Anonymous AE

John Chan and Phillip Rogaway

Department of Computer Science
University of California, Davis, USA

Abstract. 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.

Keywords: anonymous encryption · authenticated encryption · nonces · privacy · provable security · symmetric encryption

1 Introduction

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 [6,10,16,17,19], anAE treats privacy as a first-class goal. We insist that ciphertexts contain everything the receiver needs to decrypt other than its secret state (including its keys), and ask for privacy 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 [14,16,19], requires the user to provide a nonce not only to encrypt a plaintext, but also to decrypt a ciphertext. 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 paper 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 [17]). 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*, Nx, 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, Lx. 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 becomes 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 Head $= E_{K_1}(N)$. This is sent along with an nAE-encrypted Body $= \mathcal{E}^{N,A}_{K_2}(M)$. But the ciphertext $C =$ Head $\|$ 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 Head $= E_{K_1}(N \,\|\, 0^\rho \,\|\, H(AD))$. Look ahead to Fig. 2 for our scheme, NonceWrap. As a concrete example, if the nonce N is 12 bytes [12] and we use the degenerate hash $H(x) = \varepsilon$ (the empty string), then one could encrypt a plaintext M as $C = \mathrm{AES}_{K_1}(N \| 0^{32}) \,\|\, \mathrm{GCM}^{N,A}_{K_2}(M)$. 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 and 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.

Related work. 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) [7]. 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 [2], and none became finalists. Namprempre, Rogaway, and Shrimpton formalized Bernstein's idea by adjusting the nAE syntax and security notion [13]. 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 [5]. 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 [4,11,8,20]. 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* [3] 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.

2 Nonce-Based AE (nAE)

Background. An nAE scheme, a nonce-based AE scheme supporting associated data (AD), is determined by a function $\mathcal{E}$, the *encryption algorithm*, with signature $\mathcal{E}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$. We insist that $\mathcal{E}(K, N, A, \cdot)$ be injective for any K, N, A. This ensures that there's a well-defined function $\mathcal{D} = \mathcal{E}^{-1}$ with signature $\mathcal{D}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ defined by $\mathcal{D}(K, N, A, C) = M$ if

$\mathcal{E}(K, N, A, M) = C$ for some (unique) $M \in \mathcal{M}$, while $\mathcal{D}(K, N, A, C) = \perp$ otherwise. The symbol $\perp$ is used to indicate invalidity. We may write $\mathcal{E}_K^{N,A}(M)$ and $\mathcal{D}_K^{N,A}(C)$ for $\mathcal{E}(K, N, A, M)$ and $\mathcal{D}(K, N, A, C)$. We require that the message space $\mathcal{M} \subseteq \{0,1\}^*$ be a set of strings for which $M \in \mathcal{M}$ implies $\{0,1\}^{|M|} \subseteq \mathcal{M}$. Finally, we assume that $|\mathcal{E}_K^{N,A}(M)| = |M| + \tau$ where τ is a constant. We refer to τ as the *expansion* of the scheme.

Let $\mathcal{E} \colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ be an nAE scheme with expansion τ. A customary way to define nAE security [19,14] associates to an adversary $\mathcal{A}$ the real number $\mathbf{Adv}^{\mathrm{nae}}_{\mathcal{E}}(\mathcal{A}) = \Pr[K \xleftarrow{} \mathcal{K} \colon \mathcal{A}^{E_K(\cdot,\cdot,\cdot),\, D_K(\cdot,\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot,\cdot),\, \perp(\cdot,\cdot,\cdot)} \Rightarrow 1]$ where the four oracles behave as follows: oracle $E_K(\cdot,\cdot,\cdot)$, on input N, A, M, returns $\mathcal{E}(K, N, A, M)$; oracle $D_K(\cdot,\cdot,\cdot)$, on input N, A, C, returns $\mathcal{D}(K, N, A, C)$; oracle $\$(\cdot,\cdot,\cdot)$, on input N, A, M, returns $|M| + \tau$ uniform random bits; and oracle $\perp(\cdot,\cdot,\cdot)$, on input N, A, C, returns $\perp$. The adversary $\mathcal{A}$ is forbidden from asking its first oracle a query (N, A, M) if it previously asked a query (N, A', M'); nor may it ask its second oracle (N, A, C) if it previously asked its first oracle a query (N, A, M) and received a response of C.

Privacy-violating assumptions of nAE. The nAE definition quietly embeds a variety of privacy-unfriendly choices. Beginning with syntax, decryption is understood to be performed directly by a function, $\mathcal{D}$, that requires input of K, N, and A. This suggests that the receiver *knows* the right key to use, and that the ciphertext will be delivered within some context that explicitly identifies which session the communication is a part of. But explicitly flowing such information is damaging to privacy. Similarly, the nonce N and AD A are needed by the decrypting party, but flowing either will often prove fatal to anonymity.

Indistinguishability from random bits is routinely understood to buy anonymity: after all, if the encryption of M under keys K and K' are indistinguishable from random bits then they are indistinguishable from each other. But this glosses over the basic problem that the thing that's indistinguishable from random bits isn't everything the adversary will see.

3 Anonymous Nonce-Based AE (anAE)

Privacy principle. Our anAE notion can be seen as arising from a basic tenet of secure encryption, which we now make explicit.

> **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.

The principle implies that it is not OK to just exclude from our understanding of the word *ciphertext* the privacy-violating parts of a transmission that are needed to decrypt. One needs to understand the ciphertext more expansively.

Stated as above, the privacy principle may seem so obvious that it is silly to spell it out. But the fact that nAE blatantly violates this principle, despite being understood as an extremely strong notion of security, suggests otherwise.

While this paper focuses on privacy, attending to the full ciphertext would seem to be the appropriate move when it comes to understanding confidentiality and authenticity as well. Our formulation of anAE does so.

Figuring out *how* to reflect the privacy principle in a definition is non-trivial. We now turn to that task.

SYNTAX. An anAE scheme extends an nAE scheme with five additional algorithms. Formally, an anAE scheme is a six-tuple of deterministic algorithms $\Pi = (\mathtt{Init}, \mathtt{Term}, \mathtt{Asso}, \mathtt{Disa}, \mathtt{Enc}, \mathtt{Dec})$. They create a session, terminate a session, register an AD, deregister an AD, encrypt a plaintext, and decrypt a ciphertext. The encryption algorithm $\mathcal{E} = \mathtt{Enc}$ must be an nAE scheme in its own right. In particular, this means that $\mathtt{Enc}$ automatically has an inverse $\mathcal{D} = \mathtt{Enc}^{-1}$, which is *not* what we are denoting $\mathtt{Dec}$. Algorithms $\mathtt{Init}$, $\mathtt{Term}$, $\mathtt{Asso}$, $\mathtt{Disa}$, and $\mathtt{Dec}$ are run by the decrypting party (they are, in effect, an alternative to $\mathcal{D} = \mathtt{Enc}^{-1}$) and able to mutate its persistent state $\boldsymbol{K} \in \mathcal{K}$. Specifically,

- $\mathtt{Init}$, the receiver's *session-initialization algorithm*, takes a key $K \in \mathcal{K}$ and returns a session-ID $\ell \in \mathcal{L}$ that will subsequently be used to name this session. We assume that returned SIDs are always distinct.
- $\mathtt{Term}$, the receiver's *session-termination algorithm*, takes a session-ID $\ell \in \mathcal{L}$ and returns nothing.
- $\mathtt{Asso}$, the receiver's *AD-association algorithm*, on input of either $A \in \mathcal{A}$ or $(A, \ell) \in \mathcal{A} \times \mathcal{L}$, returns nothing.
- $\mathtt{Disa}$, the receiver's *AD-disassociation algorithm*, on input of either $A \in \mathcal{A}$ or $(A, \ell) \in \mathcal{A} \times \mathcal{L}$, returns nothing.
- $\mathtt{Dec}$, the receiver's *decryption algorithm*, takes as input a ciphertext $C \in \mathcal{C}$ and returns either $(\ell, N, A, M) \in \mathcal{L} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ or the symbol $\perp$.

The sets referred to above, all nonempty, are as follows:

- $\mathcal{A}$ is an arbitrary set, the *AD space*.
- $\mathcal{C}$ is a set of strings, the *ciphertext space*.
- $\mathcal{K}$ is a finite set of strings, the *key space*.
- $\boldsymbol{\mathcal{K}}$ is an arbitrary set, the receiver's *persistent state*.
- $\mathcal{L}$ is an arbitrary set, the *session names*.
- $\mathcal{M}$ is a set of strings, the *message space*.
- $\mathcal{N}$ is a finite set, the *nonce space*.

Observe that decryption via $\mathtt{Dec}$ is only given the ciphertext C (and, implicitly, the state that the receiver maintains) but is expected, from this alone, to return not only the message but also the operative SID, nonce, and AD. The SID identifies the operative key. For the remainder of the text, we treat the SIDs as natural numbers, that is, we assume as $\mathcal{L} = \mathbb{N}$.

NONCE POLICY. In an AE scheme with stateful decryption [4,8,11,20] the receiver will, at any given point in time, have some set of nonces that it deems acceptable. We allow this set to depend on the nonces already received, but on nothing else. We formalize this by defining a *nonce policy* as a function $\mathrm{Nx}\colon \mathcal{N}^{\leq d} \to \mathcal{P}(\mathcal{N})$.

By $\mathcal{P}(\mathcal{S})$ we mean the set of all subsets of the set $\mathcal{S}$. The name Nx is meant to suggest the words *next* and *nonce*. The set $Nx(\boldsymbol{N})$ are the *permissible nonces* given the history $\boldsymbol{N}$. The history is a list of previously received nonces. The value $d = \mathrm{depth}(Nx) \in \mathbb{N} \cup \{\infty\}$ is the *depth* of the policy, capturing how many nonces one needs to record in order to know what the next nonce may be. One could reasonably argue that practical nonce policies must have bounded depth, as they would otherwise require the receiver to maintain unlimited state, and decryption would slow as connections grew old. The value $b = \mathrm{breadth}(Nx) = \max_{\boldsymbol{N} \in \mathrm{dom}(Nx)} |Nx(\boldsymbol{N})|$ is the *breadth* of the policy, the maximum number of permissible nonces. For a function $F\colon \mathcal{A} \to \mathcal{B}$ we are writing $\mathrm{dom}(F) = \mathcal{A}$ for its domain. Similarly, we write $\mathrm{range}(F) = \mathcal{B}$ for its range.

We single out two policy extremes. The *permissive* policy $Nx(\Lambda) = \mathcal{N}$ captures what happens in a stateless AE scheme, where repetitions, omissions, and out-of-order delivery are all *permitted*. (The symbol Λ denotes the empty list.) The permissive policy has depth $d = 0$ and breadth $b = |\mathcal{N}|$. Note that while the decryption algorithm *itself* treats all nonces as permissible, there could be some other, higher-level process that restricts this. At the other extreme, assuming a nonce space of $\mathcal{N} = [0..N_{\max}]$, the *strict* policy $Nx(\Lambda) = \{0\}$, $Nx((N)) = \{N+1\}$ (for $N < N_{\max}$), and $Nx((N_{\max})) = \emptyset$ demands an absence of repetitions, omissions, and out-of-order delivery. The nonce starts at zero and must keep incrementing. The depth d and breadth b are both 1. On a reliable channel, this is a natural policy. There is a rich set of policies between these extremes [4,8,11,20].

AD REGISTRATION. A sender may have some data that needs to be authenticated with the ciphertext it sends. Flowing that data in the clear would compromise anonymity. Instead, the receiver will maintain a set of AD values for each session. We can register or remove AD values one-by-one with `Asso` and `Disa`.

There are use cases where an AD value may not be specific to a session. For example, the use of AD in TLS 1.3 does not involve session-specific information; instead, the AD consists of several constants along with the ciphertext length.[1] To accommodate this, we envisage a further set of AD values that are effectively registered to *all* sessions. We refer to this as the set of *global* ADs. These too are added and removed one at a time. When a ciphertext needs to be decrypted, the only AD values that can match it are the global ones and those registered for the session that the ciphertext is seen as belonging to (which the decrypting party will have to determine).

Despite the generality of this treatment, the utility of AD is limited in anAE precisely because AD values *can't* flow in the clear; the only AD values that parties should use are those that can be determined *a priori* by the receiver.

DEFINING SECURITY. Let $\Pi = (\mathtt{Init}, \mathtt{Term}, \mathtt{Asso}, \mathtt{Disa}, \mathtt{Enc}, \mathtt{Dec})$ be an anAE scheme and let Nx be a nonce policy. The anAE security of Π with respect

[1] While we define anAE to accommodate this use case, it was pointless for TLS to put length of the ciphertext in the AD: nAE ensures that ciphertexts are authenticated, which implies that their length is authenticated. Throwing $|C|$ into the AD contributes nothing to security but does add complexity.

to Nx is captured by the pair of games in Fig. 1. The adversary interacts with either the $\mathrm{Real}^{\mathrm{anae}}_{\Pi,Nx}$ game or the $\mathrm{Ideal}^{\mathrm{anae}}_{\Pi,Nx}$ game and tries to guess which. The advantage of $\mathcal{A}$ attacking Π with respect to Nx is defined as

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

the difference in probability that the adversary outputs "1" in the two games.

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

We use *associative arrays* (also called *maps* or *dictionaries*) both in our games defining security and in the NonceWrap scheme itself. These are collections of (key, value) pairs with at most one value per key. We write $A[K]$ for doing a `lookup` in A for the value associated to the key K, returning that value. We write $A[K] \leftarrow X$ to mean adding or reassigning value X to key K. We write A.keys to denote the set of all keys in A. Similarly, A.values denotes the set of all values in A. The 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.

Explanation. The "real" anAE game surfaces to the adversary the six procedures of an anAE scheme. Modeling correct use, the `Init` procedure generates random keys, while calls to `Enc` may not repeat a nonce within the given session, nor may they employ a fictitious SID or the SID of a terminated session. The game does the needed bookkeeping to keep track of those things, with K_ℓ being the key associated to session ℓ and L recording the set of active session labels and $\mathrm{NE}[\ell]$ being the set of nonces already used for session ℓ.

The "ideal" anAE game provides the same entry points as the "real" one but employs the protocol Π only insofar as INIT returns the same sequence of labels used by `Init` and, also, the ideal game uses the expansion constant σ from `Enc`. The sequence of labels returned by INIT could just as well have been fixed as $1, 2, 3, \ldots$. The central idea is that encryption returns uniformly random bits (line 242) regardless of the SID, nonce, AD, or plaintext. This captures both confidentiality and anonymity, and in a strong sense. The same idea is used in the ind -form of the nAE definition, but the constraint isn't on the full ciphertext.

As with the all-in-one definition for nAE [19], authenticity is ensured by having the counterpart of the real decryption oracle routinely return $\perp$. When should it *not* return $\perp$? As with nAE, we want the ideal game to return $\perp$ if the ciphertext C was not previously returned from an ENC query (line 250). But we *also* want DEC to return $\perp$ if the relevant session has been torn down, if the relevant nonce is out-of-policy, or if the relevant AD is unregistered. We also want DEC to return $\perp$ if there is more than one in-policy explanation for this ciphertext. All of this is captured in lines 250–253. To express those lines, we need more bookkeeping than the real game did, also recording, in H (for "history"), the (ℓ, N, A, M) value(s) that gave rise to C (line 244); recording

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

procedure INIT()
100 $K \twoheadleftarrow \mathcal{K}$
101 $\ell \leftarrow \Pi.\texttt{Init}(K)$ *Guaranteed new*
102 $\text{K}[\ell] \leftarrow K$; $\text{L} \xleftarrow{\cup} \{\ell\}$; $\text{NE}[\ell] \leftarrow \emptyset$
103 **return** ℓ

procedure TERM(ℓ)
110 $\Pi.\texttt{Term}(\ell)$; $\text{L} \xleftarrow{>} \{\ell\}$

procedure ASSO(A)
120 $\Pi.\texttt{Asso}(A)$
procedure ASSO(A, ℓ)
121 $\Pi.\texttt{Asso}(A, \ell)$

procedure DISA(A)
130 $\Pi.\texttt{Disa}(A)$
procedure DISA(A, ℓ)
131 $\Pi.\texttt{Disa}(A, \ell)$

procedure ENC(ℓ, N, A, M)
140 **if** $\ell \notin \text{L}$ **or** $N \in \text{NE}[\ell]$ **then**
141 **return** $\perp$
142 $\text{NE}[\ell] \xleftarrow{\cup} \{N\}$
143 **return** $\Pi.\texttt{Enc}(\text{K}[\ell], N, A, M)$

procedure DEC(C)
150 **return** $\Pi.\texttt{Dec}(C)$

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

procedure INIT()
200 $K \twoheadleftarrow \mathcal{K}$
201 $\ell \leftarrow \Pi.\texttt{Init}(K)$
202 $\text{A}[\ell] \leftarrow \emptyset$; $\text{L} \xleftarrow{\cup} \{\ell\}$
203 $\text{NE}[\ell] \leftarrow \emptyset$; $\text{ND}[\ell] \leftarrow \Lambda$
204 **return** ℓ

procedure TERM(ℓ)
210 $\text{L} \xleftarrow{>} \{\ell\}$

procedure ASSO(A)
220 $\text{AD} \xleftarrow{\cup} \{A\}$
procedure ASSO(A, ℓ)
221 $\text{A}[\ell] \xleftarrow{\cup} \{A\}$

procedure DISA(A)
230 $\text{AD} \xleftarrow{>} \{A\}$
procedure DISA(A, ℓ)
231 $\text{A}[\ell] \xleftarrow{>} \{A\}$

procedure ENC(ℓ, N, A, M)
240 **if** $\ell \notin \text{L}$ **or** $N \in \text{NE}[\ell]$ **then**
241 **return** $\perp$
242 $C \twoheadleftarrow \{0,1\}^{|M|+\tau}$
243 $\text{NE}[\ell] \xleftarrow{\cup} \{N\}$
244 $\text{H}[C] \xleftarrow{\cup} \{(\ell, N, A, M)\}$
245 **return** C

procedure DEC(C)
250 **if** $H[C] = \emptyset$ **then return** $\perp$
251 **if** $\exists$ unique $(\ell, N, A, M) \in \text{H}[C]$ s.t.
252 $\ell \in \text{L}$ **and** $N \in Nx(\text{ND}[\ell])$ **and**
253 $A \in \text{AD} \cup \text{A}[\ell]$ **then**
254 $\text{ND}[\ell] \xleftarrow{\|} N$
255 **if** $|\text{ND}[\ell]| > d$ **then**
256 $\text{ND}[\ell] \leftarrow \text{tail}(\text{ND}[\ell])$
257 **return** (ℓ, N, A, M)
258 **return** $\perp$

Fig. 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.

in ND[ℓ] the sequence of nonces already observed on session ℓ (lines 255–256 truncate the history to only that needed for our decision making); and recording in associative arrays AD and A[ℓ] the currently registered AD values.

1AD/Session. We anticipate that, in most settings, the user will associate a single AD to a session at any given time. It might be associated to a particular session, or to all sessions, but, once a session has been identified, there is an understood AD for it. A decrypting party that operates in this way is said to be following the *one-AD-per-session restriction*, abbreviated 1AD/session.

Stateless schemes. Our formalization treats the decrypting party as stateful. Even if there was only one session and one AD, the decrypting party should register K with an `Init` call, register A with an `Asso` call, and then call `Dec`(C). But this sort of use of state is an artifact of the generality of our formulation. To draw out this distinction, we say that an anAE scheme is *stateless* if calls to its `Dec` algorithm never modify the receiver state.

For stateless anAE, one might provide an alternative API in which keys and AD are provided on each call, as in `Dec1`(K, A, C). Alternatively, one could initialize a data structure to hold the operative keys and AD values, and this data structure would be provided for decryption, but not side-effected by it. That is what happens in most crypto libraries today, where it is not a string-valued key that is passed to the encryption or decryption algorithms, but an opaque data structure created by a key-preprocessing step.

4 The NonceWrap Scheme

Ciphertext structure. Encryption under NonceWrap is illustrated in Fig. 2. The method uses two main primitives: an n-bit blockcipher E and an nAE scheme $\mathcal{E}$. The blockcipher is invoked once for each message encrypted, while the nAE scheme does the bulk of the work. NonceWrap also employs a hash function H, but it is used only for AD processing, outputs only a few bits (we do not seek collision-resistance), and indeed there is no security property from H on which we depend. A poor choice of H (like the constant function) would slow down decryption (in the case of multiple AD values per session), but would have no other adverse effect.

There are two parts to a NonceWrap-produced ciphertext: a header and a body. The header Head would typically be 16 bytes. It not only encodes the nonce N, which would usually be 12 bytes [12], but also some redundancy and a hash of the AD. To create a ciphertext C, the header is generated using a blockcipher E and is prepended to the ciphertext body Body, which is produced using nAE encryption on the nonce, AD, and plaintext. The total length of the ciphertext for M is $|M| + \lambda + \tau$ where λ is the header length—which is, for now, the blocksize $\lambda = n$ of E, and τ is the expansion of the nAE scheme.

When presented with a ciphertext $C = \text{Head} \,\|\, \text{Body}$, a receiver will often be able to determine that it does not belong to a candidate session just by looking at the prefix Head. It is deciphered with the candidate session key and if the

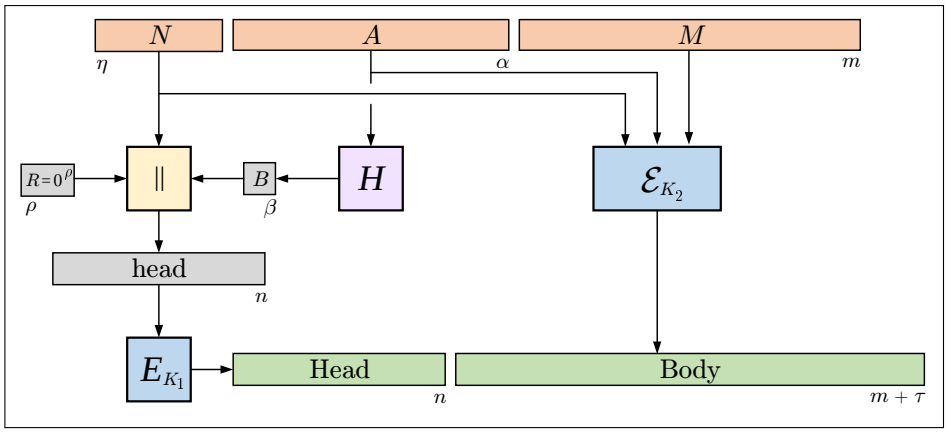


Fig. 2. Scheme illustration. NonceWrap encryption outputs a ciphertext that consists of two parts: a header Head, which is produced from a blockcipher E, and a body Body, which is produced from an nAE scheme $\mathcal{E}$. The hashed AD in the header can be omitted in the customary case where there is one AD per session at any time.

resulting block does not contain the mandated block of zero bits, or the nonce is not within nonce policy, or if the hash field does not contain the hash of a registered AD for this session, then the ciphertext as a whole must be invalid.

The hash of the AD is omitted if 1AD/session is assumed (equivalently, the hash returns the empty string). With a 16-byte header encrypting a 12-byte nonce, there would then be 4 bytes of zeros and roughly a 2^{-32} chance that a header for one session would be considered as a plausible candidate for another. When that does happen, it results in an nAE decryption of a ciphertext Body, not attribution of a ciphertext to an incorrect session. For *that* to happen, the ciphertext body would also have to verify as authentic when decrypted under the incorrect key. It is a little-mentioned property of nAE-secure encryption that a plaintext encrypted under one random key will almost always be deemed inauthentic when decrypted under an independent random key.

Anticipated nonces. Since the header is computed from the nonce and AD, it may be possible for the receiver to precompute a header before it arrives. This is because the nonce must fall within the protocol's nonce policy and the AD must be registered either specifically to a session or globally across sessions. But even under the 1AD/session assumption, the number of potential headers to precompute would be large if the breadth of the nonce policy is large (as with the permissive policy $Nx(\Lambda) = \mathcal{N}$). To get around this, we introduce a function to name the *anticipated* (or *likely*) nonces, Lx. Given the last few nonces received so far, it returns the nonce or set of nonces that are likely to come next. This is in contrast with Nx, which names the set of nonces that are *permissible* to come next—anything else should be deemed inauthentic. Like the nonce policy Nx,

the signature of the anticipated-nonce function is $Lx\colon \mathcal{N}^{\leq d} \to \mathcal{P}(\mathcal{N})$. We demand that that which is likely is possible: $Lx(\boldsymbol{n}) \subseteq Nx(\boldsymbol{n})$ for all $\boldsymbol{n} \in \mathcal{N}^{\leq d}$.

An anAE scheme that employs Lx and Nx functions is said to be *sharp* if $Lx = Nx$. With a sharp scheme, a ciphertext must be deemed invalid if it employs an unanticipated nonce. Sharpness can aid in efficient decryption.

Algorithmic details. We now descend more deeply into the structure of NonceWrap. The construction is defined in Fig. 3 and a list of data structures employed is given in Fig. 4.

The NonceWrap scheme maintains a number of dictionaries. The dictionary LNA maps anticipated headers to the set of session, nonce, and AD 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 is torn down, all headers belonging to that session are expunged from the dictionary. When a new AD is registered globally, headers are precomputed for each session and their anticipated nonces. If the AD is registered specific to a session, headers are computed for just that session. ADs are also managed in their own associative arrays—one for global and one for session-specific—that map AD hashes to sets of ADs that are preimages of the hash. Deregistering an AD removes it from its respective array and expunges its associated headers from the main dictionary.

NonceWrap decryption comes in three phases. Phase-1 attempts to use the precomputed headers in LNA to quickly determine which session, nonce, and AD are associated to a received ciphertext. As there may be multiple (ℓ, N, A) triples mapped to the header, the receiver tries to decrypt the ciphertext body with each until it arrives at a valid message. If no message is found within this phase, then it falls through to the phase-2, where it attempts to extract the nonce and AD directly by trial-deciphering the header. The receiver tries each session key on the header until it finds a nonce within the session's policy appended with ρ redundant 0-bits. If there are multiple AD values per session, the hash of an AD would be appended. If the AD is properly registered with the receiver, then the receiver has a mapping between the AD hash and its possible preimages. With this, the receiver may now trial-decrypt the ciphertext body. The second phase is repeated until a valid message is found. If none is, then decryption returns $\perp$ and the ciphertext is deemed invalid.

If either phase-1 or phase-2 recovers a valid plaintext, they go into phase-3, where precomputation for the next anticipated header occurs. Entering phase-3 means the receiver knows the (ℓ, N, A, M) for the ciphertext. It can then compute the old set of anticipated nonces prior to receiving N using Lx. It can also compute a new set of anticipated nonces with a nonce history updated with N. With the former, it can expunge all old headers from LNA and, with the latter, it can populate LNA with the next expected headers.

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

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

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

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

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

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

$\underline{\Pi.\texttt{Dec}(C)}$ *Phase-1 (starting at 51)*
50 Head $\parallel$ Body $\leftarrow C$ **where** $|\text{Head}| = n$
51 **for** $(\ell, N, A) \in \boldsymbol{K}.\text{LNA}[\text{Head}]$ **do**
52 $K_1 \leftarrow \boldsymbol{K}.\text{K1}[\ell]$; $K_2 \leftarrow \boldsymbol{K}.\text{K2}[\ell]$
53 $M \leftarrow \mathcal{D}(K_2, N, A, \text{Body})$
54 **if** $M \neq \perp$ **then goto** 5F
55 **for** $\ell \in \boldsymbol{K}.\text{L}$ **do** *Phase-2*
56 $K_1 \leftarrow \boldsymbol{K}.\text{K1}[\ell]$; $K_2 \leftarrow \boldsymbol{K}.\text{K2}[\ell]$
57 $N \parallel R \parallel B \leftarrow E_{K_1}^{-1}(\text{Head})$
58 **where** $|N| = \eta$ and $|R| = \rho$
59 **if** $R \neq 0^\rho$ **or** $N \notin Nx(\boldsymbol{K}.\text{N}[\ell])$ **then**
5A **continue**
5B **for** $A \in \boldsymbol{K}.\text{A}[\ell][B] \cup \boldsymbol{K}.\text{AD}[B]$ **do**
5C $M \leftarrow \mathcal{D}(K_2, N, A, \text{Body})$
5D **if** $M \neq \perp$ **then goto** 5F
5E **return** $\perp$
5F Old $\leftarrow Lx(\boldsymbol{K}.\text{N}[\ell])$ *Phase-3*
5G $\boldsymbol{K}.\text{N}[\ell] \xleftarrow{\parallel} N$
5H **if** $|\boldsymbol{K}.\text{N}[\ell]| > d$ **then**
5I $\boldsymbol{K}.\text{N}[\ell] \leftarrow \text{tail}(\boldsymbol{K}.\text{N}[\ell])$
5J New $\leftarrow Lx(\boldsymbol{K}.\text{N}[\ell])$
5K **for** $N' \in$ Old $\setminus$ New **do**
5L **for** $S \in \boldsymbol{K}.\text{LNA.values}$ **do**
5M $S \xleftarrow{\supset} \{\ell\} \times \{N'\} \times \mathcal{A}$
5N **for** $N' \in$ New $\setminus$ Old **do**
5O **for** $B \in \boldsymbol{K}.\text{A}[\ell].\text{keys} \cup \boldsymbol{K}.\text{AD.keys}$ **do**
5P Head $\leftarrow E_{K_1}(N' \parallel 0^\rho \parallel B)$
5Q **for** $A' \in \boldsymbol{K}.\text{A}[\ell][B] \cup \boldsymbol{K}.\text{AD}[B]$ **do**
5R $\boldsymbol{K}.\text{LNA}[\text{Head}] \xleftarrow{\cup} \{(\ell, N', A')\}$
5S **return** (ℓ, N, A, M)

Fig. 3. Constructing an anAE scheme. Scheme $\Pi = \text{NonceWrap}[E, H, \mathcal{E}, Lx, Nx]$ depends on a blockcipher $E\colon \{0,1\}^{k_1} \times \{0,1\}^n \to \{0,1\}^n$, a nonce policy $Nx\colon \{0,1\}^{\leq d} \to \mathcal{P}(\mathcal{N})$, a hash function $H\colon \{0,1\}^* \to \{0,1\}^\beta$, an nAE scheme $\mathcal{E}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ and an anticipated nonce function Lx which always outputs a subset of what policy Nx permits. Data structures employed are described in Fig. 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

Fig. 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 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 a sharp policy we may safely omit phase-2 and get a decryption time of $O(m + a + t + \lg s)$ for any ciphertext.

Optimizations. For a *sharp* scheme, $Nx = Lx$, the anticipated nonces within LNA encompass all valid nonces; a header not stored in the dictionary is necessarily invalid. For such a scheme, phase-2 can be ignored. This improves efficiency and allows for some natural simplifications. In addition, in this case we never compute the inverse E^{-1} of the blockcipher, so there is not longer any *need* for it to be a blockcipher. One can therefore replace the blockcipher $E\colon \{0,1\}^n \to \{0,1\}^n$ by a PRF $F\colon \{0,1\}^n \to \{0,1\}^\lambda$ where λ is considerable smaller than n. One or two bytes would typically suffice. After all, all that happens when a collision *does* occur is that one needs to perform a trial decryption of the ciphertext body. A convenient way to construct the PRF F would be by truncating the blockcipher E, setting $F_{K_1}(x) = (E_{K_1}(x))[1..\lambda]$.

At the opposite extreme, when $Lx(\boldsymbol{n}) = \emptyset$ for all $\boldsymbol{n}$, NonceWrap does not anticipate *any* nonces. In that case *only* the phase-2 is executed, and LNA can be disregarded. This variant is close to standard nAE decryption, and is useful when we need a stateless receiver.

5 NonceWrap Security

Alternative characterization of nAE. We will find it convenient to use the following alternative formulation of nAE security, which directly models

multiple keys and more precisely attends to what is possible for a given expansion. 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 $f^{-1}(T,Y) = \bot$ otherwise. Now given an adversary $\mathcal{A}$, define its advantage in attacking the nae-security of $\mathcal{E}$ as the real number

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

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 E_f, on query (i,N,A,M), returns $f((i,N,A),M)$; and oracle D_f, 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, following the PRI-characterization of misuse-resistant AE schemes [18], to show the equivalence of nae (presented in section 2) and nae$*$ security.

Multi-key strong-PRP security. It's also useful for us 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

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

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 1. *There exists a reduction , 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 $\mathcal{E}\colon \mathcal{K}_\mathcal{E} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}_\mathcal{E}$ be an nAE scheme, and let $Nx\colon \mathcal{N}^{\le d} \to \mathcal{P}(\mathcal{N})$ be a nonce policy with depth d. Let Lx be an anticipated-nonce function with the same signature as Nx such that*

$Lx(\boldsymbol{n}) \subseteq Nx(\boldsymbol{n})$ for all $\boldsymbol{n} \in \mathcal{N}^{\leq d}$. Let Π = NonceWrap$[E, H, \mathcal{E}, Lx, Nx]$ 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 transforms $\mathcal{A}$ into a pair of adversaries ($\mathcal{B}_1$,$\mathcal{B}_2$) such that*

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

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 $\text{Inj}^{\mathbb{N}}(\{0,1\}^n)$. 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:

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

The next hybrid G_2 replaces NonceWrap's underlying nAE scheme $\mathcal{E}$ with a random function F from $\text{Inj}^{\mathbb{N}\times\mathcal{N}\times\mathcal{A}}_{\tau}(\mathcal{M})$. 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:

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

At this point we have a real anae game using a 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. 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

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

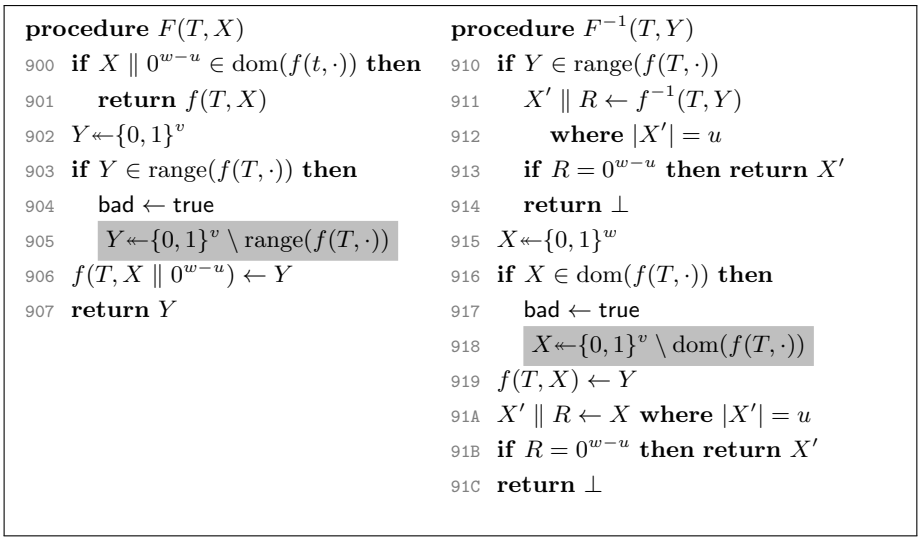
The next modification only changes how ciphertexts are generated. Instead of randomly sampling from $\{0,1\}^{|M|+\tau}$ on an encryption query, the encryption oracle will instead use a pair of PRIs to generate a "header" and "body" to create the ciphertext. To do this, we modify the code for the ENC oracle to use the procedure F defined in the top half of Fig. 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 $\mathrm{Inj}_{\tau}^{\mathbb{N}\times\mathcal{N}\times\mathcal{A}}(\mathcal{M})$ for creating the body and one from $\mathrm{Inj}^{\mathbb{N}}(\{0,1\}^n)$ 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 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 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

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

Our next hybrid G_5 changes the decryption oracle and is shown in Fig. 6. The other oracles remain the same. Instead of identifying the SID, nonce, and AD using H[C] right away, the oracle will search for the tuple by going through all $\ell \in \mathrm{L}$, $N \in N\mathrm{x}(\mathrm{ND}[\ell])$, and $A \in \mathrm{A}[\ell] \cup \mathrm{AD}$. For each of those tuples, it will try to invert the injection on Body to recover M. Now it's possible that the



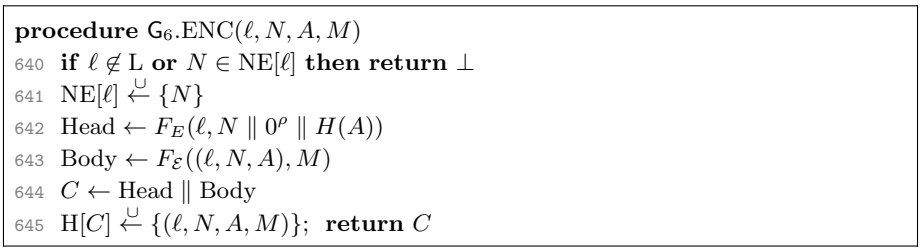


Fig. 5. Top. Lazy-sampling of random functions or injections in the multi-key setting. With the code in grey, the procedures simulate a random injection for each T from u bits to w bits. Without the code in grey, the procedures simulate a random function for each T. **Bottom.** Modified encryption oracle 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.

inversion results in an M that wasn't recorded in H since F^{-1} as defined in Fig. 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

```
procedure G5.DEC(C)
550 Head ‖ Body ← C where |Head| = n
551 for ℓ ∈ L do
552   for N ∈ Nx(ND[ℓ]) do
553     for A ∈ A[ℓ] ∪ AD do
554       M ← F_E^{-1}((ℓ, N, A), Body)
555       if (ℓ, N, A, M) ∈ H[C] then
556         ND[ℓ] ←‖ N
557         if ND[ℓ] ∉ dom(Nx) then ND[ℓ] ← tail(ND[ℓ])
558         return (ℓ, N, A, M)
559 return ⊥
```

Fig. 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.

```
procedure G4.ASSO(A)
420 B ← H(A); AD[B] ←∪ {A}
procedure G4.ASSO(A, ℓ)
421 B ← H(A); A[ℓ][B] ←∪ {A}

procedure G4.DISA(A)
430 B ← H(A); AD[B] ←⊻ {A}
procedure G4.DISA(A, ℓ)
431 B ← H(A); A[ℓ][B] ←⊻ {A}
```

```
procedure G4.DEC(C)   Resembles phase-2
450 Head ‖ Body ← C where |Head| = n
451 for ℓ ∈ L do
452   N ‖ R ‖ B ← F_E^{-1}(ℓ, Head)
453     where |N| = η and |R| = r
454   if R ≠ 0^ρ or N ∉ Nx(ND[ℓ])
455     then continue
456   for A ∈ A[ℓ][B] ∪ AD[B] do
457     M ← f_E^{-1}((ℓ, N, A), Body)
458     if (ℓ, N, A, M) ∈ H[C] then
459       ND[ℓ] ←‖ N
45A       if ND[ℓ] ∉ dom(Nx) then
45B         ND[ℓ] ← tail(ND[ℓ])
45C       return (ℓ, N, A, M)
45D return ⊥
```

Fig. 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.

hybrid's encryption contains the nonce 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. 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 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. 8, but disregard the phase-1 decryption block for now. First, we want to assert a property of LNA.

Lemma 2. *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* $\ell \in \mathrm{L}$, $N \in \mathit{Nx}(\mathrm{ND}[\ell])$, *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: $Lx(\mathrm{ND}[\ell])$ and $Lx(\mathrm{ND}[\ell] \parallel N')$. The former is Old and the latter is New in the pseudocode. The procedure iterates over New$\setminus$Old, which is a subset of $Nx(\mathrm{ND}[\ell] \parallel N')$ when adding new tuples.
  - When tuple is removed in DEC, the sets Old and New are used again. The procedure iterates over Old $\setminus$ New and removes tuples containing those nonces from LNA. Hence, any tuple with a nonce not in $Lx(\mathrm{ND}[\ell] \parallel N')$ is removed.

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

procedure 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 $\leftarrow N \parallel 0^\rho \parallel H(A)$
307 Head $\leftarrow F_E(\ell, \text{head})$
308 $\mathrm{LNA}[\text{Head}] \xleftarrow{\cup} \{(\ell, N, A)\}$
309 **return** ℓ

procedure G_3.TERM(ℓ)
310 **for** $S \in$ LNA.values **do**
311 $S \xleftarrow{\supset} \{\ell\} \times \mathcal{N} \times \mathcal{A}$
312 $\mathrm{L} \xleftarrow{\supset} \{\ell\}$

procedure G_3.ASSO(A)
320 $B \leftarrow H(A)$; $\mathrm{AD}[B] \xleftarrow{\cup} \{A\}$
321 **for** $\ell \in$ L **do**
322 **for** $N \in Lx(\mathrm{ND}[\ell])$ **do**
323 Head $\leftarrow F_E(\ell, N \parallel 0^\rho \parallel B)$
324 $\mathrm{LNA}[\text{Head}] \xleftarrow{\cup} \{(\ell, N, A)\}$
procedure G_3.ASSO(A, ℓ)
325 $B \leftarrow H(A)$; $\mathrm{A}[\ell][B] \xleftarrow{\cup} \{A\}$
326 **for** $N \in Lx(\mathrm{ND}[\ell])$ **do**
323 Head $\leftarrow F_E(\ell, N \parallel 0^\rho \parallel B)$
324 $\mathrm{LNA}[\text{Head}] \xleftarrow{\cup} \{(\ell, N, A)\}$

procedure G_3.DISA(A)
330 $B \leftarrow H(A)$; $\mathrm{AD}[B] \xleftarrow{\supset} \{A\}$
331 **for** $S \in$ LNA.values **do**
332 $S \xleftarrow{\supset} \mathcal{L} \times \mathcal{N} \times \{A\}$
procedure G_3.DISA(A, ℓ)
333 $B \leftarrow H(A)$; $\mathrm{A}[\ell][B] \xleftarrow{\supset} \{A\}$
334 **for** $S \in$ LNA.values **do**
335 $S \xleftarrow{\supset} \{\ell\} \times \mathcal{N} \times \{A\}$

procedure G_3.DEC(C) *Phase-1*
350 Head $\parallel$ Body $\leftarrow C$ **where** $|\text{Head}| = n$
351 **for** $(\ell, N, A) \in \mathrm{LNA}[\text{Head}]$ **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** $\ell \in$ L **do** *P-2, same as* G_4*'s*
356 $N \parallel R \parallel B \leftarrow F_E^{-1}(\ell, \text{Head})$
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 **return** $\perp$

35F Old $\leftarrow Lx(\mathrm{ND}[\ell])$ *Phase-3*
35G $\mathrm{ND}[\ell] \xleftarrow{\parallel} N$
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 $S \xleftarrow{\supset} \{\ell\} \times \{N'\} \times \mathcal{A}$
35N **for** $N' \in$ New $\setminus$ Old **do**
35O **for** $B \in \mathrm{A}[\ell].\text{keys} \cup \mathrm{AD}.\text{keys}$ **do**
35P Head $\leftarrow F_E(\ell, N' \parallel 0^\rho \parallel B)$
35Q **for** $A' \in \mathrm{A}[\ell][B] \cup \mathrm{AD}[B]$ **do**
35R LNA.[Head] $\xleftarrow{\cup} \{(\ell, N', A')\}$
35S **return** (ℓ, N, A, M)

Fig. 8. Hybrid game resembling NonceWrap. Game G_3 executes procedures similar to those of NonceWrap. For decryption on a ciphertext to succeed, it follows the ideal game. If decryption returns a tuple, then that tuple must have been used to make the queried ciphertext. The encryption oracle is omitted as it is the same as G_5's, which is in Fig. 6.

As per lemma 2, 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 $N\mathrm{x}(\mathrm{ND}[\ell])$, A must be in $\mathrm{A}[\ell] \cup \mathrm{AD}$, and the entire tuple must be in $\mathrm{H}[C]$. 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 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. 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.

□

6 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 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 [9], 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 [15]. 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?

Acknowledgments

We thank Dan Bernstein for inspiring this work. Within the CAESAR call, he suggested the use of "secret message numbers" in lieu of nonces; in private communications with the second author, he asked how one might efficiently demultiplex multiple AE communication streams without having marked them in a privacy-compromising manner.

We thank the anonymous ASIACRYPT referees. Their comments brought home that anonymous AE was a concern that transcended our formulation of it. They suggested the name anAE.

This work was supported by NSF CNS 1717542 and NSF CNS 1314855. Many thanks to the NSF for their years of financial support.

References

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. 4
2. 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
3. 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. 4

4. 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. 4, 6, 7
5. M. Bellare, R. Ng, and B. Tackmann. Nonces are noticed: AEAD revisited. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, Lecture Notes in Computer Science, pages 235–265, Santa Barbara, CA, USA, Aug. 18–22, 2019. Springer, Heidelberg, Germany. 4
6. 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. 1
7. D. Bernstein. Cryptographic competitions: CAESAR call for submissions. Webpage, Jan. 2014. `https://competitions.cr.yp.to/caesar-call.html`. 4
8. 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. 4, 6, 7
9. 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. 24
10. 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. 1
11. T. Kohno, A. Palacio, and J. Black. Building secure cryptographic transforms, or how to encrypt and MAC. *IACR Cryptology ePrint Archive*, 2003:177, 2003. 4, 6, 7
12. D. McGrew. An interface and algorithms for authenticated encryption. IETF RFC 5116, Jan. 2018. 3, 10
13. C. Namprempre, P. Rogaway, and T. Shrimpton. AE5 security notions: Definitions implicit in the CAESAR call. *IACR Cryptology ePrint Archive*, 2013:242, 2013. 4
14. 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. 1, 5
15. E. Rescorla, H. Tschofenig, and N. Modadugu. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-dtls13-31, Internet Engineering Task Force, Mar. 2019. Work in Progress. 25
16. P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 98–107, Washington D.C., USA, Nov. 18–22, 2002. ACM Press. 1
17. P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM CCS 01: 8th Conference on Computer and Communications Security*, pages 196–205, Philadelphia, PA, USA, Nov. 5–8, 2001. ACM Press. 1, 2
18. P. Rogaway and T. Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. Cryptology ePrint Archive, Report 2006/221, 2006. `http://eprint.iacr.org/2006/221`. 15

19. 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. 1, 5, 8
20. 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. 4, 6, 7