \neq T_j$

procedure Initialize()
for $i \in \mathbb{N}$ do
    $K_i \twoheadleftarrow \mathcal{K}$; $\boldsymbol{N}_i \leftarrow \emptyset$
$\mathsf{S}, \mathsf{K_c}, \mathsf{K_r} \leftarrow \emptyset$

procedure $\mathrm{ENC}(i, N, A, M)$
if $K_i \notin \mathsf{K_r} \cup \mathsf{K_c} \wedge N \in \boldsymbol{N}_i$
    then ret $\bot$
$C \leftarrow \Pi.\mathcal{E}(K_i, N, A, M)$
$\mathsf{S} \overset{\cup}{\leftarrow} \{(K_i, N, A, M, C)\}$
ret $C$

procedure $\mathrm{DEC}(i, N, A, C)$
$M \leftarrow \Pi.\mathcal{D}(K_i, N, A, C)$
$\mathsf{S} \overset{\cup}{\leftarrow} \{(K_i, N, A, M, C)\}$
ret $M$

procedure $\mathrm{REV}(i)$
$\mathsf{K_r} \overset{\cup}{\leftarrow} \{K_i\}$; ret $K_i$

procedure $\mathrm{COR}(i, K)$
$K_i \leftarrow K$; $\mathsf{K_c} \overset{\cup}{\leftarrow} \{K_i\}$
```

FIGURE 4.8. **An alternative** CAE**-security game.** This definition of CAE-security parametrizes the game with a four-bit string $\mathtt{u}$ that dictates which elements—key, nonce, AD, message—should count as an adversarial win when misattributed.

committing to the key, nonce, and AD; and committing to everything. The framework here allows the expression of all sixteen ways one can commit to encryption inputs.

CHAPTER 5

Concluding Remarks

5.1. Summary

This work targets two misguided assumptions that users may have of nonce-based AE. In particular, one may assume from an nAE privacy definition that anonymity comes with nAE privacy. One may also assume from an nAE authenticity definition that decrypting a ciphertext with inputs other than the one used to create it would result in failure. Neither of these assumptions hold. In this dissertation, we give two primitives that actually satisfy these assumptions: anonymous AE and committing AE.

Anonymous AE. We give an anonymous nonce-based AE syntax that serves as an extension of an nAE scheme by adding an alternative decryption algorithm accompanied with four other state modification algorithms. This alternative decryption algorithm takes in only a ciphertext and is asked to recover the message, the operative nonce, the operative AD, and a pointer to the operative key. The result of this kind of interface is that the ciphertext given to decryption is understood as the *full* ciphertext, meaning everything that is required to decrypt. We then give a security definition for anAE that captures both privacy (along with anonymity) and authenticity.

We propose a construction NonceWrap for achieving anonymous nAE. Each NonceWrap-encrypted ciphertext consists of a header and a body. The body is the conventional nAE ciphertext. The header contains an enciphering of the nonce, some redundant bits used for rapid rejection, and a hash of an AD value. NonceWrap is able to utilize a number of dictionaries and header precomputation to quickly identify the operative key, nonce, and AD for a given ciphertext. Should NonceWrap not be able to identify these inputs through its dictionaries, it can fall back to trial decrypting. Even so, its trial decrypting is fast, as it first deciphers the headers and checks the presence of the redundant bits among other checks. This allows it to immediately reject candidate inputs without having to actually decrypt the body. We show that $\mathsf{NonceWrap}$ satisfies our security definition. A prototype implementation of $\mathsf{NonceWrap}$ as well as some of its predecessors are also described.

Committing AE. We give a committing AE security definition for nonce-based AE with associated data. The definition asks that an adversary find a colliding ciphertext. The definition is fully committing, meaning it requires all encryption inputs—the key, nonce, AD, and plaintext—to all be committed in the ciphertext. Furthermore, our definition considers adversarial relationships with the keys involved in constructing its colliding ciphertext. That is, either the key remains honest and hidden from the adversary, the key is known to the adversary, or the key is chosen by the adversary. Recall that we represent these as $\mathtt{0}$, $\mathtt{1}$, and $\mathtt{X}$ respectively. We show that conventional nAE implies $\mathrm{CAE}_{\mathtt{00}}$-security.

We then present our scheme CTX for achieving fully committing AE. It is a simple transform that uses a tag-based nAE scheme to generate a ciphertext C and its tag T. CTX then exploits the injectivity of nAE encryption to commit to the message without having to process it. A CTX ciphertext consists of $C \parallel T^*$ where T^* is the collision-resistant hash of K, N, A, T. We show that this provides $\mathrm{CAE}_{\mathtt{XX}}$ security in the standard model while showing that CTX has nAE privacy and authenticity in the random oracle model Lastly, we compare CTX and our cAE-security definition with other cAE-related definitions and constructions.

5.2. Future Work

Strengthening the anAE Definition. Although the anAE definition that we present in Chapter 3 captures a strong notion of privacy that encompasses anonymity, we discuss one of the ways to strengthen it further here. One issue with the anAE definition is that it assumes that all the keys in the anAE security game are chosen uniformly at random. Due to this assumption, the definition fails to capture security against adversaries who may learn the keys of some parties or register their own keys with the receiver.

An adversary may break the anAE security of an anAE scheme is by creating a misattribution. We show in Chapter 3.3 that, for $\mathsf{NonceWrap}$, this is difficult for any adversary to do assuming the blockcipher and nAE scheme it uses are prp$*$-secure and nae$*$-secure respectively. This is consistent with what we know about creating colliding ciphertexts, which is needed for an adversary to create a misattribution. For nAE-secure nAE schemes, it is difficult to find a ciphertext that decrypts validly under two different honestly chosen and hidden keys. We prove this in Chapter 4.1.

But if we adjust our anAE definition to account for potential adversarial keys, then misattributions may be possible in NonceWrap. Will such misattributions be meaningful in undermining anonymity? While we do not formalize it, one can imagine the addition of an oracle, say a key-installation oracle, to the anAE security games. This oracle in the real game adds an adversarial key input to the vector of active keys with some SID. The ideal one will ignore the adversarial key, but set up the bookkeeping necessary to keep track of the SID associated with it.

Suppose the adversary in this modified anAE game initializes some honest keys and installs some of its own malicious keys. Then suppose it is able to create a misattribution—it submits a ciphertext that decrypts validly under two of the keys. If the two keys that the ciphertext decrypts under are both adversarial keys, then we argue that this is not a meaningful misattribution. In attacking an anAE scheme, an adversary may want to disrupt service or spoof messages for the receiver. Spoofing messages under its own identity does not matter nor would disrupting service for itself. Hence, the modified anAE game would need a mechanism to identify misattributions that involve at least one honest key in order for the adversary to win. In the language of our cAE framework, XX-type collisions do not seem to be meaningful against anAE schemes.

However, we show in Chapter 4.3 that for nAE schemes AES-GCM and OCB, an adversary can create a colliding ciphertext under two keys—one that it has knowledge of and one honestly chosen and completely hidden from the adversary. This extends to an installed key as if the adversary has control over it, then it also has knowledge of it. Following this, an adversary may be able to create a winning misattribution for the modified anAE game against NonceWrap should NonceWrap be instantiated with AES-GCM or OCB. We suspect that this is possible, but have not investigated it further. We also suspect that one can mitigate such misattributions for NonceWrap by using an CAE_{XX}-secure scheme as its nAE component like CTX. We leave the exploration of a strengthened anAE definition, this potential weakness in NonceWrap, and the mitigation of this weakness for future work.

Regarding non-CAE$_{\mathsf{XX}}$ security. As demonstrated by the above modified anAE definition, there are contexts where CAE$_{\mathsf{00}}$-security is insufficient but stronger notions such as CAE$_{\mathsf{XX}}$ is not required.

Looking back at Theorem 4.2.1 from Chapter 4.2, we see that the CAE$_{\mathsf{XX}}$-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$_{\mathsf{XX}}$) 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$_{\mathsf{0X}}$-security of CTX using a birthday attack under standard assumptions on Π.

Let Π' be a tag-based nAE scheme that uses a PRG G to generate a one time pad for the plaintext M using the key K and the nonce N to get a ciphertext core C and a tagging function $\mathcal{E}_2(K, N, A, M)$ for an AD A to produce a tag T. It returns $C \parallel T$ as its final ciphertext. AES-GCM is an example of such a scheme. Now let Π be a scheme that runs exactly the same as Π' except if the key input is a special reserved key, say $K^* = 0^k$ where k is the key length of Π, then $\mathcal{E}_2$ outputs a tag $T = 0^t$ for all N, A, M where t is the tag length of Π.

We give an adversary $\mathcal{A}$ against $\mathsf{CTX}[\Pi, H]$ even if H is a collision-resistant hash function. Let μ be the output length of H and q be the number of encryption queries $\mathcal{A}$ makes. Adversary $\mathcal{A}$ fixes $N_1, \ldots, N_q$ distinct nonces as well as some message M and some AD A. It then makes q queries to its encryption oracle where the ith query is in the form $C_i \parallel T_i \leftarrow \mathrm{ENC}(N_i, A, M)$. Note that these queries are made under some hidden key K where K is the target honest key that $\mathcal{A}$ is trying to collide with.

Now $\mathcal{A}$ chooses a fresh nonce N distinct from its previous nonces and chooses distinct AD values $A_1, \ldots, A_q$ that are different from the previous A. It then computes a q candidate tags where the jth hash computation is in the form $U_j \leftarrow H(0^k, N, A_j, 0^t)$.

By the birthday paradox, with probability $\Omega(q^2/2^\mu)$, there are indices i, j such that $T_i = U_j$. Now $\mathcal{A}$ computes $M_j \leftarrow G(0^k, N) \oplus C_i$. We end up with the misattribution $\Pi.\mathcal{E}(0^k, N, A_j, M_j) = C_i \parallel T_i$ through the following equations:

$$\begin{aligned}\Pi.\mathcal{E}(0^k, N, A_j, M_j) &= G(0^k, N) \ \oplus \ M_j \parallel H(0^k, N, A_j, \mathcal{E}_2(0^k, N, A_j, M_j)) \\ &= G(0^k, N) \ \oplus \ G(0^k, N) \ \oplus \ C_i \parallel H(0^k, N, A_j, 0^t) \quad \text{// by def of } M_j \text{ and } \mathcal{E}_2 \\ &= C_i \parallel U_j = C_i \parallel T_i \quad \text{// by def of } U_j \text{ and birthday collision}\end{aligned}$$

So, CTX is unable to avoid a birthday bound even for $\mathrm{CAE}_{\mathtt{OX}}$-security. It is possible that it may be able to do so with additional assumptions on the nAE scheme it uses. We leave the investigation of such assumptions and the possibility of other cAE schemes with better $\mathrm{CAE}_{\mathtt{OX}}$-security to future work.

Beyond Anonymous AE and Committing AE. We sketch another variant of AE that enhances usability beyond the ones provided by anAE with security goals further than that of anAE and cAE. We refer to it as *usability-optimized, metadata-concealing AE* or μAE.

The metadata-concealing goal is mostly covered by anAE, which hides the nonce, AD, and any identifier for a key used to create a particular ciphertext. However, even anAE leaks the length of the message. This is potentially identity-revealing; one can imagine an adversary that can identify, with some probability, the sender of ciphertexts based on their lengths and its knowledge of which senders are predisposed to sending longer or shorter messages.

To address this, μAE would target *message length obfuscation* on top of the goals of anAE. It would do so by padding plaintexts before encrypting them. This can only hide message lengths to an extent. An observer will always knows that the plaintext length is less than or the same as that of the ciphertext regardless of the encryption scheme used. However, suppose one knows the high-level application that is employing the μAE scheme and knows that there will never be communications of more than n bits at a time. Padding messages to n bits before encrypting would then prevent any adversary from learning the true message's length from just observing the ciphertext.

On the usability end, there is room for a number of improvements. First off, we propose *flexible keying.* Typically, AE schemes assume that users possess a fixed-length, uniformly-random key. However, users may instead possess a password or a low-entropy key. Hence, there is some disconnect between what AE schemes expect and what users might possess. Usually, a tool outside of the AE scheme is expected to transform what the user possesses into something usable by the

AE scheme. We want such a tool to be a component of the scheme itself in μAE, guaranteeing some level of security when arbitrary passwords are used as keys.

Some other desirable features may be *ciphertext encoding* and *self-documenting ciphertexts.* For the former, secure AE schemes produce apparently random bytes when encrypting a plaintext. These bytes, however, may not be compatible with the user's application. For example, their usage might demand a base64url-encoded string or human-readable characters. Having the encoding be part of the encryption pipeline can enhance usability. For the latter, within AE, there are a variety of choices one can make for ciphers and parameters to encrypt a message. Rarely is it mentioned how the receiver of a ciphertext is supposed to know what parameters to use to decrypt. Like the transfer of metadata, this is usually considered out-of-scope. This feature would annotate ciphertexts in such a way that receivers can discern which ciphers and parameters were used in their creation.

We imagine that all of these features are bundled up into a single primitive: μAE. The primitive would shift away as much burden from the user as possible. It takes whatever they possess as a key (perhaps they do not even possess a password) and outputs a metadata-protected, length-obfuscated, encoded ciphertext with parameter annotations, fully ready for decryption. This covers an entire pipeline going from what little a user may provide (a plaintext and maybe a password) to a ciphertext that only needs a key to decrypt (as per anAE in addition to the documenting and encoding features). We leave the studying and development of such a primitive and the exploration of any other desirable features to future work.

Bibliography

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, Mar. 2002.

[2] M. Abdalla, M. Bellare, and G. Neven. Robust encryption. In D. Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 480–497, Zurich, Switzerland, Feb. 9–11, 2010. Springer, Heidelberg, Germany.

[3] F. Abed, C. Forler, and S. Lucks. General classification of the authenticated encryption schemes for the CAESAR competition. *Computer Science Review*, 22:13–26, 2016.

[4] A. Albertini, T. Duong, S. Gueron, S. Kölbl, A. Luykx, and S. Schmieg. How to abuse and fix authenticated encryption without key commitment. Cryptology ePrint Archive, Report 2020/1456, 2020. `https://eprint.iacr.org/2020/1456`.

[5] J. Albrecht. Introducing subscribe with google, Mar 2018. `https://blog.google/outreach-initiatives/google-news-initiative/introducing-subscribe-google/`.

[6] Amazon web services key management service, 2022. `https://aws.amazon.com/kms/`.

[7] Microsoft azure key vault, 2022. `https://azure.microsoft.com/en-us/services/key-vault/#product-overview`.

[8] F. Banfi and U. Maurer. Anonymous symmetric-key communication. In C. Galdi and V. Kolesnikov, editors, *SCN 20: 12th International Conference on Security in Communication Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 471–491, Amalfi, Italy, Sept. 14–16, 2020. Springer, Heidelberg, Germany.

[9] M. Bellare. 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`.

[10] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 566–582, Gold Coast, Australia, Dec. 9–13, 2001. Springer, Heidelberg, Germany.

[11] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, Florida, Oct. 19–22, 1997. IEEE Computer Society Press.

[12] M. Bellare and V. T. Hoang. Efficient schemes for committing authenticated encryption. In O. Dunkelman and S. Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture*

Notes in Computer Science, pages 845–875, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.

[13] M. Bellare, T. Kohno, and C. Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm. *ACM Trans. Inf. Syst. Secur.*, 7(2):206–241, 2004.

[14] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, Oct. 2008.

[15] M. Bellare, R. Ng, and B. Tackmann. Nonces are noticed: AEAD revisited. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 235–265, Santa Barbara, CA, USA, Aug. 18–22, 2019. Springer, Heidelberg, Germany.

[16] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330, Kyoto, Japan, Dec. 3–7, 2000. Springer, Heidelberg, Germany.

[17] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[18] M. Bellare and B. Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 247–276, Santa Barbara, CA, USA, Aug. 14–18, 2016. Springer, Heidelberg, Germany.

[19] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, May 2015. `https://www.rfc-editor.org/info/rfc7540`.

[20] NaCl: networking and cryptography library, Mar 2016. `https://nacl.cr.yp.to/index.html`.

[21] D. Bernstein. Cryptographic competitions: CAESAR call for submissions. Webpage, Jan. 2014. `https://competitions.cr.yp.to/caesar-call.html`.

[22] D. J. Bernstein, T. Lange, and P. Schwabe. The security impact of a new cryptographic library. In A. Hevia and G. Neven, editors, *Progress in Cryptology - LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America*, volume 7533 of *Lecture Notes in Computer Science*, pages 159–176, Santiago, Chile, Oct. 7–10, 2012. Springer, Heidelberg, Germany.

[23] C. Boyd, B. Hale, S. F. Mjølsnes, and D. Stebila. From stateless to stateful: Generic authentication and authenticated encryption constructions with application to TLS. In K. Sako, editor, *Topics in Cryptology – CT-RSA 2016*,

volume 9610 of *Lecture Notes in Computer Science*, pages 55–71, San Francisco, CA, USA, Feb. 29 – Mar. 4, 2016. Springer, Heidelberg, Germany.

[24] R. T. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, Oct. 1989. `https://www.rfc-editor.org/info/rfc1122`.

[25] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In B. S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 90–104, Santa Barbara, CA, USA, Aug. 17–21, 1997. Springer, Heidelberg, Germany.

[26] J. Chan and P. Rogaway. Anonymous AE. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 183–208, Kobe, Japan, Dec. 8–12, 2019. Springer, Heidelberg, Germany.

[27] J. Chan and P. Rogaway. On committing authenticated encryption. In *ESORICS 2022*, Lecture Notes in Computer Science, 2022.

[28] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In M. Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX, 2004.

[29] Y. Dodis, P. Grubbs, T. Ristenpart, and J. Woodage. Fast message franking: From invisible salamanders to encryptment. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 155–186, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany.

[30] P. Farshim, B. Libert, K. G. Paterson, and E. A. Quaglia. Robust encryption, revisited. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 352–368, Nara, Japan, Feb. 26 – Mar. 1, 2013. Springer, Heidelberg, Germany.

[31] P. Farshim, C. Orlandi, and R. Roşie. Security of symmetric primitives under incorrect usage of keys. Cryptology ePrint Archive, Report 2017/288, 2017. `https://eprint.iacr.org/2017/288`.

[32] Y. Gertner and A. Herzberg. Committing encryption and publicly-verifiable signcryption. Cryptology ePrint Archive, Report 2003/254, 2003. `https://eprint.iacr.org/2003/254`.

[33] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[34] Cryptography: Android Developers, Oct 2021. `https://developer.android.com/guide/topics/security/cryptography`.

[35] Tink cryptographic library, Mar 2022. `https://developers.google.com/tink`.

[36] P. Grubbs, J. Lu, and T. Ristenpart. Message franking via committing authenticated encryption. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 66–97, Santa Barbara, CA, USA, Aug. 20–24, 2017. Springer, Heidelberg, Germany.

[37] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299, New York, NY, USA, Apr. 10–12, 2001. Springer, Heidelberg, Germany.

[38] T. Kohno, A. Palacio, and J. Black. Building secure cryptographic transforms, or how to encrypt and MAC. *IACR Cryptology ePrint Archive*, 2003:177, 2003.

[39] T. Krovetz and P. Rogaway. The software performance of authenticated-encryption modes. In A. Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327, Lyngby, Denmark, Feb. 13–16, 2011. Springer, Heidelberg, Germany.

[40] J. Len, P. Grubbs, and T. Ristenpart. Partitioning oracle attacks. Cryptology ePrint Archive, Report 2020/1491, 2020. `https://eprint.iacr.org/2020/1491`.

[41] J. Len, P. Grubbs, and T. Ristenpart. Partitioning oracle attacks. In M. Bailey and R. Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 195–212. USENIX Association, Aug. 11–13, 2021.

[42] libsodium - introduction, Mar 2022. `https://doc.libsodium.org/`.

[43] D. McGrew. An interface and algorithms for authenticated encryption. IETF RFC 5116, Jan. 2018. `https://datatracker.ietf.org/doc/html/rfc5116`.

[44] D. McGrew and J. Viega. The galois/counter mode of operation (gcm). *submission to NIST Modes of Operation Process*, 20:0278–0070, 2004.

[45] D. A. McGrew and J. Viega. The security and performance of the Galois/counter mode (GCM) of operation. In A. Canteaut and K. Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference in Cryptology in India*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355, Chennai, India, Dec. 20–22, 2004. Springer, Heidelberg, Germany.

[46] C. Namprempre, P. Rogaway, and T. Shrimpton. AE5 security notions: Definitions implicit in the CAESAR call. *IACR Cryptology ePrint Archive*, 2013:242, 2013.

[47] C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering generic composition. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[48] Anonymous ae schemes, 2021. `https://github.com/jmchan100/Anonymous-AE`.

[49] OpenSSL: Cryptography and SSL/TLS Toolkit, 2022. `https://www.openssl.org/docs/manmaster/man7/crypto.html`.

[50] Oracle key vault, 2022. `https://www.oracle.com/security/database-security/key-vault/`.

[51] pyca/cryptography, 2022. `https://cryptography.io/en/latest`.

[52] Python 2.7.14, 2017. `https://www.python.org/downloads/release/python-2714/`.

[53] E. Rescorla. HTTP Over TLS. RFC 2818, May 2000. `https://www.rfc-editor.org/info/rfc2818`.

[54] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, Aug. 2018. `https://www.rfc-editor.org/info/rfc8446`.

[55] E. Rescorla, H. Tschofenig, and N. Modadugu. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-dtls13-43, Internet Engineering Task Force, Apr. 2021. `https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43`.

[56] Internet Protocol. RFC 791, Sept. 1981. `https://www.rfc-editor.org/info/rfc791`.

[57] Transmission Control Protocol. RFC 793, Sept. 1981. `https://www.rfc-editor.org/info/rfc793`.

[58] P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 2002: 9th Conference on Computer and Communications Security*, pages 98–107, Washington, DC, USA, Nov. 18–22, 2002. ACM Press.

[59] P. Rogaway. Nonce-based symmetric encryption. In B. K. Roy and W. Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359, New Delhi, India, Feb. 5–7, 2004. Springer, Heidelberg, Germany.

[60] P. Rogaway. Formalizing human ignorance. In P. Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06: 1st International Conference on Cryptology in Vietnam*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228, Hanoi, Vietnam, Sept. 25–28, 2006. Springer, Heidelberg, Germany.

[61] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In M. K. Reiter and P. Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 196–205, Philadelphia, PA, USA, Nov. 5–8, 2001. ACM Press.

[62] P. Rogaway and T. Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. Cryptology ePrint Archive, Report 2006/221, 2006. `https://eprint.iacr.org/2006/221`.

[63] P. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[64] P. Rogaway and Y. Zhang. Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany.