Authenticated Encryption with Key Identification

Julia Len[1], Paul Grubbs[2], and Thomas Ristenpart[1]

[1] Cornell Tech
[2] University of Michigan

Abstract. Authenticated encryption with associated data (AEAD) forms the core of much of symmetric cryptography, yet the standard techniques for modeling AEAD assume recipients have no ambiguity about what secret key to use for decryption. This is divorced from what occurs in practice, such as in key management services, where a message recipient can store numerous keys and must identify the correct key before decrypting. To date there has been no formal investigation of their security properties or efficacy, and the ad hoc solutions for identifying the intended key deployed in practice can be inefficient and, in some cases, vulnerable to practical attacks.

We provide the first formalization of nonce-based AEAD that supports key identification (AEAD-KI). Decryption now takes in a vector of secret keys and a ciphertext and must both identify the correct secret key and decrypt the ciphertext. We provide new formal security definitions, including new key robustness definitions and indistinguishability security notions. Finally, we show several different approaches for AEAD-KI and prove their security.

Keywords: Key identification, authenticated encryption, key commitment, key robustness

1 Introduction

Authenticated encryption with associated data (AEAD) is ubiquitously used in practice. Standard formalizations of AEAD schemes model a "single-key" setting where a single sender sends an encrypted message to a single receiver. Though simple to analyze, this model is increasingly divorced from practice in a number of important aspects.

A setting which has received little or no attention in the cryptographic literature is one where a message recipient will store numerous keys and must identify the correct key to use before decryption can proceed. This practice can be seen in cryptographic libraries such as Google's Tink API [32], key management services (KMS) such as for Amazon Web Services (AWS) [5], and multi-user Shadowsocks [30], among others. Notably, AEAD schemes and their security models do not formally address the issue of key identification, producing a gap when translating from cryptographic theory to practice.

Two approaches for key identification are often used in practice. The first is trial decryption, where one attempts to decrypt the ciphertext under each of the keys held by the recipient. However, this is slow, even for a small number of keys. As such, a second approach is often used: the sender attaches a previously agreed-upon key identifier to each message it sends. For instance, this approach is used by Tink, which derives 5-byte strings as the identifier. The recipient can then efficiently use the identifier to look up the corresponding key. However, this approach does not work in settings where keys must remain anonymous, such as in anonymous messaging protocols. It is also unclear what security properties (for both approaches) are being achieved in the case where there is potential for adversarial modification of key identifiers and/or adversarial choice of some of the recipient keys. Adversarially chosen keys can arise in settings where the sender chooses a secret key to share with the recipient, which have resulted in several recent attacks [2, 15, 19, 23].

One example is multi-user Shadowsocks [30]. Shadowsocks is an anonymity proxy that works by having a client encrypt their traffic under a shared password with a server, which decrypts using AEAD. The multi-user mode allows multiple passwords to be specified for a single server, which means that incoming packets must be trial decrypted under every possible user password. Len et al. [23] describe an attack on this scheme where an attacker can insert a malicious password into this set of passwords, then mount a partitioning oracle attack that enables the attacker to learn some target user's password. Fundamentally, the vulnerability is that Shadowsocks's AEAD has no efficient and secure way to identify the appropriate key.

Our contributions. We initiate the formal study of AEAD that supports key identification. The starting point is nonce-based AEAD [28], which we extend to include in the formal syntax and semantics of encryption schemes the key identification task: decryption takes in a vector of secret keys as well as a nonce, associated data, and a ciphertext, and must both identify the correct secret key and decrypt the ciphertext. This change, while conceptually simple, immediately introduces a number of complexities. It forces scheme designers to specify how the right key is identified, requires changes to notions of correctness, suggests that we must give new security definitions that speak to issues such as adversaries forcing the wrong key to be identified, and more.

We formalize a new cryptographic primitive called AEAD-KI, or AEAD with key identification. Like AEAD, the primitive is composed of a triple of algorithms for key generation, encryption, and decryption. Key generation takes in what we call a key label so that AEAD-KI keys are composed of the traditional secret key as well as the key label. This label acts as optional public metadata for the key and models techniques in practice, e.g., URLs that suggest where to locate the key or other kinds of static identifiers. Encryption takes in a key, nonce, associated data, and message. Ciphertexts can opt to include a special component, called a key tag. Decryption for AEAD-KI, in turn, accepts a vector of keys, instead of one key as for AEAD. We use a vector instead of a set to preserve information about the order of keys, which could affect the decryption

outcome. Given the ciphertext (including the key tag), decryption returns both the key that correctly decrypts the ciphertext as well as the resulting message. If decryption determines that no key correctly decrypts the ciphertext, it simply returns an error symbol $\perp$.

We next consider which security definitions best capture the AEAD-KI setting. Our first goal is to extend the standard AEAD security notions of confidentiality and ciphertext integrity to AEAD-KI. A good starting point is transforming the traditional all-in-one real-versus-random indistinguishability security notion for AEAD due to Rogaway and Shrimpton [29] to the AEAD-KI setting. Specifically, we can allow the attacker to interact with multiple encryption key instances, reminiscent of the multi-user setting for encryption [6]. However, this definition only allows for honest keys, which unfortunately does not capture attacks in which a malicious key is somehow inserted into a recipient's key vector. Indeed, attacks have already been shown in practice where malicious keys are given to a recipient to prevent decryption under honest keys [2, 15].

We therefore opt for a notion of security for which adversaries can insert malicious keys into key vectors used during decryption queries. This renders more complex how the security game should handle decryption oracles in order to distinguish between honest and malicious keys. To handle these subtleties, we introduce KI-nAE, a new security notion that uses a simulation-based approach for the all-in-one definition. This definition captures a wide class of interference attacks in which a malicious user somehow inserts a malicious key into a recipient's key vector in an attempt to interfere with honest keys.

A security property intrinsic to the key identification setting is key robustness, a security goal first investigated in the context of public-key encryption [1] and later investigated for authenticated encryption [17] (see also [2,8,15,19,23]). Interestingly, robustness here functions as a form of correctness, as ensuring the correct key decrypts a given ciphertext in the presence of many (potentially adversarial) keys can only be guaranteed by robustness. We thus extend the AEAD robustness notion called full robustness (FROB) [17] to the AEAD-KI setting, which is straightforward. Interestingly, this extension however proves insufficient to rule out some attacks. In particular, when decryption is given the correct key within a key set, it should not fail to decrypt; such failures could leak information about the (honest) keys composing a key vector. One way to handle this is with an extended key robustness notion, but an observation due to Mihir Bellare is that one can instead extend correctness to rule out such decryption failures. See the body and the full version of this work for more details.

Approaches to AEAD-KI. We then turn to analyzing security of existing key identification schemes as well as suggesting new ones. A summary of our analyses appears in Figure 1. We divide key identification into several categories. The first approach utilizes the key label of the key as the key tag itself. Decryption can then find all keys whose label matches the key tag of the ciphertext and perform trial decryptions. (Note the second category, trial decryption, is a special case where all labels are the empty string.) This reflects how, in practice, labels are sometimes not unique and instead used to label a set of keys. Using key

Approach	Description	AEAD FROB?	Key anon.?	Section
Key labels	Key gen. labels each key, sent as part of ciphertext; brute-force decrypt with all keys matching key label in ciphertext	Yes	No	§4
Trial decryption	Special case of key labels where all labels are empty (ε)	Yes	Yes	§4
Static key hint	Ciphertext includes deterministic non-CR hash of key	Yes	No	§5
Static key commitment	Ciphertext includes deterministic CR hash of key	No	No	§5
Dynamic key hint	Ciphertext includes PRF of key & nonce	Yes	Yes	§6
Dynamic key commitment	Ciphertext includes CR-PRF of key & nonce	No	Yes	§6

Fig. 1: Summary of various approaches to AEAD-KI that we consider. For each approach, we also list whether it must use a FROB AEAD scheme and whether it provides key anonymity.

labels obviates achieving key anonymity, since a ciphertext produced by a certain key will always be flagged by the key's label. Nevertheless, brute-force trial decryption, a special case of the key label approach, can achieve anonymity, although with the trade-off of increased computational costs. Since key labels are not unique, they do not provide key commitment to AEAD schemes that are not key committing. Nevertheless, we see this insecure construction arises in practice, such as in the Tink library. Our analysis shows that if one instead uses the key label approach with a key-committing AEAD scheme, then the composition is key committing.

Next we turn to what we call "static" approaches, those where key tags are deterministically computed from the secret key. These are similar to "key check values", legacy schemes for ensuring integrity of the key [18,27,31]. Since the key tag for a key never changes, this approach also does not allow for key anonymity. We further divide static key identifiers into two classes: static key hints and static key commitments. Key hints are key tags computed from the key in a non-key-committing way, typically using a non-collision resistant hash of the key. This means that AEAD-KI schemes using key hints, like key labels, will need to trial decrypt on all keys matching the key hint and use a FROB AEAD scheme to achieve key robustness. The benefit of key hints is that they can often be short and efficiently computed. In contrast, static key commitments, the second class of static key identifiers, do commit to the key. These are typically a collision-resistant hash of the key. While static key commitments might be less efficient to compute than key hints, they can be used to build secure AEAD-KI from non-FROB AEAD schemes.

Finally, both key hints and key identifiers can be made "dynamic" to provide key anonymous counterparts of static schemes. This approach uses a nonce when computing the key tag so that the key tag for a key is unique for each encryption call. Like static identifiers, dynamic identifiers can be both key hints and key

commitments, with security achieved when combining with any AEAD or FROB AEAD, respectfully.

Further related work. Farshim, Orlandi, and Rosie [17] first proposed the set of key robustness notions for symmetric primitives. Their strongest definition, full robustness, represents the goal of key commitment for AEAD schemes. Grubbs et al. [19] suggest a notion of compactly committing AEAD, which is useful in abuse moderation settings. Dodis et al. [15] describe an attack against Facebook Messenger's abuse moderation tooling that relies on AES-GCM, which is not key committing. In this attack, a malicious sender can force an honest recipient to use a malicious key to decrypt a message to abusive content that cannot be reported. Len et al. [23] and Albertini et al. [2] showed that other commonly used AEAD schemes, such as AES-GCM-SIV, ChaCha20-Poly1305, and OCB3, are not key committing and they describe other practical attack scenarios that exploit non-key committing AEAD schemes.

Albertini et al. also propose two approaches to adding key commitment to AEAD schemes. One of these approaches is to compute a collision-resistant PRF of the key, both deterministically and using a nonce for key anonymity. Our static and dynamic identifiers parallel these schemes, but for the AEAD-KI setting. Bellare and Hoang [9] propose a spectrum of new definitions for commitment that capture not just committing to the key but also committing to the nonce, associated data, and plaintext message. They also propose new key-committing AEAD schemes based on AES-GCM and AES-GCM-SIV as well as a construction that transforms a legacy AEAD scheme into one that is key-committing using what they call a *committing PRF*. This construction is similar to that proposed by Albertini et al. (and indeed they note that Albertini et al.'s construction can be viewed as a specific instantiation of their scheme). Their work considers only the single-key setting, but their schemes can be used as the necessary FROB AEAD schemes in our AEAD-KI constructions.

Degabriele et al. [14] propose nonce-set AEAD, which is similar to our AEAD-KI formalism but instead for nonce sets. Their formalism considers decryption accepting a set of nonces and then returning the correct nonce along with the plaintext message.

Chan and Rogaway [13] formalize anonymous authenticated encryption, which requires that ciphertexts maintain strong privacy when considering nonces and associated data. They mention the need for robustness, although they only consider robustness for honestly generated keys, which is implied by the typical AEAD security notion as shown in [17]. Finally, Jaeger and Tyagi [22] consider multi-user simulation-based security definitions for various standard symmetric definitions where keys can be adaptively compromised.

2 Preliminaries

We follow the notational conventions used in [19]. We fix some alphabet Σ, e.g. $\Sigma = \{0,1\}$. For any $x \in \Sigma^*$, let $|x|$ denote its length. We write $x \leftarrow\!\!\$\, X$ to denote uniformly sampling from a finite set X. We write $X \| Y$ to denote

concatenation of two strings. For a string X of n bits, we will write $X[i,\ldots,j]$ for $1 \leq i < j \leq n$ to mean the substring of X beginning at index i and ending at index j. For notational simplicity, we assume that one can unambiguously parse $Z = X\|Y$ into its two parts, even for strings of varying length. For strings $X, Y \in \{0,1\}^*$ we write $X \oplus Y$ to denote taking the XOR of $X[1,\ldots,\min(|X|,|Y|)] \oplus Y[1,\ldots,\min(|X|,|Y|)]$. For some table of values where y_i is stored at key x_i, denoted as $\mathsf{T}[x_i] \leftarrow y_i$, for a set of keys $X = \{x_i : 1 \leq i \leq \kappa\}$ for some integer κ, $\mathsf{T}[X]$ denotes the set $\{\mathsf{T}[x_i] : x_i \in X\}$. We denote a vector of elements as $[\cdot]$. We denote the value stored at index i in vector $\mathbb{K}$ as $\mathbb{K}[i]$. For vector $\mathbb{K}$, we denote $x \in \mathbb{K}$ as x is an element of $\mathbb{K}$ and $|\mathbb{K}|$ as the number of elements in $\mathbb{K}$. We also denote $\mathbb{K}.\mathsf{add}(x)$ to mean adding x to the end of vector $\mathbb{K}$. We denote $[n]_\ell$ as the ℓ-bit representation of the integer n.

We use code-based games [11] to formalize security notions. Variables' types should be clear from context and are modeled as random variables in the probability distribution defined by the random coins used in execution. $\Pr[\,\mathrm{G} \Rightarrow y\,]$ denotes (over the random coins of G) that the game G outputs the value y. For a scheme S, we will sometimes use "a G_S adversary" to describe an adversary in the game G instantiated with the scheme S. For an adversary $\mathcal{A}$, $\mathrm{G}_S^{\mathcal{A}}$ denotes the game G instantiated with the scheme S and specific adversary $\mathcal{A}$. $\Pr[\,\mathrm{G}_S^{\mathcal{A}} \Rightarrow \mathsf{out}\,]$ denotes the probability that game G instantiated with scheme S and adversary $\mathcal{A}$ outputs out.

Authenticated encryption. An authenticated encryption with associated data (AEAD) scheme $\mathrm{AEAD} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ consists of a triple of algorithms. Associated to any scheme AEAD is a key space $\mathcal{K} \subseteq \Sigma^*$, nonce space $\mathcal{N} \subseteq \Sigma^*$, header space $\mathcal{A} \subseteq \Sigma^*$, message space $\mathcal{M} \subseteq \Sigma^*$, and ciphertext space $\mathcal{C} \subseteq \Sigma^*$. The randomized key generation algorithm Kg outputs a secret key $K \in \mathcal{K}$. Encryption Enc is deterministic and takes as input a 4-tuple $(K, N, AD, M) \in (\Sigma^*)^4$ and outputs ciphertext C or a distinguished error symbol $\perp$. We require that $\mathsf{Enc}(K, N, AD, M) \neq \perp$ if $(K, N, AD, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$. Decryption Dec is deterministic and takes as input a tuple $(K, N, AD, C) \in (\Sigma^*)^4$ and outputs value M or $\perp$. An AEAD scheme is correct if for any $(K, N, AD, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ it holds that $\mathsf{Dec}(K, N, AD, \mathsf{Enc}(K, N, AD, M)) = M$.

The security notion we consider for AEAD schemes is nonce-based real-or-random security under chosen-ciphertext attack [29]. We generalize this to a multi-user setting [12] where an adversary can interact with multiple instances of the AEAD scheme, which we call MU-nAE$_{\mathrm{AEAD}}$. The game pseudocode is presented in Figure 2. We restrict our attention to nonce-respecting adversaries, meaning they never query the same nonce twice to Enc for the same key identifier id, and they only query a key identifier $\mathsf{id} < i$ to Enc and Dec. The MU-nAE$_{\mathrm{AEAD}}$ advantage of an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\text{mu-nae}}_{\mathrm{AEAD}}(\mathcal{A}) = |\Pr[\,\text{MU-nAE1}^{\mathcal{A}}_{\mathrm{AEAD}} \Rightarrow 1\,] - \Pr[\,\text{MU-nAE0}^{\mathcal{A}}_{\mathrm{AEAD}} \Rightarrow 1\,]|.$$

Full robustness. We use the full robustness notion for AEAD schemes from Farshim et al. [17], but adapted to the nonce-based setting. Albertini et al. [2]

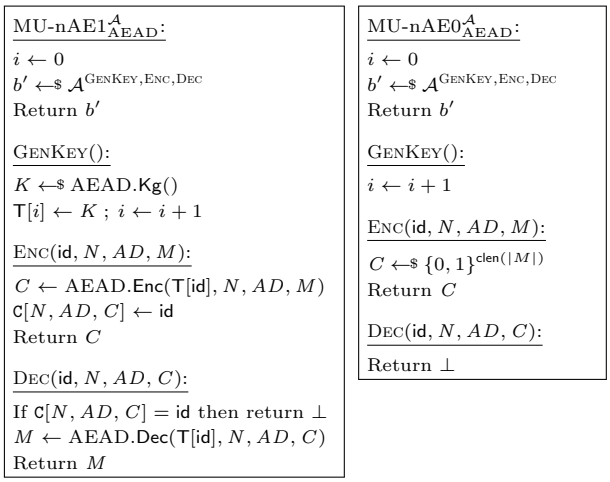

Fig. 2: Games MU-nAE1 and MU-nAE0 are used for MU-nAE$_{\mathrm{AEAD}}$, or multi-user real-or-random security, for scheme AEAD = (Kg, Enc, Dec).

also provide a FROB notion for nonce-based AEAD, with slightly different syntax — our formulation is equivalent to theirs. Roughly, FROB security tasks an adversary with providing two keys and a ciphertext such that both keys successfully decrypt the ciphertext. We define the game in Figure 3.

The FROB$_{\mathrm{AEAD}}$ advantage of an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\mathrm{frob}}_{\mathrm{AEAD}}(\mathcal{A}) = \Pr\left[\,\mathrm{FROB}^{\mathcal{A}}_{\mathrm{AEAD}} \Rightarrow 1\,\right].$$

Pseudo-random functions. We use a multi-user variant of the traditional pseudo-random function (PRF) definition (q.v., [7]) for two functions, where the adversary is given access to oracles for both functions. We define the games in Figure 3. The game gives the adversary an additional GenKey oracle that allows it to generate multiple keys. The MU-PRF$_{\mathsf{F}}$ advantage of an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\text{mu-prf}}_{\mathsf{F}_0,\mathsf{F}_1}(\mathcal{A}) = |\Pr\left[\,\mathrm{REAL}^{\mathcal{A}}_{\mathsf{F}_0,\mathsf{F}_1} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{IDEAL}^{\mathcal{A}} \Rightarrow 1\,\right]|.$$

Collision resistance. The collision resistance (CR) game for function $\mathsf{F} = \{0,1\}^{\kappa} \times \{0,1\}^{\ell} \to \{0,1\}^{n}$ measures the ability of an adversary to find two key-value pairs such that the evaluation of F on these inputs evaluates to the same output. More formally, the CR$_{\mathsf{F}}$ advantage of an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{F}}(\mathcal{A}) = \Pr\left[\,K_0, x_0, K_1, x_1 \leftarrow \mathcal{A} : (K_0, x_0) \neq (K_1, x_1) \wedge \mathsf{F}(K_0, x_0) = \mathsf{F}(K_1, x_1)\,\right].$$

Pre-image resistance. The pre-image resistance game for function $\mathsf{F} = \{0,1\}^{\kappa} \times \{0,1\}^{\ell} \to \{0,1\}^{n}$ measures the ability of an adversary to find the pre-image of a

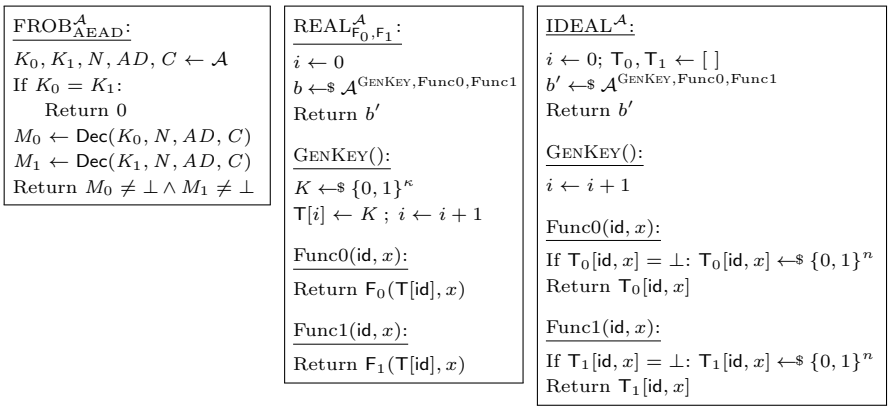

Fig. 3: **(Left)** Game $\mathrm{FROB}_{\mathrm{AEAD}}$ is the full robustness security notion for scheme $\mathrm{AEAD} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$. **(Center/Right)** Games REAL and IDEAL are used for MU-PRF$_{\mathsf{F}_0,\mathsf{F}_1}$, or multi-user PRF security for functions $\mathsf{F}_0 = \{0,1\}^{\kappa} \times \{0,1\}^{\ell_0} \rightarrow \{0,1\}^{n_0}$ and $\mathsf{F}_1 = \{0,1\}^{\kappa} \times \{0,1\}^{\ell_0} \rightarrow \{0,1\}^{n_0}$.

random range point for F. More formally, the $\mathrm{PRE}_{\mathsf{F}}$ advantage of an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\mathrm{pre}}_{\mathsf{F}}(\mathcal{A}) = \Pr\left[\, y \leftarrow\$ \{0,1\}^n; K, x \leftarrow \mathcal{A}(y) : \mathsf{F}(K, x) = y \,\right].$$

3 Defining AEAD with Key Identification

We start by formalizing the notion of AEAD with key identification (AEAD-KI). AEAD-KI extends AEAD schemes to the setting where a recipient stores multiple keys and must therefore choose which key to use for decryption. At a high level, our formalization extends prior ones on AEAD in the following ways:

- We add a notion of key labels, which are potentially public, application-defined strings associated to secret keys. For notational simplicity, we will redefine a key to be a label, secret key pair.
- Decryption takes as input a vector of keys, instead of a single key. Decryption must determine both which key to use, and the corresponding plaintext. We model the keys used by decryption as a vector, instead of a set, to preserve information about the order.
- Ciphertexts may include a key identification tag, to assist decryption. We will explore a variety of ways to construct key identification tags, each with different security and performance profiles.

These changes to our conceptualization of syntax and semantics of AEAD necessarily require revisiting security as well. Later in this section we propose a new simulator-based all-in-one security definition that captures confidentiality and integrity for AEAD-KI schemes. Furthermore, we will see that the shift to

AEAD-KI introduces a number of subtleties related to ciphertexts potentially being decryptable under more than one key. We will therefore provide new key robustness (also called key commitment) notions for AEAD-KI.

Syntax and semantics. An AEAD-KI scheme is a triple of algorithms combined with a a key space $\mathcal{K} \subseteq \Sigma^*$, key label space $\mathcal{L} \subseteq \Sigma^*$, nonce space $\mathcal{N} \subseteq \Sigma^*$, associated data space $\mathcal{A} \subseteq \Sigma^*$, message space $\mathcal{M} \subseteq \Sigma^*$, ciphertext space $\mathcal{C} \subseteq \Sigma^*$, and key tag space $\mathcal{T} \subseteq \Sigma^*$. We will often leave the spaces implicit and clear from context. Thus we write that an AEAD-KI scheme $\mathsf{AEKI} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms:

- $K \leftarrow\!\!\$\ \mathsf{AEKI.Kg}(\mathsf{kid})$
  The randomized key generation algorithm takes as input the key label kid to use for the generated secret key. The key label operates as metadata for the secret key. Key generation outputs a key $K \in \mathcal{L} \times \mathcal{K}$, which is a pair composed of the key label and the secret encryption/decryption key. While the encryption key must be kept secret, the key label can be public.
- $(T_k, C) \leftarrow \mathsf{AEKI.Enc}(K, N, AD, M)$
  The nonce-based deterministic encryption algorithm takes as input tuple $(K, N, AD, M) \in (\Sigma^*)^4$ and outputs pair $(T_k, C) \in \mathcal{T} \times \mathcal{C}$ or a distinguished error symbol $\perp$. Notice that encryption returns, in addition to the encrypted plaintext, a bit string T_k, which can be empty. We will refer to this as the key tag, as we discuss more below. Both the key tag and the encrypted plaintext form the ciphertext. We require that if $(K, N, AD, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ then $\mathsf{Enc}(K, N, AD, M) \neq \perp$.
- $(K, M) \leftarrow \mathsf{AEKI.Dec}(\mathbb{K}, N, AD, T_k, C)$
  The decryption algorithm's input is $(\mathbb{K}, N, AD, T_k, C) \in \mathcal{K}^* \times (\Sigma^*)^4$ and its output is $(K, M) \in \mathcal{K} \times \mathcal{M}$ or $\perp$. Decryption is deterministic. Notice that instead of a single key, decryption takes as input a vector of keys $\mathbb{K}$, and we denote vectors of length one or greater by $\mathcal{K}^*$. Furthermore, in addition to the plaintext, decryption returns the corresponding key that produced the plaintext. If no key can decrypt, then the error symbol $\perp$ is returned.

Correctness. When extending AEAD to allow for multiple keys, a meaningful definition of correctness becomes more complex. We expect that when encrypting with a key K to produce ciphertext C, and then decrypting C with any vector $\mathbb{K}$ that includes K, the original plaintext should be recovered. However, this cannot for practical schemes be guaranteed absolutely, since there may exist another key that successfully decrypts the ciphertext. Indeed, the correct outcome that we expect of a scheme that allows decryption to accept multiple keys now becomes more like a security property of key robustness, where we have computational guarantees that decryption succeeds for a single key. We therefore provide here a simpler, absolute correctness definition and later focus on capturing the behavior we want from AEAD-KI through key robustness, which we cover below.

Definition 1. *An AEAD-KI scheme is correct if the following hold:*

(1) *For any* (K, N, AD, M) *it holds that* $\Pr[\,(K', M') = (K, M)\,] = 1$ *where* $(K', M') \leftarrow \mathsf{Dec}([K], N, AD, \mathsf{Enc}(K, N, AD, M))$ *and the probability is over the coins used by encryption;*

(2) *For any* $(\mathbb{K}, N, AD, T_k, C)$ *and* $(K, M) \leftarrow \mathsf{Dec}(\mathbb{K}, N, AD, T_k, C)$ *it must be that either* $(K, M) = \perp$ *or* $K \in \mathbb{K}$*; and*

(3) *For any* $\mathbb{K}, \mathbb{K}'$ *and any* (N, AD, T_k, C)*, let* $(K, M) \leftarrow \mathsf{Dec}(\mathbb{K}, N, AD, T_k, C)$ *and* $(K', M') \leftarrow \mathsf{Dec}(\mathbb{K}', N, AD, T_k, C)$*. If* $(K, M) \neq \perp$ *and* $K \in \mathbb{K}'$*, then* $(K', M') \neq \perp$*.*

The first condition lifts traditional perfect correctness for AEAD to the syntax of AEAD-KI for decryption with a single key. The second condition additionally asks that decryption only ever output a key that was in the key vector. The third correctness condition roughly requires that if Dec outputs some key K for a key set $\mathbb{K}$, any other $\mathbb{K}'$ containing K must decrypt to non-$\perp$. (Note we do not require decryption with $\mathbb{K}'$ outputs K; this property is guaranteed by our key robustness security notion, which we discuss next.) For all schemes we consider, correctness is easily established via inspection of their decryption algorithm; we therefore will omit explicit analysis.

3.1 Key robustness

As mentioned, in the AEAD-KI setting, key robustness is partly about correctness: we expect that the key used to encrypt a plaintext should be the only one to correctly decrypt the resulting ciphertext. However, when decryption allows for multiple keys—some of which may be adversarially-chosen—this property cannot be satisfied without some form of key robustness. Briefly, key robustness guarantees that only a single key can be used to decrypt a given ciphertext.

Farshim et al. [16] first defined several key robustness notions for AEAD schemes. Their strongest notion is called full robustness (FROB), and requires an adversary to discover two keys that each successfully decrypt an adversarially chosen nonce, associated data, and ciphertext (see Section 2). Bellare and Hoang [9] recently introduced even stronger notions, such as their CMT3, which require commitment to not just the key but also nonces and associated data. For simplicity we stick with adapting the FROB notion. Analogous adaptations can be made to lift CMT3 to the key identification setting, but some schemes would require modification to meet them (e.g., including nonce and associated data in key check value computations).

We define KI-FROB security for an AEAD-KI scheme via the game shown in Figure 4. It requires an adversary to find two key vectors and a nonce, associated data, and ciphertext. Decryption is run with each key vector, and the adversary wins should both decryptions succeed and the returned secret key, message pairs are distinct. Note here that we are focused on the secret key, not key including key label, thereby explicitly excluding as a win having distinct key labels. This is to allow schemes that use multiple key labels for the same key. The $\text{KI-FROB}_{\mathsf{AEKI}}$ advantage of an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\text{ki-frob}}_{\mathsf{AEKI}}(\mathcal{A}) = \Pr\left[\,\text{KI-FROB}^{\mathcal{A}}_{\mathsf{AEKI}} \Rightarrow 1\,\right].$$

$\underline{\text{KI-FROB}^{\mathcal{A}}_{\mathsf{AEKI}}}$:

$\mathbb{K}_0, \mathbb{K}_1, N, AD, T_k, C \xleftarrow{\$} \mathcal{A}$
$(K_0, M_0) \leftarrow \mathsf{Dec}(\mathbb{K}_0, N, AD, T_k, C)$
$(K_1, M_1) \leftarrow \mathsf{Dec}(\mathbb{K}_1, N, AD, T_k, C)$
$(\mathsf{kid}_0, K_0^*) \leftarrow K_0$; $(\mathsf{kid}_1, K_1^*) \leftarrow K_1$
If $(K_0, M_0) \neq \perp \wedge (K_1, M_1) \neq \perp \wedge K_0^* \neq K_1^*$:
  Return 1
Return 0

Fig. 4: Game $\text{KI-FROB}_{\mathsf{AEKI}}$ represents full robustness for AEAD-KI scheme $\mathsf{AEKI} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$.

Finally, note an important property of our KI-FROB security notion: any scheme meeting it is in some sense agnostic to the *ordering* of keys in the key vector input to decryption. If two different orderings of the same key vector caused different keys to be output, these two key vectors would give a KI-FROB win. (Note that correctness condition (3) above implies that two different orderings of the same key vector must both either output $\perp$ or both non-$\perp$.) Looking ahead, it also means our KI-nAE definition need not account for distinguishing attacks caused by the order of keys in the key vectors; they are ruled out for any KI-FROB scheme.

3.2 All-in-one confidentiality and integrity

Just as for an AEAD scheme, we expect an AEAD-KI scheme to maintain confidentiality and integrity. Towards formalizing what this means, one starting point is existing indistinguishability style security definitions for AEAD (e.g., [29]). However, an important modeling question is how to handle key vectors during decryption. This suggests we should start instead with a multi-user style AEAD security notion [12] which allows the adversary to request generation of many keys and obtain encryption of plaintexts under keys of their choice.

To model key anonymity, we may additionally require that adversaries not be able to distinguish between encryptions under different honest keys. To model chosen-ciphertext attacks and, in particular, ciphertext integrity, we face additional choices about how much control to give adversaries over key vectors during decryption. One option would be to allow adversaries to choose the key vector but only allow honestly generated keys to be added to the vector. Unfortunately, this would not capture attacks in which a malicious key is somehow inserted into a recipient's key vector, a scenario that arises in practice. For instance, Albertini et al. [2] describe a vulnerability arising from such a scenario in the context of key rotation within key management services.

We therefore opt for a stronger notion of security for which adversaries can insert malicious keys into key vectors. This renders more complex how the security game should handle decryption oracles, because we need to somehow demarcate between decryption queries that correspond to ciphertext forgeries and ones that

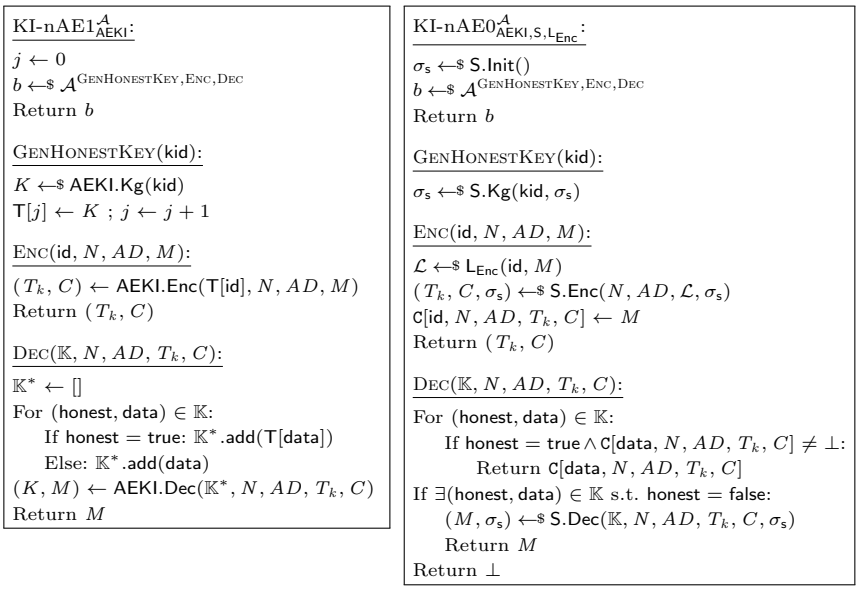

Fig. 5: Game KI-nAE$_{\mathsf{AEKI}}$ is the all-in-one security notion for AEAD-KI schemes.

should not: an adversary can always generate ciphertexts that decrypt under an adversary-chosen key.

To handle these subtleties, we use a simulation-based approach and an all-in-one confidentiality and ciphertext integrity notion. The pseudocode games for the resulting KI-nAE$_{\mathsf{AEKI}}$ security notion are shown in Figure 5. KI-nAE$_{\mathsf{AEKI}}$ is parameterized by the simulator, a stateful tuple of algorithms $\mathsf{S} = (\mathsf{Init}, \mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$. The adversary is given access to an honest key generation oracle GenHonestKey, an encryption oracle Enc, and a decryption oracle Dec. Dec accepts as input a key vector $\mathbb{K}$ as well as ciphertext tuple (N, AD, T_k, C). The key vector $\mathbb{K}$ is composed of tuples $(\mathsf{honest}, \mathsf{data})$, where $\mathsf{honest} = \mathsf{true}$ indicates that the key is honestly generated and data is the game-generated key identifier for the key. If $\mathsf{honest} = \mathsf{false}$, then the key is malicious and data is itself the key.

The game KI-nAE1 models interactions with the real scheme, and therefore calls the relevant AEKI algorithm to answer oracle queries. The ideal game KI-nAE0 instead uses simulator S to generate the oracle outputs for the adversary. Encryption provides leakage to the simulator. The encryption leakage algorithm $\mathsf{L}_{\mathsf{Enc}}(\mathsf{id}, M)$ takes as input the game-generated key identifier and the plaintext message and outputs the encryption leakage. We specify two concrete leakage functions, $\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}$ and $\mathsf{L}^{\mathsf{anon}}_{\mathsf{Enc}}$. The non-key anonymous algorithm $\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}$ returns as leakage both the game-generated key identifier and the size of the plaintext, while $\mathsf{L}^{\mathsf{anon}}_{\mathsf{Enc}}$ only returns the size of the plaintext. In the latter case, this of course means that the simulator will have no knowledge of which honest key the adversary chose for encryption.

The decryption oracle in KI-nAE0 works in two parts. In the first part, the table of previous ENC outputs is scanned for each honest key. If the queried ciphertext is in the table, the message is returned. Otherwise, if there are malicious keys in $\mathbb{K}$, the simulator is given its state, the ciphertext tuple, and $\mathbb{K}$.

It may not be immediately obvious why this DEC is the "right" one. The main advantage in defining DEC as we have is that for honest keys this decryption oracle ensures our definition implies a variant of ciphertext integrity for AEAD-KI: no matter the simulator's behavior, crafting a new valid ciphertext for an honest key automatically gives a distinguisher between real and ideal games.

We have two versions of KI-nAE, one key anonymous and one not. For a simulator S, the more general (non-anonymous) KI-nAE$_{\mathsf{AEKI}}$ advantage of an adversary $\mathcal{A}$ with respect to S is defined as

$$\mathbf{Adv}^{\text{ki-nae}}_{\mathsf{AEKI},\mathsf{S}}(\mathcal{A}) = \left| \Pr\left[\, \text{KI-nAE1}^{\mathcal{A}}_{\mathsf{AEKI}} \Rightarrow 1 \,\right] - \Pr\left[\, \text{KI-nAE0}^{\mathcal{A}}_{\mathsf{AEKI},\mathsf{S},\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}} \Rightarrow 1 \,\right] \right|.$$

Meanwhile, the key anonymous KI-nAE-KA$_{\mathsf{AEKI}}$ advantage of an adversary $\mathcal{A}$ with respect to S is defined as

$$\mathbf{Adv}^{\text{ki-nae-anon}}_{\mathsf{AEKI},\mathsf{S}}(\mathcal{A}) = \left| \Pr\left[\, \text{KI-nAE1}^{\mathcal{A}}_{\mathsf{AEKI}} \Rightarrow 1 \,\right] - \Pr\left[\, \text{KI-nAE0}^{\mathcal{A}}_{\mathsf{AEKI},\mathsf{S},\mathsf{L}^{\mathsf{anon}}_{\mathsf{Enc}}} \Rightarrow 1 \,\right] \right|.$$

We further note that while we chose to base our definitions on those for nonce-based AEAD from [12, 29], we can also adapt these definitions to other settings, such as those for randomized symmetric encryption and the nonce-hiding framework for the AE2 definitions proposed by Bellare et al. [10].

Malicious keys and ciphertexts. As discussed before, we opt to capture the adversary providing both malicious keys and malicious ciphertexts since this models real-world settings. However, we must explicitly give the simulator all malicious keys because the adversary knows these keys as well and could otherwise trivially distinguish. This makes it difficult for KI-nAE to capture key robustness notions like KI-FROB. We therefore opt for separate key robustness notions, mirroring the AEAD setting. We believe it is possible to give an all-in-one definition that also implies KI-FROB; we leave this difficult, but interesting, modelling question to future work.

4 Key Labels

One simple technique for key identification is assigning to each key a static label and then prepending the key's label to every ciphertext it produces. In practice, labels typically are randomly-generated strings, URLs indicating where to fetch the key, or even user identifiers. This approach is widely used by both key management services (KMS), such as the Amazon Web Services (AWS) KMS [5], Microsoft Azure Key Vault [25], and Oracle Key Vault [26]; as well as cryptography libraries, such as Google's Tink [32]. Relatedly, the popular cryptographic library Libsodium [24] also recommends using a key label as a way to add robustness to AEAD schemes.

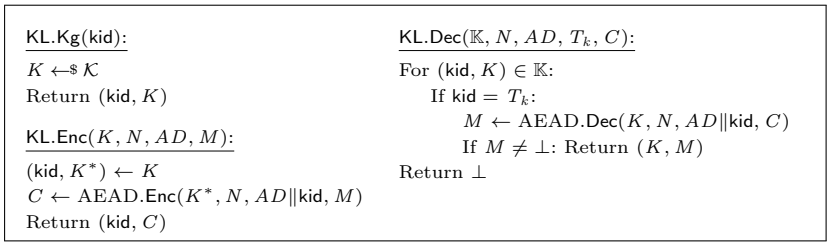

Fig. 6: A typical key label scheme KL which is parameterized by an AEAD scheme.

While key labels appear efficient and straightforward, they have not been formally analyzed. For instance, key labels are often used with non-FROB AEAD schemes, such as AES-GCM and ChaCha20-Poly1305. We will see in our analysis that key labels do not automatically produce a KI-FROB AEAD-KI scheme.

We generalize the key labels construction as KL[AEAD] and provide the pseudocode in Figure 6. The scheme is parameterized by an AEAD scheme, used for encryption and decryption. To simplify notation, we also refer to the scheme as KL when the specific AEAD scheme used can be arbitrary or is obvious from context. Key generation allows the caller to specify the key label kid, which is then stored as part of the key. We model the label as input to generalize label creation out-of-band and to enable adversarial inputs. Meanwhile, encryption simply uses kid as the key tag. Decryption iterates through $\mathbb{K}$ to find the first key with an identifier matching the key tag that successfully decrypts the ciphertext.

Analyzing robustness. Utilizing a key label at first seems like a trivial and practical way to add robustness to any AEAD-KI scheme. A ciphertext with some key identifier can only be decrypted by the corresponding key. However, this method fails when multiple keys can have the same label and a non-FROB AEAD scheme is chosen. Notice, for instance, that KL does not enforce uniqueness of labels. An adversary could choose two AEAD keys so that they have the same label, compute an AEAD key multi-collision ciphertext [23] for these keys, and attach the label to the ciphertext as the key tag. Decryption will then proceed successfully for both keys because their labels match the key tag of the ciphertext.

In the following theorem, we show that an FROB AEAD is both sufficient and necessary for KL to be KI-FROB.

Theorem 1. *Let $\mathcal{A}$ be a* KI-FROB *adversary for scheme* KL[AEAD]*. Then we give* FROB *adversary $\mathcal{B}$ for* AEAD *such that*

$$\mathbf{Adv}^{\text{ki-frob}}_{\mathsf{KL}[\text{AEAD}]}(\mathcal{A}) \leq \mathbf{Adv}^{\text{frob}}_{\text{AEAD}}(\mathcal{B}).$$

Furthermore, let $\mathcal{C}$ be an FROB *adversary for* AEAD*. Then we give a* KI-FROB *adversary $\mathcal{D}$ for* KL[AEAD] *such that*

$$\mathbf{Adv}^{\text{frob}}_{\text{AEAD}}(\mathcal{C}) \leq \mathbf{Adv}^{\text{ki-frob}}_{\mathsf{KL}[\text{AEAD}]}(\mathcal{D}).$$

$\mathcal{B}$ runs in time that of $\mathcal{A}$ and $\mathcal{D}$ runs in time that of $\mathcal{C}$.

Proof. We first construct FROB adversary $\mathcal{B}$ against AEAD as follows. $\mathcal{B}$ runs $\mathcal{A}$, which returns $\mathbb{K}_0, \mathbb{K}_1, N, AD, T_k, C$. Let $\mathsf{Dec}(\mathbb{K}_0, N, AD, T_k, C)$ and $\mathsf{Dec}(\mathbb{K}_1, N, AD, T_k, C)$ return (K_0, M_0) and (K_1, M_1), respectively, where $K_0 = (\mathsf{kid}_0, K_0^*)$ and $K_1 = (\mathsf{kid}_1, K_1^*)$. Note that $\mathcal{A}$ can only win the FROB game if $(K_0, M_0), (K_1, M_1) \neq \perp$ and $K_0^* \neq K_1^*$, so let this be the case. $\mathcal{B}$ can return K_0^*, K_1^*, N, AD, C, where again both keys successfully decrypt N, AD, C. Thus, $\mathcal{B}$ wins game FROB for AEAD when $\mathcal{A}$ wins game KI-FROB for KL[AEAD].

We now construct KI-FROB adversary $\mathcal{D}$ against KL as follows. $\mathcal{D}$ runs $\mathcal{C}$, which returns K_0, K_1, N, AD, C. Then $\mathcal{D}$ creates KL key vectors $\mathbb{K}_0 \leftarrow [(0^n, K_0)]$ and $\mathbb{K}_1 \leftarrow [(0^n, K_1)]$ and finally returns $\mathbb{K}_0, \mathbb{K}_1, N, AD, 0^n, C$. Since both K_0, K_1 have the same identifier, which matches the given key tag, the scheme KL will try both keys when decrypting and successfully decrypt. Thus, we have that $\mathcal{D}$ wins game KI-FROB for KL[AEAD] when $\mathcal{C}$ wins game FROB for AEAD. □

Analyzing KI-nAE. We now show that KL[AEAD] is KI-nAE secure for leakage algorithm $\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}$ when AEAD is both FROB and MU-nAE secure. Notably, encryption cannot be key anonymous: the label is static across calls to Enc and the simulator can only simulate this by knowing which key was queried for encryption. We provide the theorem statement and proof sketch below. The full proof is provided in the full version of this work.

Theorem 2. *Using $\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}$, let $\mathcal{A}$ be a* KI-nAE *adversary making at most q queries to its oracles and querying at most m malicious keys. Then we give KI-nAE simulator S and adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{ki-nae}}_{\mathsf{KL}[\text{AEAD}],\mathsf{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{frob}}_{\text{AEAD}}(\mathcal{B}) + \mathbf{Adv}^{\text{mu-nae}}_{\text{AEAD}}(\mathcal{C}) + mq/|\mathcal{K}|.$$

Adversaries $\mathcal{B}, \mathcal{C}$ run in time that of $\mathcal{A}$ with an $\mathcal{O}(q)$ overhead and simulator S runs in time $\mathcal{O}(mq)$.

Proof sketch: The KI-nAE simulator can simulate Enc queries by keeping track of the label for each key and returning it as the key tag along with a random string of the correct length for the encrypted plaintext. For Dec queries, the simulator can iterate through the list of key data given in the key vector $\mathbb{K}$ and check for malicious keys, for which it is directly given the secret key and can decrypt itself for any that have a key label matching the key tag. If there are no malicious keys or none that correctly decrypt, then the simulator returns $\perp$.

We bound the advantage of $\mathcal{A}$ with a sequence of game hops. We first transition to a game in which Dec keeps iterating through $\mathbb{K}$ if $\mathcal{A}$ provides a malicious key in $\mathbb{K}$ that was honestly generated by a call to GenHonestKey. We bound the ability of $\mathcal{A}$ to distinguish between these games by $mq/|\mathcal{K}|$, since there are at most m malicious keys and at most q honest keys and $\mathcal{A}$ can only at best guess one of the honest keys. We next transition to a game in which Dec skips any malicious key that can decrypt a ciphertext output from a call to Enc. We bound the ability of $\mathcal{A}$ to distinguish between these games by the FROB security

of AEAD. Finally, we transition to a game in which Enc generates a random string as the encrypted plaintext and Dec skips honest keys in the key vector if they were not used to produce the queried ciphertext from a call to Enc. We bound the ability of the adversary to distinguish between these games by the MU-nAE security of AEAD. Since this last game guarantees that no malicious key can decrypt an honestly generated ciphertext and that no honest key can decrypt a malicious ciphertext, iterating through $\mathbb{K}$ in order in this game is identical to iterating through the honest keys first and then the malicious keys, proving our claim. □

Using unique random identifiers. Thus far, we have relied on an FROB AEAD scheme, due to the fact that KL allows duplicate key identifiers, which follows practice (e.g., the Tink library). One might instead suggest somehow enforcing uniqueness of key labels at the "application layer", and using a non-FROB AEAD. We argue below that this approach either fails to meet natural security goals or greatly increases the complexity of the application layer; thus, FROB AEADs are superior as the "base" AEAD for an KL-like construction.

To study this question, we first need to express application-level enforcement of unique key labels in our AEKI formalism. A simple way to do this is to have Dec check the identifiers for all keys in $\mathbb{K}$ and output $\perp$ if two different keys have the same identifier. This approach does not meet condition (3) of our correctness notion above. Any key vector with repeated labels will cause decryption to fail, even if it contains the correct key.

More subtly, this approach also fails to provide KI-FROB for the AEKI scheme. To see why, note that because decryption is stateless, uniqueness cannot be checked across different invocations. Thus, if the underlying AEAD is not FROB, an adversary can choose two keys, produce a key multi-collision ciphertext, assign both keys the same label that matches the ciphertext's key tag, and then put the keys in separate key vectors. Decryption will succeed for both key vectors, even though their keys have non-unique labels.

Preventing this attack and providing KI-FROB for the AEKI scheme requires stateful decryption: namely, the application must track all key identifier-secret key pairs seen across *all* decryption operations. This seems difficult to implement correctly and efficiently, and is certain to increase the complexity of the application. Thus, we believe it is better to use FROB AEAD to cryptographically guarantee KI-FROB security of AEKI.

Analyzing trial decryption. A special case of the key labels scheme is brute-force trial decryption, which we refer to as TD[AEAD]. This scheme simply assigns the empty string ε as the key label for all keys, meaning there are no key identifiers, and decryption must always trial decrypt for all keys in the key vector. The Tink library, for instance, allows keys to also have "raw" labels, which indicate they have no identifier.

Notably, the multi-user Shadowsocks protocol we describe in the introduction falls into this category. The attack described by Len et al. [23] is made possible by the fact that the Shadowsocks protocol uses a non-FROB AEAD scheme. The

benefit of our KI-FROB definition is that it demonstrates that such a scheme is insecure when not using a FROB AEAD scheme.

While trial decryption still requires the use of an FROB AEAD scheme, it is key anonymous. In the below theorem, we show that trial decryption meets our stronger key anonymous encryption leakage model. We provide the full proof in the full version of this work.

Theorem 3. *Using* $\mathsf{L}^{\mathsf{anon}}_{\mathsf{Enc}}$*, let* $\mathcal{A}$ *be a* KI-nAE-KA *adversary making at most* q *queries to its oracles and querying at most* m *malicious keys. Then we give* KI-nAE-KA *simulator* S *and adversaries* $\mathcal{B}, \mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{ki-nae-anon}}_{\mathsf{TD}[\text{AEAD}],\mathsf{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{frob}}_{\text{AEAD}}(\mathcal{B}) + \mathbf{Adv}^{\text{mu-nae}}_{\text{AEAD}}(\mathcal{C}) + mq/|\mathcal{K}|.$$

Adversaries $\mathcal{B}, \mathcal{C}$ *run in time that of* $\mathcal{A}$ *with an* $\mathcal{O}(q)$ *overhead and simulator* S *runs in time* $\mathcal{O}(mq)$*.*

Proof sketch: The KI-nAE simulator can simulate Enc queries by returning a random string of the correct length for the encrypted plaintext. For Dec queries, the simulator can iterate through the list of key data given in the key vector $\mathbb{K}$ and check for malicious keys, for which it is directly given the secret key and can decrypt itself. If there are no malicious keys or none that correctly decrypt, then the simulator returns $\perp$.

We bound the advantage of $\mathcal{A}$ with a sequence of game hops. We first transition to a game in which Dec keeps iterating through $\mathbb{K}$ if $\mathcal{A}$ provides a malicious key in $\mathbb{K}$ that was honestly generated by a call to GenHonestKey. We bound the ability of $\mathcal{A}$ to distinguish between these games by $mq/|\mathcal{K}|$, since there are at most m malicious keys and at most q honest keys and $\mathcal{A}$ can only at best guess one of the honest keys. We next transition to a game in which Dec skips any malicious key that can decrypt a ciphertext output from a call to Enc. We bound the ability of $\mathcal{A}$ to distinguish between these games by the FROB security of AEAD. Finally, we transition to a game in which Enc generates a random string as the encrypted plaintext and Dec skips honest keys in the key vector if they were not used to produce the queried ciphertext from a call to Enc. We bound the ability of the adversary to distinguish between these games by the MU-nAE security of AEAD. Since this last game guarantees that no malicious key can decrypt an honestly generated ciphertext and that no honest key can decrypt a malicious ciphertext, iterating through $\mathbb{K}$ in order in this game is identical to iterating through the honest keys first and then the malicious keys, proving our claim. □

While key labels are a simple and commonly used approach in practice for identifying keys, here we have shown that a formal analysis surfaces subtleties. In particular, key labels do not provide key anonymity, unless all keys have the same label as in the less efficient trial decryption-based scheme. Moreover, a key label approach must rely on the underlying AEAD scheme being FROB. In the next section, we explore a different tactic called static key identifiers, which computes the identifier from the key itself.

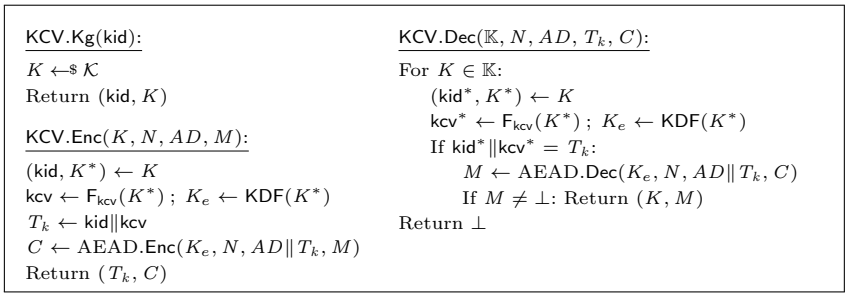

Fig. 7: A key check value scheme KCV parameterized by encryption scheme AEAD = (Kg, Enc, Dec) with associated key space $\mathcal{K} = \{0,1\}^{\mathsf{k}}$, key check value function $\mathsf{F}_{\mathsf{kcv}} : \{0,1\}^{\kappa} \to \{0,1\}^{\mathsf{n}}$, and encryption key derivation function $\mathsf{KDF} : \{0,1\}^{\kappa} \to \{0,1\}^{\mathsf{k}}$.

5 Static Key Identifiers

In this section we describe a class of AEAD-KI that uses what we call *static key identifiers*. This technique computes a static identifier from the key that along with the key label is used as the key tag. In practice, this static identifier is often referred to as a *key check value*, also known as a key checksum value. We formalize this approach as the AEAD-KI scheme $\mathsf{KCV}[\mathsf{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}]$, shown in Figure 7. The scheme has key space $\{0,1\}^{\kappa}$ and is parameterized by AEAD scheme AEAD = (Kg, Enc, Dec) with associated key space $\mathcal{K} = \{0,1\}^{\mathsf{k}}$, key check value function $\mathsf{F}_{\mathsf{kcv}} : \{0,1\}^{\kappa} \to \{0,1\}^{\mathsf{n}}$, and encryption key derivation function $\mathsf{KDF} : \{0,1\}^{\kappa} \to \{0,1\}^{\mathsf{k}}$. For simplicity, we will sometimes refer to the scheme as KCV when the parameters are obvious.

Key generation generates a secret key and attaches the input label kid as part of the AEAD-KI key. For static identifier schemes, the key tag is composed of both the key label and the key check value. This models what happens in practice, as schemes may use the key label as a way to locate a key or set of keys and then use the key check value as a commitment or integrity check of the key. For instance, AWS uses the Amazon Resource Name (ARN) as a URL for looking up keys, while an extra commitment string verifies this is the correct key [4].

Encryption derives the key check value using the function $\mathsf{F}_{\mathsf{kcv}}$ and separately derives the AEAD secret key using the function KDF. Encryption adds the key tag to the authentication scope of AEAD by appending it to the authenticated data. Decryption iterates through the key vector to compute each key's identifier using $\mathsf{F}_{\mathsf{kcv}}$ and find the first key with one matching T_k. If AEAD decryption with this key succeeds, then the corresponding decrypted plaintext is returned.

Key check values have been widely used in practice [2, 4, 18, 27, 31] to derive a value from the key, typically using a hash function or block cipher, that can then be used to confirm the integrity of or identify the key during decryption. These static key identifiers can be used in two ways: as static key hints or as static key commitments. Static key hints use a non-CR function to derive an

Application	Key Check Value	KH or KC?
AWS Encryption SDK [4]	$\mathsf{SHA256}(K \Vert \texttt{0x436f6d6d69740102})$	KC
GlobalPlatform [18]	$\mathsf{msb}_{24}(\mathsf{AES}_K(([1]_8)^{16}))$	KH
Telegram [31]	$\mathsf{lsb}_{64}(\mathsf{SHA1}(K))$	KH
PKCS#11 [27]	$\mathsf{msb}_{24}(\mathsf{AES}_K(0^{128}))$	KH

Fig. 8: Key check value functions used in practice. We list whether each function is a key hint (KH) or key commitment (KC).

identifier from the key, meaning that key hints are not unique to a single key. Because they are not used to commit to a key, they can be short and efficiently computed, while still enabling AEAD-KI schemes to narrow down the scope of keys to check during decryption. For instance, one common technique is taking 24 bits from the AES evaluation of the key over some fixed string. However, key hints must be used with an FROB AEAD scheme in practice to guarantee robust key identification.

Conversely, static key commitments do commit to the key and therefore do not need to be used with an FROB AEAD scheme. In practice, this means key commitments must employ a CR key check value function, which can be more computationally intensive and require longer key tags. For instance, Albertini et al. [2] suggest a variant of this method as their "generic solution" for adding key commitment to AEAD schemes. Their Type I and II schemes in particular feature a static identifier by computing two SHA256 hashes over the key and using these values for the key identifier and AEAD encryption key. This scheme has been adopted as the default method of key identification by AWS [4]. We summarize some sample schemes used for both key hints and key commitments in Figure 8. Later in this section we will show detailed security results for both types of static key identifiers.

Using a key derivation function. Whenever a fixed value is computed from the key as a key tag and then composed with an AEAD scheme that uses the same key, there is the potential that the AEAD scheme uses the same value for its internal computation. Since the key tag is sent in the clear, this could lead to confidentiality or integrity vulnerabilities. Indeed, Iwata and Wang [21] have shown that this does happen in practice. They describe forgery attacks for several variants of CBC-MAC proposed by ISO/IEC 9797-1:2011 [20] when used with the key check value suggested by ANSI X9.24-1:2009 [3].

To simplify the analysis of composing $\mathsf{F}_{\mathsf{kcv}}$ with an AEAD scheme, we also use key derivation function KDF to derive an independent AEAD encryption key. In this section, we will show that KDF will need to be a CR PRF. For similar reasons, Albertini et al. also use a CR PRF to derive a separate AEAD key. This of course results in extra overhead and could be unnecessary if the key identifier is never used in the internal computation of AEAD. One could analyze specific key check value functions and AEAD schemes that can be safely used together

without the need for a separate key derivation function; we leave this as an open problem.

5.1 Static Key Hint

Static key hints are a sub-class of static key identifiers which compute the key check value using a non-collision-resistant PRF. This means they can be short and efficiently computed, e.g. using a universal hash or by truncating the output of a hash function or block cipher. While key hints cannot be used to commit to a key, they can be used to narrow down the search space of the given key vector during decryption. However, in order to ensure that key robustness is achieved, these key hints must rely on using an FROB AEAD scheme, as we show below.

Analyzing robustness. Here we show that static key hints rely on using an FROB AEAD scheme as well as collision-resistant function KDF to achieve KI-FROB.

Theorem 4. *Let $\mathcal{A}$ be an* KI-FROB *adversary for scheme* $\mathsf{KCV}[\mathrm{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}]$. *Then we give adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{ki-frob}}_{\mathsf{KCV}[\mathrm{AEAD},\mathsf{F}_{\mathsf{kcv}},\mathsf{KDF}]}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{KDF}}(\mathcal{B}) + \mathbf{Adv}^{\text{frob}}_{\mathrm{AEAD}}(\mathcal{C}).$$

Adversaries $\mathcal{B}, \mathcal{C}$ run in time that of $\mathcal{A}$.

Proof. We prove the theorem using a sequence of game hops. Let game G_0 be the game FROB with the call to the decryption algorithm Dec replaced by the pseudocode for $\mathsf{KCV}[\mathrm{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}].\mathsf{Dec}()$. Next we transition to game G_1, which is identical to G_0 except that when KDF is called to derive the encryption key from the keys in $\mathbb{K}_1$, it checks if there was some other different key K^* prior to this call that output to the same encryption key derived from the keys in $\mathbb{K}_0$. If this happens, then G_1 will return 0.

We can upper bound the difference in advantage of $\mathcal{A}$ in G_0 and G_1 by the probability that KDF finds a collision. We then provide the CR adversary $\mathcal{B}$ such that its advantage in the CR game for KDF upper bounds this probability. $\mathcal{B}$ runs $\mathcal{A}$, which returns $\mathbb{K}_0, \mathbb{K}_1, N, AD, T_k, C$. $\mathcal{B}$ then checks if there is some key $K_0 \in \mathbb{K}_0$ such that $(\mathsf{kid}_0, K_0^*) \leftarrow K_0$ and $K_e \leftarrow \mathsf{KDF}(K_0^*)$ and some key $K_1 \in \mathbb{K}_1$ such that $(\mathsf{kid}_1, K_1^*) \leftarrow K_1$ and $K_e \leftarrow \mathsf{KDF}(K_1^*)$, and returns K_0^*, K_1^*. Notice that whenever KDF finds the collision in G_1, then $\mathcal{B}$ wins the CR game for KDF.

Finally, we upper bound the advantage of $\mathcal{A}$ in game G_1 by the advantage of the following FROB adversary $\mathcal{C}$ against AEAD. $\mathcal{C}$ runs $\mathcal{A}$, which returns $\mathbb{K}_0, \mathbb{K}_1, N, AD, T_k, C$. Let $\mathsf{Dec}(\mathbb{K}_0, N, AD, T_k, C)$ and $\mathsf{Dec}(\mathbb{K}_1, N, AD, T_k, C)$ return (K_0, M_0) and (K_1, M_1), respectively. Also let $(\mathsf{kid}_0, K_0^*) \leftarrow K_0$ and $(\mathsf{kid}_1, K_1^*) \leftarrow K_1$. Note that $\mathcal{A}$ can only win game G_1 if $(K_0, M_0) \neq \perp$ and $(K_1, M_1) \neq \perp$ and $K_0^* \neq K_1^*$, so let this be the case. $\mathcal{C}$ can return K_0^*, K_1^*, N, AD, C, where both keys successfully decrypt N, AD, C. Thus, $\mathcal{C}$ wins FROB for AEAD when $\mathcal{A}$ wins G_1 for KCV. □

Analyzing KI-nAE. We now show that $\mathsf{KCV}[\text{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}]$ is KI-nAE secure for leakage algorithm $\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}$ when AEAD is both FROB and MU-nAE secure, $\mathsf{F}_{\mathsf{kcv}}$ and KDF are multi-user PRFs, and KDF is pre-image resistant. Notably, this means that encryption is not key anonymous. We provide the theorem statement and proof sketch below. We show the full proof in the full version of this work.

Theorem 5. *Using* $\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}$*, let* $\mathcal{A}$ *be a* KI-nAE *adversary making at most* q *queries to its oracles, of which* q_k *are to* GenHonestKey*, and querying at most* m *malicious keys. Then we give KI-nAE simulator* S *and adversaries* $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ *such that*

$$\mathbf{Adv}^{\text{ki-nae}}_{\mathsf{KCV}[\text{AEAD},\mathsf{F}_{\mathsf{kcv}},\mathsf{KDF}],\mathsf{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{mu-prf}}_{\mathsf{F}_{\mathsf{kcv}},\mathsf{KDF}}(\mathcal{B}) + q_k \cdot \mathbf{Adv}^{\text{pre}}_{\mathsf{KDF}}(\mathcal{C}) + \mathbf{Adv}^{\text{frob}}_{\text{AEAD}}(\mathcal{D}) + \mathbf{Adv}^{\text{mu-nae}}_{\text{AEAD}}(\mathcal{E}) + \frac{q_k^2}{2^{\kappa+1}}.$$

$\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ *run in time that of* $\mathcal{A}$ *with* $\mathcal{O}(q)$ *overhead and* S *runs in time* $\mathcal{O}(mq)$*.*

Proof sketch: The KI-nAE simulator can simulate Enc queries by keeping track of the label for each key and generating a random n-bit string as the key check value kcv. It can then return the label appended with the key check value as the key tag along with a random string of the correct length for the encrypted plaintext. For Dec queries, the simulator can iterate through the list of key data given in the key vector $\mathbb{K}$ and check for malicious keys, for which it is directly given the secret key and can decrypt itself for any that have a matching key tag. If there are no malicious keys or none that correctly decrypt, then the simulator returns $\perp$.

We bound the advantage of $\mathcal{A}$ with a sequence of game hops. We first transition to a game in which calls to $\mathsf{F}_{\mathsf{kcv}}$ and KDF for honest keys are replaced with calls to random functions. We bound the ability of $\mathcal{A}$ to distinguish these games by the MU-PRF security of $\mathsf{F}_{\mathsf{kcv}}$ and KDF. We next transition to a game in which malicious keys in $\mathbb{K}$ queried to Dec are skipped if for KDF they are the pre-image of some honestly generated AEAD encryption key K_e computed in GenHonestKey. We bound the ability of $\mathcal{A}$ to distinguish these games by the pre-image resistance security of KDF, multiplied by a factor of q_k. Then we transition to a game in which malicious keys in $\mathbb{K}$ queried to Dec are skipped if they can decrypt some honestly generated ciphertext output by GenHonestKey. We bound the ability of $\mathcal{A}$ to distinguish between these games by the FROB security of AEAD. We then transition to a game in which we eliminate collisions when key K is chosen at random from the key space $\mathcal{K}$, for which we use the birthday bound $q_k^2/2^{\kappa+1}$ to bound. Finally, we transition to a game in which Enc generates a random string as the encrypted plaintext and Dec skips honest keys in the key vector if they were not used to produce the queried ciphertext from a call to Enc. We bound the distinguishing advantage by the MU-nAE security of AEAD. Since this last game guarantees that no malicious key can decrypt an honestly generated ciphertext and that no honest key can decrypt a malicious ciphertext, iterating through $\mathbb{K}$ in order in this game is identical to

iterating through the honest keys first and then the malicious keys, proving our claim. □

5.2 Static Key Commitment

Static key commitments are the second subclass of static key identifiers. They compute the key check value using using a collision-resistant PRF. While this means they must be longer and less efficient than key hints, they can commit to the key, and thus can be used with non-FROB AEADs, as we now prove.

Theorem 6. *Let $\mathcal{A}$ be a* KI-FROB *adversary for* $\mathsf{KCV}[\mathrm{AEAD}, \mathsf{F_{kcv}}, \mathsf{KDF}]$. *Then we give CR adversary $\mathcal{B}$, running in time that of $\mathcal{A}$, for* $\mathsf{F_{kcv}}$ *such that*

$$\mathbf{Adv}^{\text{ki-frob}}_{\mathsf{KCV}[\mathrm{AEAD},\mathsf{F_{kcv}},\mathsf{KDF}]}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{F_{kcv}}}(\mathcal{B}).$$

Proof. We construct adversary $\mathcal{B}$ as follows. It runs $\mathcal{A}$, which returns $\mathbb{K}_0, \mathbb{K}_1, N$, AD, T_k, C. Let the values returned by decryption of the ciphertext for each key vector be $(K_0, M_0), (K_1, M_1)$. Also let $(\mathsf{kid}_0, K_0^*) \leftarrow K_0$ and $(\mathsf{kid}_1, K_1^*) \leftarrow K_1$. We know that $(K_0, M_0), (K_1, M_1) \neq \perp$ and $K_0^* \neq K_1^*$ for $\mathcal{A}$ to win. This also means that $\mathsf{F_{kcv}}(K_0^*) = \mathsf{F_{kcv}}(K_1^*)$. $\mathcal{B}$ can then return K_0, K_1 as a collision for $\mathsf{F_{kcv}}$. □

Analyzing KI-nAE. While Albertini et al. do not explicitly prove security for this scheme, they claim that it meets their real-or-random AE security definition. However, any encryption scheme that attaches a fixed string to a ciphertext trivially cannot meet this definition. The benefit of our KI-nAE definition is that it captures security for non-key anonymous schemes. Indeed, our result shows that if $\mathsf{F_{kcv}}$ is a pre-image resistant multi-user PRF, KDF is a multi-user PRF, and AEAD is MU-nAE-secure, then KCV is KI-nAE-secure, for leakage $\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}$. We provide the theorem statement and proof sketch here; the full proof is in the full version of this work.

Theorem 7. *Using* $\mathsf{L}^{\mathsf{id}}_{\mathsf{Enc}}$, *let $\mathcal{A}$ be a* KI-nAE *adversary making at most q queries to its oracles, of which q_k are to* GenHonestKey, *and querying at most m malicious keys. Then we give adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}$ such that*

$$\begin{aligned}\mathbf{Adv}^{\text{ki-nae}}_{\mathsf{KCV}[\mathrm{AEAD},\mathsf{F_{kcv}},\mathsf{KDF}],\mathsf{S}}(\mathcal{A}) \leq\ & \mathbf{Adv}^{\text{mu-prf}}_{\mathsf{F_{kcv}},\mathsf{KDF}}(\mathcal{B}) + q_k \cdot \mathbf{Adv}^{\text{pre}}_{\mathsf{F_{kcv}}}(\mathcal{C}) \\ & + \mathbf{Adv}^{\text{mu-nae}}_{\text{AEAD}}(\mathcal{D}) + \frac{q_k^2}{2^{\kappa+1}}.\end{aligned}$$

$\mathcal{B}, \mathcal{C}, \mathcal{D}$ run in time that of $\mathcal{A}$ with a $\mathcal{O}(q)$ overhead and S *runs in time $\mathcal{O}(mq)$.*

Proof sketch: The proof uses the same KI-nAE simulator as that for Theorem 5. We again bound the advantage of $\mathcal{A}$ with a sequence of game hops. We first transition to a game in which calls to $\mathsf{F_{kcv}}$ and KDF for honest keys are replaced with calls to random functions. We bound the ability of $\mathcal{A}$ to distinguish these games by the MU-PRF security of $\mathsf{F_{kcv}}$ and KDF. We next transition to a game in which malicious keys in $\mathbb{K}$ queried to Dec are skipped if for $\mathsf{F_{kcv}}$ they are the pre-image

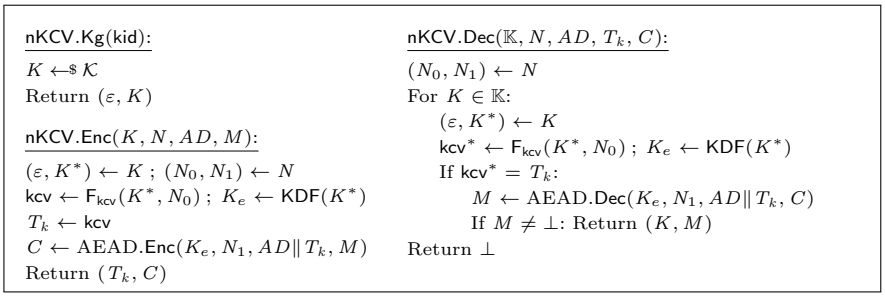

Fig. 9: A nonce-based key check value scheme nKCV parameterized by AEAD = (Kg, Enc, Dec) with key space $\mathcal{K} = \{0,1\}^{\mathsf{k}}$; key check value function $\mathsf{F}_{\mathsf{kcv}} : \{0,1\}^{\kappa} \times \{0,1\}^{r} \to \{0,1\}^{\mathsf{n}}$, and encryption key derivation function $\mathsf{KDF} : \{0,1\}^{\kappa} \to \{0,1\}^{\mathsf{k}}$.

of some honestly generated key check value kcv computed in GenHonestKey. We bound the ability of $\mathcal{A}$ to distinguish these games by the pre-image resistance security of $\mathsf{F}_{\mathsf{kcv}}$, multiplied by a factor of q_k. We then transition to a game in which we eliminate collisions when key K is chosen at random from the key space $\mathcal{K}$, for which we use the birthday bound $q_k^2/2^{\kappa+1}$ to bound. Finally, we transition to a game in which Enc generates a random string as the encrypted plaintext and Dec skips honest keys in the key vector if they were not used to produce the queried ciphertext from a call to Enc. We bound the distinguishing advantage by the MU-nAE security of AEAD. Since this last game guarantees that no malicious key can decrypt an honestly generated ciphertext and that no honest key can decrypt a malicious ciphertext, iterating through $\mathbb{K}$ in order in this game is identical to iterating through the honest keys first and then the malicious keys, proving our claim. □

While static key identifiers are versatile in that they can be used either as key hints or key commitments, they unfortunately do not provide key anonymity. Next, we will see how dynamic identifiers enable anonymous key identification.

6 Dynamic Key Identifiers

In this section we describe the key anonymous counterpart to static key identifiers, which we call *dynamic key identifiers*. A dynamic identifier is computed from the secret key during encryption using part of the input nonce. Unlike the static identifier approach, this scheme cannot use key labels as part of the key tag because key labels are fixed for a key and would therefore break key anonymity. We formalize dynamic key identifiers as an AEAD-KI scheme with $\mathsf{nKCV}[\mathrm{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}]$, shown in Figure 9. The scheme has key space $\{0,1\}^{\kappa}$ and is parameterized by encryption scheme AEAD = (Kg, Enc, Dec) with associated key space $\mathcal{K} = \{0,1\}^{\mathsf{k}}$, key check value function $\mathsf{F}_{\mathsf{kcv}} : \{0,1\}^{\kappa} \times \{0,1\}^{r} \to \{0,1\}^{\mathsf{n}}$, and encryption key derivation function $\mathsf{KDF} : \{0,1\}^{\kappa} \to \{0,1\}^{\mathsf{k}}$. For simplicity, we may refer to the scheme as nKCV when the parameters are obvious.

Encryption now takes in a nonce for which one part is used to derive the key check value and the other is used in the AEAD computation. These nonces do not need to be distinct. Encryption derives the key check value and AEAD key from the secret key using the functions $\mathsf{F}_{\mathsf{kcv}}$ and KDF, respectively. The key check value kcv is computed on part of the nonce so that it changes for each encryption call. Meanwhile, the AEAD key is computed on just the secret key, meaning the AEAD key is fixed for each nKCV secret key. Unlike for KCV, here the key check value by itself forms the key tag. Encryption adds the key tag to the authentication scope of AEAD by appending it to the authenticated data. Decryption iterates through the key vector to compute each key's identifier using $\mathsf{F}_{\mathsf{kcv}}$ and find the first key with one matching T_k.

6.1 Dynamic Key Hint

Dynamic key hints are a subclass of dynamic key identifiers which compute the key check value using using a non-collision-resistant PRF. Similar to static key hints, they can be short and more efficiently computed than key commitments. They are useful for narrowing down the search space of the given key vector during decryption. However, in order to ensure that key robustness is achieved, these key hints must rely on using an FROB AEAD scheme, as we show below.

Analyzing robustness. Here we show that dynamic key hints rely on using a CR function KDF and an FROB AEAD scheme to achieve KI-FROB.

Theorem 8. *Let $\mathcal{A}$ be a* KI-FROB *adversary for scheme* $\mathsf{nKCV}[\mathrm{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}]$. *Then we give adversaries $\mathcal{B}, \mathcal{C}$ running in time that of $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\text{ki-frob}}_{\mathsf{nKCV}[\mathrm{AEAD},\mathsf{F}_{\mathsf{kcv}},\mathsf{KDF}]}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{KDF}}(\mathcal{B}) + \mathbf{Adv}^{\text{frob}}_{\mathrm{AEAD}}(\mathcal{C}).$$

Proof. We prove the theorem using a sequence of game hops. Let game G_0 be the game KI-FROB with the call to the decryption algorithm Dec replaced by the pseudocode for $\mathsf{nKCV}[\mathrm{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}].\mathsf{Dec}()$. Next we transition to game G_1, which is identical to G_0 except that when KDF is called to derive the encryption key from the keys in $\mathbb{K}_1$, it checks if there was some other different key K^* prior to this call that output to the same encryption key derived from the keys in $\mathbb{K}_0$. If this happens, then G_1 will return 0.

We can upper bound the difference in advantage of $\mathcal{A}$ in G_0 and G_1 by the probability that KDF finds a collision. We then provide the CR adversary $\mathcal{B}$ such that its advantage in the CR game for KDF upper bounds this probability. $\mathcal{B}$ runs $\mathcal{A}$, which returns $\mathbb{K}_0, \mathbb{K}_1, N, AD, T_k, C$. $\mathcal{B}$ then checks if there is some key $K_0 \in \mathbb{K}_0$ such that $(\mathsf{kid}_0, K_0^*) \leftarrow K_0$ and $K_e \leftarrow \mathsf{KDF}(K_0^*)$ and some key $K_1 \in \mathbb{K}_1$ such that $(\mathsf{kid}_1, K_1^*) \leftarrow K_1$ and $K_e \leftarrow \mathsf{KDF}(K_1^*)$, and returns K_0^*, K_1^*. Notice that whenever KDF finds the collision in G_1, then $\mathcal{B}$ wins the CR game for KDF.

Finally, we upper bound the advantage of $\mathcal{A}$ in game G_1 by the advantage of the following FROB adversary $\mathcal{C}$ against AEAD. $\mathcal{C}$ runs $\mathcal{A}$, which returns $\mathbb{K}_0, \mathbb{K}_1, N, AD, T_k, C$. Let $\mathsf{Dec}(\mathbb{K}_0, N, AD, T_k, C)$ and $\mathsf{Dec}(\mathbb{K}_1, N, AD, T_k, C)$ return (K_0, M_0) and (K_1, M_1), respectively. Also let $(\varepsilon, K_0^*) \leftarrow K_0$, $(\varepsilon, K_1^*) \leftarrow$

K_1, and $N_0 \| N_1 \leftarrow N$. Note that $\mathcal{A}$ can only win game G_1 if $(K_0, M_0) \neq \perp$ and $(K_1, M_1) \neq \perp$ and $K_0^* \neq K_1^*$, so let this be the case. $\mathcal{C}$ can return K_0^*, K_1^*, N_1, AD, C, where again both keys decrypt N_1, AD, C. Thus, $\mathcal{C}$ wins FROB for AEAD when $\mathcal{A}$ wins G_1 for nKCV. □

Analyzing KI-nAE. We now show that $\mathsf{nKCV}[\text{AEAD}, \mathsf{F_{kcv}}, \mathsf{KDF}]$ is KI-nAE-KA secure for leakage algorithm $\mathsf{L}^{\mathsf{anon}}_{\mathsf{Enc}}$ when AEAD is both FROB and MU-nAE secure, KDF is a pre-image resistant multi-user PRF, and $\mathsf{F_{kcv}}$ is a multi-user PRF. Notably, this means that encryption is key anonymous. We assume that the adversary $\mathcal{A}$ is a nonce-respecting adversary that never queries the same N_0 or N_1 to Enc. We provide the theorem statement and proof sketch below. The proof is provided in the full version of this work.

Theorem 9. *Using* $\mathsf{L}^{\mathsf{anon}}_{\mathsf{Enc}}$*, let* $\mathcal{A}$ *be a* KI-nAE *adversary making at most* q *queries to its oracles, of which* q_k *are to* GenHonestKey*, and querying at most* m *malicious keys. Then we give adversaries* $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ *such that*

$$\mathbf{Adv}^{\text{ki-nae-anon}}_{\mathsf{nKCV}[\text{AEAD},\mathsf{F_{kcv}},\mathsf{KDF}],\mathsf{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{mu-prf}}_{\mathsf{F_{kcv}},\mathsf{KDF}}(\mathcal{B}) + q_k \cdot \mathbf{Adv}^{\text{pre}}_{\mathsf{KDF}}(\mathcal{C}) + \mathbf{Adv}^{\text{frob}}_{\text{AEAD}}(\mathcal{D}) + \mathbf{Adv}^{\text{mu-nae}}_{\text{AEAD}}(\mathcal{E}) + \frac{q_k^2}{2^{\kappa+1}}.$$

$\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ *run in time that of* $\mathcal{A}$ *with a* $\mathcal{O}(q)$ *overhead and* S *runs in time* $\mathcal{O}(mq)$.

Proof sketch: The KI-nAE simulator can simulate Enc queries by generating a random n-bit string as the key check value kcv. It can then return the key check value as the key tag along with a random string of the correct length for the encrypted plaintext. For Dec queries, the simulator can iterate through the list of key data given in the key vector $\mathbb{K}$ and check for malicious keys, for which it is directly given the secret key and can decrypt itself for any that have a matching key tag. If there are no malicious keys or none that correctly decrypt, then the simulator returns $\perp$.

We bound the advantage of $\mathcal{A}$ with a sequence of game hops. We first transition to a game in which calls to $\mathsf{F_{kcv}}$ and KDF for honest keys are replaced with calls to random functions. We bound the ability of $\mathcal{A}$ to distinguish these games by the MU-PRF security of $\mathsf{F_{kcv}}$ and KDF. We next transition to a game in which malicious keys in $\mathbb{K}$ queried to Dec are skipped if for KDF they are the pre-image of some honestly generated AEAD encryption key K_e computed in GenHonestKey. We bound the ability of $\mathcal{A}$ to distinguish these games by the pre-image resistance security of KDF, multiplied by a factor of q_k. Then we transition to a game in which malicious keys in $\mathbb{K}$ queried to Dec are skipped if they can decrypt some honestly generated ciphertext output by GenHonestKey. We bound the ability of $\mathcal{A}$ to distinguish between these games by the FROB security of AEAD. We then transition to a game in which we eliminate collisions when key K is chosen at random from the key space $\mathcal{K}$, for which we use the birthday bound $q_k^2/2^{\kappa+1}$ to bound. Finally, we transition to a game in which Enc generates a random string as the encrypted plaintext and Dec skips honest

keys in the key vector if they were not used to produce the queried ciphertext from a call to Enc. We bound the distinguishing advantage by the MU-nAE security of AEAD. Since this last game guarantees that no malicious key can decrypt an honestly generated ciphertext and that no honest key can decrypt a malicious ciphertext, iterating through $\mathbb{K}$ in order in this game is identical to iterating through the honest keys first and then the malicious keys, proving our claim. □

6.2 Dynamic Key Commitment

Dynamic key commitments are the second subclass of dynamic key identifiers; they instead compute the key check value using using a collision-resistant PRF. While this means they must be longer and less efficient than their key hint counterpart, they can be used to commit to the key. This also means that they can be used with non-FROB AEAD schemes. Dynamic key commitments parallel the Type III generic construction from Albertini et al. [2]. They are also similar to the UtC transform proposed by Bellare and Hoang [9], although this scheme is only considered in the traditional single-key setting. The UtC transform uses a committing PRF that takes as input a key and nonce and outputs pair (P, L) such that P is a string that commits to the key. L is then used as the encryption key. We note that a committing PRF can be used in place of KDF and $\mathsf{F}_{\mathsf{kcv}}$, although our formalism allows for analyzing the security requirements for deriving the key tag separately from deriving the key.

Furthermore, the NonceWrap scheme proposed by Chan and Rogaway [13] can be considered a type of dynamic key commitment scheme. Their scheme encrypts the ciphertext as $C = \text{AES}(K_1, N \| 0^3 2) \| \text{AES-GCM}(K_2, N, AD, M)$, where the first string is a 128-bit "header". During decryption, the correct key is found from a set of possible keys by re-computing the header and verifying the 32-bit all-zeros string remains intact. Interestingly, this scheme may be considered a key commitment scheme when the nonce must be specified along with the ciphertext, as in our formalization of KI-FROB. However, if, as the setting in this work intends, the nonce does not need to be specified, then this scheme does not meet KI-FROB and a key-committing AEAD should be used instead.

Analyzing robustness. Here we show that dynamic key commitments rely only on the collision-resistance of the function $\mathsf{F}_{\mathsf{kcv}}$ to achieve KI-FROB. In particular, this means that AEAD does not in fact have to be FROB.

Theorem 10. *Let $\mathcal{A}$ be a* KI-FROB *adversary for* $\mathsf{KCV}[\text{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}]$. *Then we give CR adversary $\mathcal{B}$, running in time that of $\mathcal{A}$, for* $\mathsf{F}_{\mathsf{kcv}}$ *such that*

$$\mathbf{Adv}^{\text{ki-frob}}_{\mathsf{nKCV}[\text{AEAD},\mathsf{F}_{\mathsf{kcv}},\mathsf{KDF}]}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{F}_{\mathsf{kcv}}}(\mathcal{B}).$$

$\mathcal{B}$ runs in time that of $\mathcal{A}$.

Proof. We construct CR adversary $\mathcal{B}$ against $\mathsf{F}_{\mathsf{kcv}}$ as follows. $\mathcal{B}$ runs $\mathcal{A}$, which returns $\mathbb{K}_0, \mathbb{K}_1, N, AD, T_k, C$. Let $\mathsf{Dec}(\mathbb{K}_0, N, AD, T_k, C)$ and

$\mathsf{Dec}(\mathbb{K}_1, N, AD, T_k, C)$ return (K_0, M_0) and (K_1, M_1), respectively. Also let $(\varepsilon, K_0^*) \leftarrow K_0$, $(\varepsilon, K_1^*) \leftarrow K_1$, and $N_0 \| N_1 \leftarrow N$. We know that $(K_0, M_0) \neq \perp$ and $(K_1, M_1) \neq \perp$ and $K_0^* \neq K_1^*$ for $\mathcal{A}$ to win. $\mathcal{B}$ can then return $(K_0^*, N_0), (K_1^*, N_0)$ as a collision for $\mathsf{F}_{\mathsf{kcv}}$ since $\mathsf{F}_{\mathsf{kcv}}(K_0^*, N_0) = \mathsf{F}_{\mathsf{kcv}}(K_1^*, N_0) = T_k$. We therefore have that $\mathcal{B}$ wins game CR for $\mathsf{F}_{\mathsf{kcv}}$ when $\mathcal{A}$ wins game KI-FROB for nKCV.

Analyzing KI-nAE. We now show that $\mathsf{nKCV}[\text{AEAD}, \mathsf{F}_{\mathsf{kcv}}, \mathsf{KDF}]$ is KI-nAE-KA secure for leakage algorithm $\mathsf{L}_{\mathsf{Enc}}^{\mathsf{anon}}$ when AEAD MU-nAE secure, KDF is a multi-user PRF, and $\mathsf{F}_{\mathsf{kcv}}$ is a CR multi-user PRF. Again, this means that encryption is key anonymous. We assume that the adversary $\mathcal{A}$ is a nonce-respecting adversary that never queries the same N_0 or N_1 across queries to Enc. We provide the theorem statement below; the full proof is provided in the full version of this work.

Theorem 11. *Using* $\mathsf{L}_{\mathsf{Enc}}^{\mathsf{anon}}$*, let* $\mathcal{A}$ *be a* KI-nAE *adversary making at most* q *queries to its oracles, of which* q_k *are to* GenHonestKey *and* q_e *are to* Enc*, and querying at most* m *malicious keys. Then we give adversaries* $\mathcal{B}, \mathcal{C}, \mathcal{D}$ *such that*

$$\begin{aligned}\mathbf{Adv}^{\text{ki-nae-anon}}_{\mathsf{nKCV}[\text{AEAD},\mathsf{F}_{\mathsf{kcv}},\mathsf{KDF}],\mathsf{S}}(\mathcal{A}) \leq\ & \mathbf{Adv}^{\text{mu-prf}}_{\mathsf{F}_{\mathsf{kcv}},\mathsf{KDF}}(\mathcal{B}) + q_e \cdot \mathbf{Adv}^{\text{pre}}_{\mathsf{F}_{\mathsf{kcv}}}(\mathcal{C}) \\ & + \mathbf{Adv}^{\text{mu-nae}}_{\text{AEAD}}(\mathcal{D}) + \frac{q_k^2}{2^{\kappa+1}}.\end{aligned}$$

$\mathcal{B}, \mathcal{C}, \mathcal{D}$ *run in time that of* $\mathcal{A}$ *with a* $\mathcal{O}(q)$ *overhead and* S *runs in time* $\mathcal{O}(mq)$*.*

Proof sketch: The proof uses the same KI-nAE simulator as that for Theorem 9. We again bound the advantage of $\mathcal{A}$ with a sequence of game hops. We first transition to a game in which calls to $\mathsf{F}_{\mathsf{kcv}}$ and KDF for honest keys are replaced with calls to random functions. We bound the ability of $\mathcal{A}$ to distinguish these games by the MU-PRF security of $\mathsf{F}_{\mathsf{kcv}}$ and KDF. We next transition to a game in which malicious keys in $\mathbb{K}$ queried to Dec are skipped if for $\mathsf{F}_{\mathsf{kcv}}$ they are the pre-image of some honestly generated key check value kcv computed in Enc. We bound the ability of $\mathcal{A}$ to distinguish these games by the pre-image resistance security of $\mathsf{F}_{\mathsf{kcv}}$, multiplied by a factor of q_e. We then transition to a game in which we eliminate collisions when key K is chosen at random from the key space $\mathcal{K}$, for which we use the birthday bound $q_k^2/2^{\kappa+1}$ to bound. Finally, we transition to a game in which Enc generates a random string as the encrypted plaintext and Dec skips honest keys in the key vector if they were not used to produce the queried ciphertext from a call to Enc. We bound the distinguishing advantage by the MU-nAE security of AEAD. Since this last game guarantees that no malicious key can decrypt an honestly generated ciphertext and that no honest key can decrypt a malicious ciphertext, iterating through $\mathbb{K}$ in order in this game is identical to iterating through the honest keys first and then the malicious keys, proving our claim. □

Acknowledgments

The authors thank Mihir Bellare for suggesting an improved correctness notion and various improvements in security definitions, as well as other helpful feedback on an early draft of the paper. The authors also thank Ian Miers and Nirvan Tyagi for their help in the early stages of this project. Finally, the authors are grateful to the anonymous reviewers of Asiacrypt 2022 for their feedback and suggestions. This work was supported in part by NSF grant CNS-2120651 and the NSF Graduate Research Fellowship under Grant No. DGE–2139899.

References

1. Abdalla, M., Bellare, M., Neven, G.: Robust encryption. In: Theory of Cryptography Conference. pp. 480–497. Springer (2010)
2. Albertini, A., Duong, T., Gueron, S., Kölbl, S., Luykx, A., Schmieg, S.: How to abuse and fix authenticated encryption without key commitment. In: USENIX Security (2022)
3. ANSI: Retail financial services symmetric key management Part 1: Using symmetric techniques. Standard, ANSI X9.24-1:2009 (2009)
4. Improved client-side encryption: Explicit KeyIds and key commitment. `https://aws.amazon.com/blogs/security/improved-client-side-encryption-explicit-keyids-and-key-commitment/` (2020)
5. Amazon Web Services Key Management Service, `https://aws.amazon.com/kms/`
6. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 259–274. Springer (2000)
7. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: The cascade construction and its concrete security. In: Proceedings of 37th Conference on Foundations of Computer Science. pp. 514–523. IEEE (1996)
8. Bellare, M., Hoang, V.T.: Efficient schemes for committing authenticated encryption. In: Eurocrypt. pp. 845–875. Lecture Notes in Computer Science, Springer (2022)
9. Bellare, M., Hoang, V.T.: Efficient schemes for committing authenticated encryption. In: Advances in Cryptology - EUROCRYPT (2022)
10. Bellare, M., Ng, R., Tackmann, B.: Nonces are noticed: AEAD revisited. In: Advances in Cryptology - CRYPTO. Lecture Notes in Computer Science (2019)
11. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 409–426. Springer (2006)
12. Bellare, M., Tackmann, B.: The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In: Robshaw, M., Katz, J. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 9814, pp. 247–276. Springer (2016)
13. Chan, J., Rogaway, P.: Anonymous AE. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT. vol. 11922, pp. 183–208. Springer (2019)
14. Degabriele, J.P., Karadžić, V., Melloni, A., Münch, J.P., Stam, M.: Rugged pseudorandom permutations and their applications (2022), `https://rwc.iacr.org/2022/program.php`, real World Crypto

15. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryptment. In: CRYPTO. pp. 155–186 (2018)
16. Farshim, P., Libert, B., Paterson, K.G., Quaglia, E.A.: Robust encryption, revisited. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 7778, pp. 352–368. Springer (2013)
17. Farshim, P., Orlandi, C., Rosie, R.: Security of symmetric primitives under incorrect usage of keys. IACR Transactions on Symmetric Cryptology (2017)
18. GlobalPlatform Technology Card Specification Version 2.3.1. Standard, GlobalPlatform (March 2018), https://globalplatform.org/wp-content/uploads/2018/05/GPC_CardSpecification_v2.3.1_PublicRelease_CC.pdf
19. Grubbs, P., Lu, J., Ristenpart, T.: Message franking via committing authenticated encryption. In: Katz, J., Shacham, H. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 10403, pp. 66–97. Springer (2017)
20. ISO/IEC: Information technology - security techniques - message authentication codes (MACs) - part 1: Mechanisms using a block cipher. Standard, ISO/IEC 9797-1:2011 (2011)
21. Iwata, T., Wang, L.: Impact of ANSI X9. 24-1: 2009 key check value on ISO/IEC 9797-1: 2011 MACs. In: International Workshop on Fast Software Encryption. pp. 303–322. Springer (2014)
22. Jaeger, J., Tyagi, N.: Handling adaptive compromise for practical encryption schemes. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 12170, pp. 3–32. Springer (2020)
23. Len, J., Grubbs, P., Ristenpart, T.: Partitioning Oracle Attacks. In: USENIX Security (2021)
24. libsodium AEAD, https://doc.libsodium.org/secret-key_cryptography/aead
25. Microsoft Key Vault, https://azure.microsoft.com/en-us/services/key-vault/#product-overview
26. Oracle Key Vault, https://www.oracle.com/security/database-security/key-vault/
27. PKCS #11 cryptographic token interface base specification version 2.40. Standard, OASIS (April 2015), http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.pdf
28. Rogaway, P.: Nonce-based symmetric encryption. In: Fast Software Encryption – FSE. pp. 348–358. Springer (2004)
29. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) Advances in Cryptology - EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 373–390. Springer (2006)
30. Shadowsocks. https://shadowsocks.org/en/index.html (2020)
31. Telegram mobile protocol, https://core.telegram.org/mtproto/description
32. Google Tink library, https://developers.google.com/tink