

An extended abstract [ARR⁺26] of this article appears in the proceedings of IACR Asiacrypt 2026. The full version of this article is available in the IACR eprint archive. The source code of our proof of concept implementation as well as a performance evaluation are available on Github [Tho26].

Symmetric and Asymmetric Anonymous Authenticated KEM

Benedikt Auerbach¹, Doreen Riepel², Paul Rösler³,
Lea Thiemt³, and Julian Thomas³

¹ PQShield

`benedikt.auerbach@pqshield.com`

² CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

`riepel@cispa.de`

³ FAU Erlangen-Nürnberg, Germany

`{paul.roesler,lea.thiemt,julian.thomas}@fau.de`

Abstract. The terms Signcryption, Split KEM, and Authenticated Key-Encapsulation Mechanism (AKEM) are often used synonymously to capture the amalgamation of a KEM and a Digital Signature Scheme in a single primitive. This means that a sender Alice can encapsulate a symmetric secret to the public key of receiver Bob, and Bob can use Alice’s public key to verify that Alice was indeed the sender. Some constructions of AKEM additionally use symmetric pre-shared key material between Alice and Bob to redundantly protect the confidentiality and authenticity of the encapsulated symmetric secret. So far, only special variants of such redundancy have been studied in the literature. Furthermore, beyond confidentiality and authenticity, anonymity of senders and receivers is a desirable property of AKEM that has received little attention yet.

In this work, we begin with formally defining strong compatible notions of confidentiality, authenticity, and anonymity for AKEM. In these notions, Alice and Bob can redundantly protect the encapsulated secret using all possible combinations of asymmetric sender key, asymmetric receiver key, and symmetric pre-shared key material. For every such combination, we develop an efficient construction from standard primitives.

Our consideration of anonymity and the use of symmetric pre-shared key material provokes the study of a tagging and detection mechanism: When Bob receives a ciphertext that is anonymously protected with a pre-shared key, he needs to detect which out of multiple candidate pre-shared keys to use for decryption. We prove that this detection cannot be substantially more efficient than performing trial decryptions, even when permitting pre-computation. This result is of independent interest and may have broader applications as we prove its equivalence to 1-out-of- n multi-key decryption, resp. verification, in Symmetric Encryption and Message Authentication Codes.

Keywords: Authenticated KEM, Signcryption, Anonymity, Redundant Security, Key Tagging, Lower Bound

Table of Contents

1	Introduction	3
1.1	Our Contributions	4
2	Preliminaries	9
2.1	Cryptographic Primitives	9
3	Authenticated KEM	11
3.1	Syntax and Correctness	11
3.2	Security	12
4	Tagging Mechanisms	15
5	AKEM Constructions	18
5.1	Purely Asymmetric AKEM	19
5.2	Symmetric Signcryption with Redundant Authenticity	21
5.3	AKEM with Full Redundancy	23
5.4	Further Variants	26
6	Performance Evaluation	26
7	A Lower Bound on the Efficiency of Tagging Mechanisms	27
7.1	Random-Oracle Based Tagging	28
7.2	A Lower Bound on Tag Detection	30
8	Redundancy vs. Combiners	34
A	Additional Preliminaries	39
B	Proof of Lemma 1	41
C	Security of the Purely Asymmetric AKEM	41
C.1	Proof of Theorem 1	41
C.2	Proof of Theorem 2	44
C.3	Proof of Theorem 3	46
D	Security of Symmetric Signcryption with Redundant Authenticity	47
D.1	Proof of Theorem 4	47
D.2	Proof of Theorem 5	49
D.3	Proof of Theorem 6	52
E	Security of the AKEM with Full Redundancy	56
E.1	Proof of Theorem 7	56
E.2	Proof of Theorem 8	60
E.3	Proof of Theorem 9	63
F	Tagging Equivalences	67
F.1	MAC with Multi-Key Verification	68
	Construction of MAC.	69
	Construction of Tagging Mechanism.	71
	Construction of Without Key Output.	72
F.2	Symmetric Encryption with Multi-Key Decryption	72
	Construction of Tagging Mechanism.	73
	Construction of SE.	74

1 Introduction

SIGNCRYPTION. Studying a primitive that combines Public-Key Encryption and Digital Signatures may have two primary goals: (1) Designing securely composed constructions and (2) optimizing efficiency of such compositions beyond modular black-box approaches. Both goals have been pursued in a long line of research under the term *Signcryption* [Zhe97]. Historically, however, large parts of the academic Signcryption literature stayed isolated from practice and, rather, focused on *crypto-for-crypto* type of problems and solutions [Rog15]; consider, for example, the list of articles on Identity-based Signcryption (see, e.g., [Boy03, BLMQ05, CM05]).

Nevertheless, the concept of Signcryption is evidently relevant in practice: The standard for end-to-end encrypted communication was, for a long time, Email encrypted with PGP [WHWY24] or SMIME [SRT19], both of which essentially instantiate Signcryption. Recent publications on real-world applications renewed interest in this primitive, starting with an analysis of Apple’s iMessage protocol [BS20].

SPLIT-KEM AND AKEM. Recent work has focused on *key encapsulation* instead of message encryption with two similar abstractions of the primitive emerging. Split Key Encapsulation Mechanism (Split-KEM) [BFG⁺20] was introduced to capture the core of one-pass key exchange protocols like Signal’s X3DH [MP16]. The broader class of real-world protocols that rely on such Split-KEM cores naturally extends to examples like Signal’s Sealed Sender mechanism [Lun18], unidirectional patterns of Noise [Per18], QUIC [IT21], etc. In the context of standardizing Hybrid Public-Key Encryption (HPKE) [BBLW22], the related notion of Authenticated KEM (AKEM) [ABH⁺21] was introduced. The main difference between Split-KEM and AKEM is that AKEM key pairs can be used equally at encapsulation and decapsulation, and Split-KEM keys are role dependent: Intuitively, a Split-KEM sender key pair authenticates the ciphertexts and a Split-KEM receiver key pair protects the confidentiality of the symmetric keys encapsulated therein. In contrast, an AKEM key pair could be a simple Diffie-Hellman tuple which protects both authenticity and confidentiality of its encapsulated and decapsulated symmetric keys. Alternatively, an AKEM key pair could also simply consist of both a Split-KEM sender and a Split-KEM receiver key pair. In this work we follow the AKEM abstraction.

REDUNDANT SECURITY. Users may not only want to rely on their asymmetric key pairs for encapsulation and decapsulation. For example, in HPKE, a sender may have a pre-shared symmetric key with a receiver that redundantly guarantees authenticity and confidentiality [AJKL23]: As long as either the pre-shared key or the users’ asymmetric key pairs stay uncorrupted, both security goals are maintained. Another application of redundant security is group messaging: Many chat protocols, like Signal, WhatsApp, MLS, etc. authentically encrypt group messages with a shared symmetric group key; to prevent impersonation attacks by malicious group members, the resulting ciphertext is additionally signed by

the sender. This form of redundancy has been formalized and studied under the term *Symmetric Signcryption* [JKS24, JK25]. The use of pre-shared keys has recently also been proposed as an inexpensive ad-hoc solution to achieve some form of *harvest-now decrypt-later* security for authenticated key-exchange protocols, concretely IKEv2’s key exchange protocol [FKMS20, Smy25]. Summing up, while different forms of AKEM making use of pre-shared key material have been studied in the literature, so far each subcase has been considered in isolation using custom-made security definitions, making it difficult to compare between different variants.

KEY PRIVACY AND ANONYMITY. Several early works on Signcryption also covered key privacy aka. anonymity. For Identity-based Signcryption, Boyen [Boy03] requires that a ciphertext should not reveal which out of two senders encrypted to which out of two receivers, even if the senders’ secret keys are leaked. Libert and Quisquater [LQ04] strengthen this notion for ordinary Signcryption by allowing the adversary to *choose* the senders’ secret keys. Furthermore, they introduce *ciphertext indistinguishability*, which means that a real ciphertext cannot be distinguished from a randomly sampled one, even if the sender’s secret key was chosen by the adversary. Subsequent works studied the relations between these notions [WMAS13] and reinvented them [AAP⁺23]⁴. We provide an overview of what these closely related works covered and compare this to our notions of security in Table 1. For the practical applications that motivate recent research on Split-KEMs and AKEMs, there has been no consideration of anonymity for the latter in the literature. To us, this is surprising as anonymity is regarded particularly relevant for many of the applications (e.g., Signal) that embed these Split-KEMs and AKEMs.

1.1 Our Contributions

In this work, we provide a systematic framework capturing confidentiality, authenticity, and anonymity of AKEM based on exclusive or combined use of senders’ and receivers’ asymmetric key pairs as well as symmetric pre-shared keys (Section 3). Thereby, the *exclusive* or *combined* use of these input keys can be configured differently for authenticity and privacy goals. For combined use, we require *redundant security*, which means that every used input key must guarantee the specified security goals for the resulting ciphertext on its own (i.e., even if other keys are corrupted). This form of redundant security introduces non-trivial challenges in designing suitable AKEM constructions. While based on standard building blocks, our constructions (Section 5) for the most relevant combinations of asymmetric and pre-shared key material carefully entangle authenticating and anonymizing layers that use mixed and derived key material

⁴ It remains unclear to us why the authors of [AAP⁺23] claim that “none of [the related publications] considers anonymity of the sender and intended receiver” and, simultaneously, provide no discussion about the anonymity definitions of the three other papers mentioned here, although they are cited therein.

from all available sources. We provide a performance evaluation for our proof of concept implementation [Tho26] in Section 6.

As a tool useful to improve the practical performance of decryption, we introduce Tagging Mechanisms as a new primitive (Section 4). We provide a simple Tagging Mechanism construction from PRFs and show it to perform optimally within a natural class of PRF-based constructions (Section 7). Furthermore, we prove the equivalence of this primitive to multi-key verification for Message Authentication Codes and multi-key decryption for Symmetric Encryption (Appendix F) to demonstrate its broader applicability.

SYSTEMATIC FRAMEWORK. Equipped with an AKEM syntax able to cover the optional and redundant use of asymmetric key pairs and pre-shared key material, we capture the relevant security properties in a unified framework. More precisely, we define *key indistinguishability*, *authenticity*, and *ciphertext indistinguishability* for AKEM with or without additional symmetric pre-shared keys. Our parameterized definition of confidentiality (IND) requires that an encapsulated key is indistinguishable from a random key, and this is achieved either (A) asymmetrically, or (S) symmetrically, or (A+S) redundantly. Thus, a construction either uses (A) an asymmetric *receiver* key pair, or (S) a symmetric pre-shared key, or (A+S) both to protect the secrecy of the encapsulated key. In the latter case, confidentiality must hold even if either only the receiver’s secret key or only the symmetric pre-shared key was corrupted. By corruption, we mean that the respective key is leaked to the adversary or even chosen maliciously.

Analogously, we define authenticity (AUTH) by requiring that forging a ciphertext must be hard, and this is achieved either (A) asymmetrically, or (S) symmetrically, or (A+S) redundantly such that a construction either uses (A) an asymmetric *sender* key pair, or (S) a symmetric pre-shared key, or (A+S) both. In the latter case, authenticity must hold even if either only the sender’s secret key or only the symmetric pre-shared key was corrupted.

Finally, our anonymity notion ANON says that real ciphertexts must be indistinguishable from ciphertexts randomly sampled from the construction’s ciphertext space, and this is, again, required (A) asymmetrically, or (S) symmetrically, or (A+S) redundantly such that a construction either uses (A) an asymmetric receiver key pair, or (S) a symmetric pre-shared key, or (A+S) both. In the latter case, anonymity must hold even if either only the receiver’s secret key or only the symmetric pre-shared key was corrupted.

CONSTRUCTIONS. We present three constructions featuring different combinations of properties in our framework. We start with an asymmetric AKEM, as it is the most commonly studied combination in the literature. We provide a simple construction following the encapsulate-then-sign approach using KEMs and signatures, followed by an encryption of the signature to achieve anonymity. We additionally encrypt the sender’s public key so that only the receiver can efficiently identify the sender. Apart from requiring anonymous ciphertexts of the KEM, we only rely on standard assumptions.

The second construction follows constructions from the literature on symmetric signcryption, as studied in [JKS24, JK25]. It provides redundant authenticity and serves as a warm-up for our final construction with full redundancy. Encapsulation carefully intertwines the use of a signature scheme and a PRF, using a sign-then-encrypt approach, thereby achieving anonymous ciphertexts. One difficulty in proving anonymity is that we cannot know whether the signing key or the pre-shared key will eventually be leaked, which is why we rely on the random oracle model.

The final construction improves upon this second one, where we additionally have redundant confidentiality and anonymity. For this, we strengthen the requirements of the underlying building blocks, especially the KEM. As in the second construction, we encrypt the signature, except that now we use a mix of asymmetric and symmetric secrets, to achieve redundant anonymity. However, we cannot easily hide the KEM ciphertext. Instead, we require that it satisfies a pseudorandomness property even if the decryption key is known (or leaked). Luckily, such a strong notion can be achieved in the random oracle model by the standard Hashed Diffie-Hellman based KEM. Apart from this, we need to apply several other careful design choices, which are related to what we describe next.

KEY DETECTION. If ciphertexts look pseudorandom, it becomes a natural problem for the receiver to identify the right key for decryption. In the simplest setting of our AKEM framework, it is reasonable to assume that each receiver only uses a single secret key for decrypting every received ciphertext. Once the receiver uses additional symmetric pre-shared keys (each corresponding to a different potential sender) at decryption, selecting the right one becomes a challenge. With the focus on pure Authenticated Encryption (AE), Len et al. [LGR22] study this problem, among other aspects, and design AE constructions that basically trial-decrypt pseudorandom ciphertexts with all candidate symmetric keys. In the context of messaging, this challenge has been solved for *stateful* anonymous communication protocols [BRT23, TRB⁺25] where every symmetric key is used at most once: The receiver pre-computes a list of decryption keys and random looking tags from a deterministic key chain. The sender starts with the first key-tag pair from this chain, uses the key for encryption, and attaches the tag to the resulting ciphertext; for the next encryption, the subsequent key-tag pair from the chain is used. At receipt, the receiver identifies the right decryption key by using the tag that is attached to the ciphertext.

STATELESS KEY DETECTION. The *statelessness* of AKEM makes this approach inapplicable. A trivial alternative would be that the receiver simply trial-decrypts the AKEM ciphertext with respect to its secret key and every candidate pre-shared key. Note that, in contrast to the pure AE setting [LGR22], trial-AKEM-decryption entails linearly many public-key operations. To alleviate this issue, in our constructions each ciphertext contains a tag (essentially a random nonce and its PRF-evaluation under the pre-shared key). This decreases the computational complexity of identifying the correct pre-shared key to linearly many PRF evaluations. We abstract this as a primitive called *Tagging Mechanism* and provide

a security definition tailored to preserving anonymity. Although asymptotically inefficient, it separates the tagging and detection task from the full decryption and, thereby, yields a tool that improves our AKEMs’ concrete performance and turns out to be as efficient as possible.

TAGGING LOWER BOUND. We show that for a natural class of PRF-based tagging mechanisms the trivial approach described above is optimal. More precisely, modeling the used PRF as a random oracle, we are able to show that any such tagging mechanism that is able to correctly identify the key used to generate a tag while generating tags that look random to outsiders must make a linear amount of random oracle calls, even if being allowed to perform unbounded pre-processing on the random oracle and set of candidate keys. In our proof we make use of pre-sampling techniques [Unr07, CDGS18] for random oracles.

BROADER IMPACT. We demonstrate that the primitive of Tagging Mechanisms is of independent interest by proving equivalence to its natural applications: multi-key verification for Message Authentication Codes and multi-key decryption for Symmetric Encryption. The latter equivalence proves that the Authenticated Encryption constructions with decryption under multiple candidate keys by Len et al. [LGR22] are optimal. Overall, we believe that these results demonstrate that tagging is indeed the fundamental problem at the core of efficient key-detection for anonymous communication. With these results, we aim to motivate further research on identifying suitable assumptions that can bypass our lower bound. A rather theoretic bypass is covered in the work by Goyal and Vaikuntanathan [GV22]: they consider secret key aggregation for Identity-based Encryption, which allows for multi-key decryption if these secret keys were delegated from the same master key pair. This idea is, however, too inefficient in practice and it is unsuitable for keys that are generated independently.

APPLICATIONS. Most of our AKEM constructions rely on standard assumptions for simple primitives, which makes them highly efficient, even when using the Tagging Mechanism. Hence, we consider them suitable drop-in replacements for AKEM, Split-KEM, Signcryption, and Symmetric Signcryption schemes currently deployed in practice. The list of candidates for this could include the key encapsulation for encryption of Emails via PGP [FDC⁺07] or SMIME [SRT19], the key encapsulation in HPKE [BBLW22], the core of QUIC [IT21] and many other authenticated key exchange protocols; furthermore, if *deniability* was considered negligible in practice [CCH25, LP25], Signal’s Sealed Sender [Lun18], the core of X3DH [MP16] and its recent post-quantum improvements [BFG⁺20, HKKP22, FG25, HKW25, FGPZ25], the core of unidirectional patterns from the Noise framework [Per18], etc.

LIMITATIONS AND FUTURE WORK. Influenced by recent literature [LP25, CCH25] that studies the (ir-)relevance of deniability in practice, we refrain from covering deniability here. Analyzing deniability for AKEM together with the properties covered in this work remains an open question for future work. In Section 3, we elaborate on why our security definitions for AKEM are too strong to additionally prove the combiner properties (i.e., hybrid security under multi-

		[Boy03]	[LQ04]	[WMAS13]	[AAP ⁺ 23]	Our Work
ANON	Sender Anon.	◐	◑	◑	◑	◑
	Receiver Anon.	◐	◑	◑	◑	◑
	Unlinkability	◐	◑	◑	×	◑
	Ciphertext Ind.	×	◑	◑	×	◑
IND	Confidentiality	◐	◑	◑	◑	◑
AUTH	Unforgeability	◐	◑	◑	◑	◑
	Strong Unforg.	○	◑	×	◑	◑

Table 1. Comparison of related Security Definitions. ×: Not covered; ○: Covered ...; ◐/◑: under *revealed* sender/receiver secret key; ◑/◑: under *chosen* sender/receiver secret key; ◑/◑: *redundantly* under chosen sender/receiver secret key and *either* chosen receiver/sender secret key *or* chosen pre-shared key.

ple, possibly failing, assumptions) that our constructions offer beyond the proved security under corruptions; briefly, since we allow for all-but-one corruptions of the used key material, an adversary can simply corrupt all secrets except for the one that belongs to the assumption that fails. It remains an open problem to define security for combiners that, simultaneously, permit all-but-one key corruptions. Finally, our lower bound on the Tagging Mechanism permits circumvention, beyond finding stronger, suitable assumptions, by permitting receivers to use evolving states or senders to synchronize among each other. Analyzing these modified settings is also left for future work.

FURTHER RELATED WORK. Table 1 provides an (informal) overview of the properties covered in closely related publications. Given the vast literature on Signcryption, we focus on articles that address anonymity. The idea of signcryption was initially proposed by Zheng [Zhe97]. Following this, early works prioritized formalizing fundamental goals. Notably, Boyen [Boy03] identified five security properties for (Identity-based) Signcryption: *Sender Anonymity* and *Receiver Anonymity*, which cover that it must be hard to identify which out of two senders sent a ciphertext to which out of two receivers; *Unlinkability* means that, using a receiver key, one can simulate an honest looking ciphertext; *Confidentiality* refers to standard message indistinguishability under chosen ciphertext attacks; Existential, resp. *Strong, Unforgeability*, which means that forging a ciphertext must be hard unless a different ciphertext from the same sender for the corresponding decrypted message, resp. this particular ciphertext, has been seen before. Boyen [Boy03] only allows the attacker to learn the secret keys of senders, resp. receivers. All remaining works permit maliciously chosen secret keys, and, due to our redundantly secure constructions, we allow that either all secret keys or pre-shared keys are maliciously chosen.

While Boyen’s notion of Anonymity hides who sent and received a ciphertext, Libert and Quisquater [LQ04] and all follow-up works require that real ciphertexts are indistinguishable from randomly sampled ones, where the ciphertext

space can be construction dependent. Wang et al. [WMAS13] later show that the latter implies the former.

2 Preliminaries

Notation. By $x \leftarrow y$ we denote the deterministic assignment of the value of variable y to variable x . For a function f , we write $x \leftarrow f(y)$ to assign x to the output of f on input y . Analogously, $x \leftarrow_{\S} g(y)$ denotes the probabilistic evaluation of algorithm g on input y with output x and $x \leftarrow g(y; r)$ denotes the deterministic evaluation of g on y with output x , with fixed randomness r . We emphasize (uniformly) random sampling from a set S and assignment to a variable x with $x \leftarrow_{\S} S$. For two sets X and Y , we use the shortcut notion $X \stackrel{\cup}{\leftarrow} Y$ to denote the union $X \leftarrow X \cup Y$. We use $[n]$, where n is a natural number, to denote a set $\{1, 2, \dots, n\}$, $(x_i)_{i \in [n]}$ for the sequence $(x_1, x_2, \dots, x_n)$, and $(x_i)_{i \in [n]} \leftarrow 0^n$ to denote the assignment $x_1 \leftarrow 0, \dots, x_n \leftarrow 0$. Given random variable x we denote its min entropy by $H_{\infty}(x)$ and its Rényi entropy by $H_2(x)$.

2.1 Cryptographic Primitives

In Appendix A we provide the syntax and the security definition of indistinguishability (IND) for pseudorandom functions F (PRF), as well as signature schemes (SIG) and the security notion of strong unforgeability SUF. We generally use multi-user/key multi-challenge security under adaptive corruptions.

Key Encapsulation Mechanisms. A Key Encapsulation Mechanism (KEM) KEM with encapsulation key space $\mathcal{EK}$, decapsulation key space $\mathcal{DK}$, ciphertext space $\mathcal{C}$, key space $\mathcal{K}$, and randomness space $\mathcal{R}$ consists of three algorithms, KEM.gen, KEM.enc and KEM.dec with the following syntax:

- KEM.gen : $\emptyset \rightarrow_{\S} \mathcal{EK} \times \mathcal{DK}$: Outputs a public encapsulation key pk and secret decapsulation key sk .
- KEM.enc : $\mathcal{EK} \times \mathcal{R} \rightarrow \mathcal{C} \times \mathcal{K}$ resp. KEM.dec : $\mathcal{EK} \rightarrow_{\S} \mathcal{C} \times \mathcal{K}$: Encapsulates a symmetric key.
- KEM.dec : $\mathcal{DK} \times \mathcal{C} \rightarrow \mathcal{K}$: Decapsulates a symmetric key.

For the correctness of a KEM KEM, we require that for all sequences (1) $(pk, sk) \leftarrow_{\S} \text{KEM.gen}$; (2) $(k, c) \leftarrow_{\S} \text{KEM.enc}(pk)$; and (3) $k' \leftarrow \text{KEM.dec}(sk, c)$ it holds that $k = k'$.

For security, we require indistinguishability under chosen ciphertext attacks (IND) as well as one-wayness under key checking attacks (OW-KCA), which we defer to Appendix A. Further, we consider two notions of anonymity for KEM schemes. The first one, ANON, essentially says that ciphertexts look random without the encapsulated key, as long as the secret key is not corrupted. The second one, EXP-ANON, asks for the same, but this should hold even if the secret key is corrupted. The security games ANON and EXP-ANON capturing anonymity are given in Figure 1. The former is similar to prior work that has

Game $\text{ANON}_{\text{KEM}}^b(\mathcal{A})$ and $\text{EXP-ANON}_{\text{KEM}}^b(\mathcal{A})$ 00 $n \leftarrow 0; XP \leftarrow \emptyset$ 01 $b' \leftarrow_{\$} \mathcal{A}$ 02 Stop with b' Oracle Gen 03 $n \leftarrow n + 1$ 04 $(pk_n, sk_n) \leftarrow_{\$} \text{KEM.gen}$ 05 $CK_n \leftarrow \emptyset$ 06 Return pk_n Oracle Expose(i) 07 Require $i \in [n]$ 08 Require $CK_i = \emptyset$ 09 $XP \leftarrow_{\$} \{i\} \parallel \text{only in game ANON}$ 10 Return sk_i	Oracle Enc(i) 11 Require $i \in [n]$ 12 Require $i \notin XP \parallel \text{only in game ANON}$ 13 $(k, c_0) \leftarrow_{\$} \text{KEM.enc}(pk_i)$ 14 $c_1 \leftarrow_{\$} \mathcal{C}$ 15 $CK_i \leftarrow_{\$} \{c_b\}$ 16 Return c_b Oracle Dec(i, c) 17 Require $i \in [n]$ 18 Require $c \notin CK_i$ 19 $k \leftarrow \text{KEM.dec}(sk_i, c)$ 20 Return k
---	--

Fig. 1. Game ANON and EXP-ANON for KEM scheme KEM, where the latter additionally allows $\mathcal{A}$ to obtain all secret keys.

used anonymous KEMs as building blocks [BCP⁺23] and the latter strengthens this notion to a setting where the adversary is even given the secret key. We define the advantage of an adversary $\mathcal{A}$ as

$$\text{Adv}_{\text{KEM}}^{\text{ANON}}(\mathcal{A}) := |\Pr[\text{ANON}_{\text{KEM}}^0(\mathcal{A}) \Rightarrow 1] - \Pr[\text{ANON}_{\text{KEM}}^1(\mathcal{A}) \Rightarrow 1]|,$$

and

$$\text{Adv}_{\text{KEM}}^{\text{EXP-ANON}}(\mathcal{A}) := |\Pr[\text{EXP-ANON}_{\text{KEM}}^0(\mathcal{A}) \Rightarrow 1] - \Pr[\text{EXP-ANON}_{\text{KEM}}^1(\mathcal{A}) \Rightarrow 1]|.$$

Instantiations of Anonymous KEMs. The EXP-ANON security definition is very strong; yet, we can achieve it very efficiently with simple tools, namely the Hashed Diffie-Hellman (HDH) KEM. Let $(\mathbb{G}, p, g)$ be the description of a group $\mathbb{G}$ with prime order p and generator g . In HDH, public keys are of the form $pk = g^{sk}$ for $sk \leftarrow_{\$} \mathbb{Z}_p$. Ciphertexts are group elements g^r and the encapsulated key is $H(pk, g^r, pk^r)$. Under the Strong Diffie-Hellman assumption [ABR01], the scheme is IND and OW-KCA secure. It turns out that it is also EXP-ANON simply by the fact that ciphertexts are random group elements that decapsulate successfully under any secret key.

More generally, IND secure KEMs often have the property of explicit rejection, such as the ones obtained by (variants of) the FO transform [FO99, HHK17]. Even those variants that use implicit rejection are not expected to be EXP-ANON secure because knowledge of the secret key will still allow one to decide whether decapsulation has internally failed. Nevertheless, the weaker notion of ANON security has been analyzed for lattice-based schemes, e.g., in [BCP⁺23]. Therefore, we leave it as an interesting open question whether there exist schemes from post-quantum assumptions that satisfy EXP-ANON.

3 Authenticated KEM

The notion of ordinary Authenticated KEM (AKEM) is well established. Yet, to cover the optional and redundant use of asymmetric key pairs and pre-shared key material, we adapt the standard syntax, correctness, and security definitions.

3.1 Syntax and Correctness

Syntax. An AKEM is a tuple of algorithms $\text{AK} = (\text{AK.gen}, \text{AK.enc}, \text{AK.dec})$ for secret key space $\mathcal{SK}$, public key space $\mathcal{PK}$, pre-shared key space $\mathcal{PSK}$, associated data space $\mathcal{AD}$, symmetric key space $\mathcal{K}$, and ciphertext space $\mathcal{C}$ such that:

- $\text{AK.gen} : \emptyset \rightarrow_{\$} \mathcal{SK} \times \mathcal{PK}$ generates an asymmetric key pair for sender and/or receiver.
W.l.o.g. we assume the existence of a deterministic algorithm $\mu : \mathcal{SK} \rightarrow \mathcal{PK}$.
- $\text{AK.enc} : (\mathcal{SK} \cup \{\perp\}) \times (\mathcal{PSK} \cup \{\perp\}) \times (\mathcal{PK} \cup \{\perp\}) \times \mathcal{AD} \rightarrow_{\$} \mathcal{K} \times \mathcal{C}$ encapsulates a key $k \in \mathcal{K}$ in ciphertext $c \in \mathcal{C}$ with associated data $ad \in \mathcal{AD}$ potentially using a sender secret key $sk \in \mathcal{SK}$, a pre-shared key $psk \in \mathcal{PSK}$, and a receiver public key $pk \in \mathcal{PK}$.

Since our AKEM constructions should offer anonymity in the form of *ciphertext indistinguishability*, the receiver of a ciphertext cannot know, in advance, which keys to use for decryption without performing any cryptographic operations. Thus, the decapsulation algorithm will take a set of candidate sender public keys and a set of candidate symmetric pre-shared keys. Taking the set of public keys as input to the decapsulation algorithm simply avoids encapsulating the sender's specific public key in the ciphertext. Using the set of candidate pre-shared keys, the receiver will detect which of them will actually be used for decapsulation; thereby, this set only changes when the receiver adds a new pre-shared key or removes an old one. To avoid repeated processing of these keys, we add a pre-processing algorithm:

- $\text{AK.prec} : \mathcal{P}(\mathcal{PSK}) \rightarrow_{\$} \mathcal{ST}$ on input of a set of pre-shared keys, pre-computes a state.
- $\text{AK.dec} : (\mathcal{SK} \cup \{\perp\}) \times (\mathcal{P}(\mathcal{PK}) \cup \{\perp\}) \times (\mathcal{ST} \cup \{\perp\}) \times \mathcal{AD} \times \mathcal{C} \rightarrow (\mathcal{PK} \cup \{\perp\}) \times (\mathcal{PSK} \cup \{\perp\}) \times \mathcal{K}$ decapsulates a key $k \in \mathcal{K}$ from ciphertext $c \in \mathcal{C}$ with associated data $ad \in \mathcal{AD}$ potentially using a receiver secret key $sk \in \mathcal{SK}$, a set of candidate sender public keys $PK \in \mathcal{P}(\mathcal{PK})$, and a pre-computed state that is derived from a set of candidate pre-shared keys $PSK \in \mathcal{P}(\mathcal{PSK})$, which may yield the sender's public key and the used pre-shared key.

Our following correctness and security definitions differ depending on which secrets are present; hence, we parameterize each game by (sets of) variables:

- $\text{CF} \subseteq \{\mathbf{a}, \mathbf{s}\}$: $\mathbf{a}$ indicates that the sender encapsulates towards the public key of a receiver (therefore, the receiver uses its asymmetric secret key to decapsulate), whereas $\mathbf{s}$ indicates that a pre-shared key is used for both encapsulation and decapsulation with the goal to provide confidentiality.

- $\text{AU} \subseteq \{\mathbf{a}, \mathbf{s}\}$: $\mathbf{a}$ indicates that the sender uses a secret key to encapsulate towards the receiver (therefore, the receiver will obtain a public key as a result of successful decapsulation), whereas $\mathbf{s}$ indicates that a pre-shared key is used for both encapsulation and decapsulation with the goal to provide authenticity.

Correctness. We define correctness via the game CORR as shown in Figure 2. The adversary can generate public key pairs via oracle Gen and pre-shared keys via oracle Psk, both honest and dishonest ones, except that dishonestly generated ones cannot match honest ones.

The encryption oracle Enc is called on an index s (referring to a secret key), an index i (referring to a pre-shared key), a public key pk , and associated data ad , where the first three may also be empty (indicated by $\perp$), depending on parameters CF and AU (cf. lines 16-19). After running the encapsulation algorithm, the game checks in line 21 whether pk is a public key that was generated by the experiment and, if so, stores the corresponding index. The query is stored in a set C , which will be used for the correctness check before the ciphertext c and key k are output to $\mathcal{A}$.

The decryption oracle Dec works similarly. The adversary provides an index r , a set PK , a set I , associated data ad and a ciphertext c to the oracle, based on which the game first checks whether the inputs match the syntax. After running the decapsulation algorithm, the correctness requirements are then checked in line 30. More specifically, the game checks incorrectness: If there exists an entry $(s', i', r, ad, c, k') \in C$ with r, ad, c as specified by $\mathcal{A}$, where i' is contained in the set I (or $\perp$ when $\mathbf{s} \notin \text{CF} \cup \text{AU}$) and $pk_{s'}$ refers to an honest public key contained in the set PK (or $\perp$ when $\mathbf{a} \notin \text{AU}$), then the adversary wins if (i) the public key output, or (ii) the pre-shared key output, or (iii) the decapsulated key does not match the recorded entry.

We define the advantage of $\mathcal{A}$ as $\text{Adv}_{\text{AK}}^{\text{CORR}}(\mathcal{A}) := \Pr[\text{CORR}_{\text{AK}}^{\text{CF}, \text{AU}}(\mathcal{A}) \Rightarrow 1]$, and we say that a scheme AK is perfectly correct if $\text{Adv}_{\text{AK}}^{\text{CORR}}(\mathcal{A}) = 0$.

3.2 Security

Redundant security. If $\text{CF} = \{\mathbf{a}, \mathbf{s}\}$, then confidentiality and anonymity are protected both by asymmetric and a pre-shared key material. Similarly, if $\text{AU} = \{\mathbf{a}, \mathbf{s}\}$, then authenticity is protected by both. In these cases we require *redundant security*, meaning that as long as one of the keys remains uncompromised, we expect the respective security property to hold. In our formal definitions presented below, this manifests in what is considered a trivial win. I.e., we only disallow the case where both the asymmetric and pre-shared keys used to generate the challenge or produce a forgery, respectively, were compromised. For a comparison to the related notion of combiner security, see Section 8.

Confidentiality. We define confidentiality via the game IND as shown in Figure 3, parameterized by a challenge bit b . The key generation and pre-shared key generation oracles are the same as in the correctness game, except that

Game $\text{CORR}_{\text{AK}}^{\text{CF}, \text{AU}}(\mathcal{A})$ 00 $(n, m) \leftarrow 0^2; sk., pk., psk. \leftarrow \perp^3$ 01 $C \leftarrow \emptyset$ 02 Invoke $\mathcal{A}$ 03 Stop with 0 Oracle $\text{Gen}(sk)$ 04 Require $\exists u \in [n] : pk_u = \mu(sk)$ 05 $n \leftarrow n + 1$ 06 If $sk \in SK$: 07 $(sk_n, pk_n) \leftarrow (sk, \mu(sk))$ 08 Else: $(sk_n, pk_n) \leftarrow \text{AK.gen}$ 09 Return (sk_n, pk_n) Oracle $\text{Psk}(psk)$ 10 Require $\exists i \in [m] : psk_i = psk$ 11 $m \leftarrow m + 1$ 12 If $psk \in \mathcal{PSK}$: 13 $psk_m \leftarrow psk$ 14 Else: $psk_m \leftarrow \text{PSK}$ 15 Return psk_m	Oracle $\text{Enc}(s, i, pk, ad)$ 16 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$ 17 Require $\mathbf{a} \in \text{AU} \iff s \in [n]$ 18 Require $\mathbf{s} \in \text{CF} \cup \text{AU} \iff i \in [m]$ 19 Require $\mathbf{a} \in \text{CF} \iff pk \in \mathcal{PK}$ 20 $(k, c) \leftarrow \text{AK.enc}(sk_s, psk_i, pk, ad)$ 21 If $\exists r' \in [n] : pk_{r'} = pk$: $r \leftarrow r'$; Else: $r \leftarrow \perp$ 22 $C \leftarrow \{(s, i, r, ad, c, k)\}$ 23 Return (k, c) Oracle $\text{Dec}(r, PK, I, ad, c)$ 24 Require $(r, PK, I) \in ([n] \times \mathcal{P}(\mathcal{PK}) \times \mathcal{P}([m])) \cup \{\perp^3\}$ 25 Require $\mathbf{a} \in \text{AU} \iff PK \subseteq \mathcal{PK}$ 26 Require $\mathbf{s} \in \text{CF} \cup \text{AU} \iff I \subseteq [m]$ 27 Require $\mathbf{a} \in \text{CF} \iff r \in [n]$ 28 $st \leftarrow \text{AK.prec}((psk_i)_{i \in I})$ 29 $(pk, psk, k) \leftarrow \text{AK.dec}(sk_r, PK, st, ad, c)$ 30 If $\exists (s', i', r, ad, c, k') \in C : i' \in I \cup \{\perp\}$ $\quad \wedge (s' = \perp \vee pk_{s'} \in PK) \wedge (s' \in [n] \wedge pk \neq pk_{s'})$ $\quad \vee i' \in [m] \wedge psk \neq psk_{i'} \vee k \neq k')$: Stop with 1 31 If $\exists i' \in [m] : psk_{i'} = psk$: $i \leftarrow i'$; Else: $i \leftarrow psk$ 32 Return (pk, i, k)
--	--

Fig. 2. Correctness for AKEM.

here, the adversary does not get the secrets as output and we use sets to record malicious or corrupted keys via their indices. Honestly generated keys can be corrupted via oracles `CorruptS` and `CorruptA`.

The syntax requirements in oracles `Enc` and `Dec` are the same as before. We model key indistinguishability, which asks the adversary to distinguish the real key (generated by running `AK.enc` honestly) from a randomly sampled one. For AKEMs, the adversary is not able to run the encapsulation itself, therefore, we provide an honest encapsulation and a challenge oracle, combined into one oracle via the additional input flag *ch*. Challenge queries (cf. lines 29-32) are stored in a set *CH*. The decapsulation oracle works as before, but does not provide any output when queried on an input that leaks the challenge bit trivially, see line 38.

At the end of the game, after $\mathcal{A}$ has output b' , we check whether the knowledge of secrets (i.e., via malicious or corrupted *psk* or *sk*) could have allowed $\mathcal{A}$ to trivially find b (cf. line 04). We define the advantage of $\mathcal{A}$ as

$$\text{Adv}_{\text{AK}}^{\text{IND}_{\text{CF}, \text{AU}}^0}(\mathcal{A}) := |\Pr[\text{IND}_{\text{AK}}^{0, \text{CF}, \text{AU}}(\mathcal{A}) \Rightarrow 1] - \Pr[\text{IND}_{\text{AK}}^{1, \text{CF}, \text{AU}}(\mathcal{A}) \Rightarrow 1]|.$$

Remark 1 (Relation to prior definitions). We capture the strongest possible notion, where the adversary can act as an *insider* via adaptive registration and corruption of secrets. For asymmetric confidentiality, this notion is considered standard. It can be relaxed to outsider security by adapting line 04. For symmetric confidentiality, our notion further allows the adversary to assign pre-shared keys to different (groups of) users, where these can change across queries, therefore allowing for maximal flexibility. Other variants have been considered in the literature, where the adversary can assign pre-shared keys to a fixed group [JKS24, JK25] or where each pair of users shares a pre-shared secret [AJKL23].

Game $\text{IND}_{\text{AK}}^{b, \text{CF}, \text{AU}}(\mathcal{A})$ 00 $(n, m) \leftarrow 0^2$; $sk, pk, psk \leftarrow \perp^3$ 01 $CH, XS, XA \leftarrow \emptyset^3$ 02 $b' \leftarrow_{\$} \mathcal{A}$ 03 For all $(s, i, r, ad, c) \in CH$: 04 Require $s \in \text{CF} \wedge i \notin XS$ 05 $\forall a \in \text{CF} \wedge \perp \neq r \notin XA$ 06 Stop with b' Oracle $\text{Gen}(sk)$ 06 Require $\nexists u \in [n] : pk_u = \mu(sk)$ 07 $n \leftarrow n + 1$ 08 If $sk \in \mathcal{SK}$ 09 $(sk_n, pk_n) \leftarrow (sk, \mu(sk))$ 10 $XA \leftarrow \{n\}$ 11 Else: $(sk_n, pk_n) \leftarrow_{\$} \text{AK.gen}$ 12 Return pk_n Oracle $\text{Psk}(psk)$ 13 Require $\nexists i \in XS : psk_i = psk$ 14 $m \leftarrow m + 1$ 15 If $psk \in \mathcal{PSK}$: 16 $psk_m \leftarrow psk$ 17 $XS \leftarrow \{m\}$ 18 Else: $psk_m \leftarrow_{\$} \mathcal{PSK}$ 19 Return Oracle $\text{CorruptS}(i)$ 20 Require $i \in [m]$ 21 $XS \leftarrow \{i\}$ 22 Return psk_i	Oracle $\text{Enc}(s, i, pk, ad, ch)$ 23 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$ 24 Require $a \in \text{AU} \iff s \in [n]$ 25 Require $s \in \text{CF} \cup \text{AU} \iff i \in [m]$ 26 Require $a \in \text{CF} \iff pk \in \mathcal{PK}$ 27 $(k_0, c) \leftarrow_{\$} \text{AK.enc}(sk_s, psk_i, pk, ad)$ 28 $k_1 \leftarrow_{\$} \mathcal{K}$ 29 If $\exists r' \in [n] : pk_{r'} = pk$: $r \leftarrow r'$; Else: $r \leftarrow \perp$ 30 If ch : 31 $CH \leftarrow \{(s, i, r, ad, c)\}$ 32 Return (k_0, c) 33 Return (k_0, c) Oracle $\text{Dec}(r, PK, I, ad, c)$ 34 Require $(r, PK, I) \in ([n] \times \mathcal{P}(\mathcal{PK}) \times \mathcal{P}([m])) \cup \{\perp^3\}$ 35 Require $a \in \text{AU} \iff PK \subseteq \mathcal{PK}$ 36 Require $s \in \text{CF} \cup \text{AU} \iff I \subseteq [m]$ 37 Require $a \in \text{CF} \iff r \in [n]$ 38 If $\exists (s', i', r, ad, c) \in CH : i' \in I \cup \{\perp\}$ 39 $\wedge (s' = \perp \vee psk_{s'} \in PK)$: 40 Return $\perp$ 41 $st \leftarrow \text{AK.prec}((psk_i)_{i \in I})$ 42 $(pk, psk, k) \leftarrow \text{AK.dec}(sk_r, PK, st, ad, c)$ 43 If $\exists i' \in [m] : psk_{i'} = psk$: $i \leftarrow i'$; Else: $i \leftarrow psk$ 44 Return (pk, i, k) Oracle $\text{CorruptA}(u)$ 44 Require $u \in [n]$ 45 $XA \leftarrow \{u\}$ 46 Return sk_u
--	--

Fig. 3. IND security for AKEM. Gray text denotes components identical to Figure 2; black text indicates different or new components.

Anonymity. We define anonymity via the game ANON as shown in Figure 4, parameterized by a challenge bit b . It is almost identical to the IND game, except for the challenge in oracle Enc, which asks the adversary to distinguish the real *ciphertext* (generated by running AK.enc honestly) from a randomly sampled one. Similar to IND, we model insider security, where the adversary can corrupt secrets, as long as they do not allow for trivially finding b . Thereby, *triviality* is defined via parameter CF, as we require that the same key material protects both confidentiality and anonymity. We define the advantage of $\mathcal{A}$ as

$$\text{Adv}_{\text{AK}}^{\text{ANON}_{\text{CF-AU}}}(\mathcal{A}) := |\Pr[\text{ANON}_{\text{AK}}^{0, \text{CF}, \text{AU}}(\mathcal{A}) \Rightarrow 1] - \Pr[\text{ANON}_{\text{AK}}^{1, \text{CF}, \text{AU}}(\mathcal{A}) \Rightarrow 1]|.$$

Authenticity. We define authenticity via the game AUTH as shown in Figure 5. The Enc oracle now always encapsulates honestly and records queries in a set C . The adversary wins if they can produce a non-trivial ciphertext forgery in a query to oracle Dec. Here, non-trivial means (cf. line 20) that c decapsulates successfully, but has not been recorded (for the same receiver public key and pre-shared key) in C and could not have been forged via corrupted keys (modeling insider security), depending on AU. We define the advantage of $\mathcal{A}$ as

$$\text{Adv}_{\text{AK}}^{\text{AUTH}_{\text{CF-AU}}}(\mathcal{A}) := \Pr[\text{AUTH}_{\text{AK}}^{\text{CF}, \text{AU}}(\mathcal{A}) \Rightarrow 1].$$

Game $\text{ANON}_{\text{AK}}^{b, \text{CF}, \text{AU}}(\mathcal{A})$	Oracle $\text{Enc}(s, i, pk, ad, ch)$
See Figure 3.	00 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$
Oracle $\text{Gen}(sk)$	01 Require $a \in \text{AU} \iff s \in [n]$
See Figure 3.	02 Require $s \in \text{CF} \cup \text{AU} \iff i \in [m]$
Oracle $\text{Psk}(psk)$	03 Require $a \in \text{CF} \iff pk \in \mathcal{PK}$
See Figure 3.	04 $(k, c_0) \leftarrow_{\$} \text{AK.enc}(sk_s, psk_i, pk, ad)$
Oracle $\text{CorruptS}(i)$	05 $c_1 \leftarrow_{\$} \mathcal{C}$
See Figure 3.	06 If $\exists r' \in [n] : pk_{r'} = pk$: $r \leftarrow r'$; Else: $r \leftarrow \perp$
Oracle $\text{CorruptA}(u)$	07 If ch :
See Figure 3.	08 $CH \leftarrow \{(s, i, r, ad, c_b)\}$
Oracle $\text{Dec}(r, PK, I, ad, c)$	09 Return (k, c_b)
See Figure 3.	10 Return (k, c_0)

Fig. 4. ANON security for AKEM. Gray text denotes components identical to the preceding definitions; black text indicates different or new components.

Remark 2 (Relation to combiner security). Our notion goes beyond traditional combiner security. Classical combiners guarantee security as long as one underlying scheme remains secure, but they do not meaningfully capture the stronger setting of adaptive corruptions of secrets. In our case, security should be preserved as long as either the symmetric or asymmetric secret remains uncorrupted, potentially under different corruption scenarios across users. We elaborate on this further in Section 8.

4 Tagging Mechanisms

Before moving to our AKEM constructions, we discuss a useful tool: *Tagging Mechanisms*. These essentially allow a sender holding a (pre-shared) key to generate a tag (looking pseudorandom to outsiders) to be sent alongside their message. The tag then allows the dedicated receiver, which might hold keys with respect to many senders, to efficiently identify the used key. This will prevent our construction from having to perform expensive trial decryptions, which would involve public-key operations.

Tagging mechanisms with evolving symmetric key chains have been studied in the context of secure messaging [BRT23, TRB⁺25]. The evolving key chain yields a fresh tag and key for every new message; the receiver pre-computes multiple tags and keys in advance to detect the right key at time of receipt without any computation. In the AKEM setting, however, the sender is stateless, which turns out to be a crucial limitation: Indeed, the PRF-based tagging construction we define in this section requires the receiver to perform about s PRF evaluations to identify the used key from a set of s candidates. Further, in Section 7 we will show by proving a matching lower bound that this is to some degree inherent for PRF-based constructions.

At first glance, our tagging mechanism from a theoretical point-of-view not outperforming trial-and-error decryption seems to make its use pointless. Nevertheless, we consider it a valuable contribution, as

Game $\text{AUTH}_{\text{AK}}^{\text{CF}, \text{AU}}(\mathcal{A})$	Oracle $\text{Enc}(s, i, pk, ad)$
00 $(n, m) \leftarrow 0^?$	05 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$
01 $sk, pk, psk \leftarrow \perp^3$	06 Require $a \in \text{AU} \iff s \in [n]$
02 $C, XS, XA \leftarrow \emptyset^3$	07 Require $s \in \text{CF} \cup \text{AU} \iff i \in [m]$
03 Invoke $\mathcal{A}$	08 Require $a \in \text{CF} \iff pk \in \mathcal{PK}$
04 Stop with 0	09 $(k, c) \leftarrow_{\$} \text{AK.enc}(sk_s, psk_i, pk, ad)$
Oracle $\text{Gen}(sk)$	10 If $i \in [m] \vee \exists r \in [n] : pk_r = pk : C \leftarrow \{(s, r, i, ad, c)\}$
See Figure 3.	11 Return (k, c)
Oracle $\text{Psk}(psk)$	Oracle $\text{Dec}(r, PK, I, ad, c)$
See Figure 3.	12 Require $(r, PK, I) \in ([n] \times \mathcal{P}(\mathcal{PK}) \times \mathcal{P}([m])) \cup \{\perp^3\}$
Oracle $\text{CorruptS}(i)$	13 Require $a \in \text{AU} \iff PK \subseteq \mathcal{PK}$
See Figure 3.	14 Require $s \in \text{CF} \cup \text{AU} \iff I \subseteq [m]$
Oracle $\text{CorruptA}(u)$	15 Require $a \in \text{CF} \iff r \in [n]$
See Figure 3.	16 $st \leftarrow \text{AK.prec}((psk_i)_{i \in I})$
	17 $(pk, psk, k) \leftarrow \text{AK.dec}(sk_r, PK, st, ad, c)$
	18 If $\exists i' \in I : psk_{i'} = psk : i \leftarrow i'$; Else: $i \leftarrow \perp$
	19 If $\exists s' \in [n] : pk_{s'} = pk : s \leftarrow s'$; Else: $s \leftarrow \perp$
	20 If $k \neq \perp \wedge (s, r, i, ad, c) \notin C$
	$\wedge (a \in \text{AU} \wedge \perp \neq s \notin XA \vee s \in \text{AU} \wedge i \notin XS)$
	21 Stop with 1
	22 If $\exists i' \in [m] : psk_{i'} = psk : i \leftarrow i'$; Else: $i \leftarrow psk$
	23 Return (pk, i, k)

Fig. 5. AUTH security for AKEM. Gray text denotes components identical to the preceding definitions; black text indicates different or new components.

- (a) only performing s PRF evaluations instead of s full AKEM decryptions including public-key operations should have a substantial impact on performance *in practice*, and
- (b) isolating tagging as the core-concept implied by other primitives that efficiently identify the sender key in an anonymous multi-key setting (cf. Appendix F) should help with developing more efficient methods, as our lower bound of Section 7 can be circumvented by using more structured building blocks than PRFs.

The benchmarks in Section 6 confirm the concrete performance benefit of tagging over trial-and-error key identification for our two AKEM constructions with pre-shared keys.

Syntax and Correctness. A Tagging Mechanism is a tuple of algorithms $T = (T.\text{tag}, T.\text{prec}, T.\text{det})$ with key space $\mathcal{K}$, tag space $\mathcal{T}$, and state space $\mathcal{ST}$ such that:

- $T.\text{tag} : \mathcal{K} \rightarrow_{\$} \mathcal{T}$ produces a tag $t \in \mathcal{T}$ for key $k \in \mathcal{K}$.
- $T.\text{prec} : \mathcal{P}(\mathcal{K}) \rightarrow_{\$} \mathcal{ST}$ pre-computes a state $st \in \mathcal{ST}$ for a set of keys.
- $T.\text{det} : \mathcal{ST} \times \mathcal{T} \rightarrow \mathcal{K}$ detects the key $k \in \mathcal{K}$ of the pre-computed state's key set from tag $t \in \mathcal{T}$.

In short, a Tagging Mechanism is correct if it detects the right key k from a set of candidates given a tag t computed from k . Formally, we require that for every $n \in \mathbb{N}$, every $(k_1, \dots, k_n) \in \mathcal{K}^n$, and every $i \in [n]$ that if $t \leftarrow_{\$} T.\text{tag}(k_i)$ and $st \leftarrow_{\$} T.\text{prec}(\{k_1, \dots, k_n\})$ then $k_i \leftarrow T.\text{det}(st, t)$.

Game $\text{IND}_T^b(\mathcal{A})$ 00 $n \leftarrow 0; X \leftarrow \emptyset$ 01 $b' \leftarrow_{\S} \mathcal{A}$ 02 Stop with b' Oracle Gen 03 $n \leftarrow n + 1$ 04 $T_n \leftarrow \emptyset; k_n \leftarrow_{\S} \mathcal{K}$ 05 Return Oracle Corrupt(i) 06 Require $i \in [n] \wedge T_i = \emptyset$ 07 $X \leftarrow^{\cup} \{i\}$ 08 Return k_i Oracle Tag(i) 09 Require $i \in [n]$ 10 $t \leftarrow_{\S} \text{T.tag}(k_i)$ 11 Return t	Oracle Chall(i) 12 Require $i \in [n] \setminus X$ 13 $t_0 \leftarrow_{\S} \text{T.tag}(k_i)$ 14 $t_1 \leftarrow_{\S} \mathcal{T}$ 15 $T_i \leftarrow^{\cup} \{(t_0, t_b)\}$ 16 Return t_b Oracle Det(I, K, t) 17 Require $I \subseteq [n] \wedge K \subseteq \mathcal{K}$ 18 For all $i \in I: K \leftarrow^{\cup} \{k_i\}$ 19 $st \leftarrow_{\S} \text{T.prec}(K)$ 20 If $\exists i \in I, t^* : (t^*, t) \in T_i: t \leftarrow t^*$ 21 $k \leftarrow \text{T.det}(st, t)$ 22 If $\exists i' \in I : k = k_{i'}: i \leftarrow i'$ 23 Else: $i \leftarrow \perp$ 24 Return i	Proc T.tag(k) 00 $n \leftarrow_{\S} \mathcal{K}$ 01 $x \leftarrow \text{F}(k, (\text{tag}, n))$ 02 $t \leftarrow (n, x)$ 03 Return t Proc T.prec(K) 04 $st \leftarrow K$ 05 Return st Proc T.det(st, t) 06 $(n, x) \leftarrow t$ 07 For all $k \in st$: 08 $x' \leftarrow \text{F}(k, (\text{tag}, n))$ 09 If $x' = x$: Return k 10 Return $\perp$
--	--	--

Fig. 6. Game IND for pseudorandomness of tags for Tag Mechanism T (left) and trivial construction of Tag Mechanism T from pseudorandom function F (right).

Output of Detection. Instead of outputting the detected key itself, a label or index of the detected key could be output. Either of both options seems reasonable to us and shifts the mapping between keys and labels to different locations. For simplicity of all our definitions, we opted for the variant presented above.

Security. We want the tags, computed by a Tagging Mechanism, not to reveal which key they were computed with, unless this key is known to the adversary. For this, we formally define multi-instance game IND^b for challenge bit b in Figure 6 in which adversary $\mathcal{A}$ can prompt the generation of new keys via oracle Gen, corrupt keys via oracle Corrupt, observe real tags via oracle Tag and real-or-random tags via oracle Chall, and pre-compute states from honest keys and adversarially chosen ones for detection of tags via oracle Det. When querying challenges, the adversary either always sees real tags (if $b = 0$) or random ones sampled uniformly from tag space $\mathcal{T}$. Random tags from oracle Chall are mapped to their real counterparts in oracle Det to avoid trivial wins. Eventually, $\mathcal{A}$ terminates with guess b' and we define the advantage in winning as

$$\text{Adv}_T^{\text{IND}}(\mathcal{A}) := |\Pr[\text{IND}_T^0(\mathcal{A}) \Rightarrow 1] - \Pr[\text{IND}_T^1(\mathcal{A}) \Rightarrow 1]|.$$

Trivial Construction. The inefficient, trivial way of building a Tagging Mechanism is via trial-computations at detection: The tag $t = (n, x)$ is composed of the output x of a pseudorandom function on input of the used key k and a random nonce n ; for detecting the key, each candidate key is evaluated together with the tag's nonce n via the pseudorandom function, and then the resulting output x' is compared with the tag's component x . Figure 6 provides the full construction.

Lemma 1. *The Tagging Mechanism from Figure 6 offers indistinguishability. More precisely, for every adversary $\mathcal{A}$ against Tagging Mechanism T from Figure 6 in game IND from Figure 6, there exists an adversary $\mathcal{R}_F$ against game*

Scheme	Security	(Security of) Components			
		KEM	SIG	T	F
AK _{a-a}	IND	IND	–	n/a	IND
	ANON	IND + ANON	–	n/a	IND
	AUTH	–	SUF	n/a	–
AK _{s-sa}	IND	n/a	–	–	IND
	ANON	n/a	–	IND	ROM
	AUTH	n/a	SUF	–	IND
AK _{sa-sa}	IND	OW-KCA	–	–	ROM
	ANON	OW-KCA + EXP-ANON	–	IND	ROM
	AUTH	–	SUF	–	IND

Table 2. Overview for the assumptions of our three AKEM constructions.

IND from Figure 11 for the underlying PRF F such that

$$\text{Adv}_T^{\text{IND}}(\mathcal{A}) \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_F) + \frac{2q^2}{|\mathcal{K}|}$$

where q is the number of queries to the challenge and tag oracle and the running time of $\mathcal{R}_F$ is about that of $\mathcal{A}$.

The proof can be found in Appendix B.

Equivalences and Lower Bound. Clearly, the provided trivial construction requires computation linear in the number of candidate keys at detection and it does not make use of the pre-computation algorithm.

In Section 7, we prove that this performance is nearly optimal for PRF-based tagging (F being modeled as a random oracle), even allowing for arbitrary pre-computation. Furthermore, in Appendix F, we show that building an efficient Tagging Mechanism is equivalent to building efficient multi-key decryption in Symmetric Encryption and multi-key verification in Message Authentication Codes (MAC), if ciphertexts and MAC tags are pseudorandom, respectively.

5 AKEM Constructions

We now present three constructions offering different combinations of properties:

- A purely asymmetric AKEM AK_{a-a} (Section 5.1), where both confidentiality and authenticity are provided by public-key primitives, namely KEMs and signatures. The construction is in line with most of the literature on AKEMs; however, we additionally make some tweaks to achieve anonymity.
- A symmetric signcryption scheme AK_{s-sa} (Section 5.2) with redundant authenticity, where confidentiality and authenticity are achieved via a pre-shared secret, but the latter can additionally be ensured by public keys. The construction uses our new tagging mechanism and also achieves anonymity.

Proc AK.gen 00 $(sk_{\text{KEM}}, pk_{\text{KEM}}) \leftarrow_{\$} \text{KEM.gen}$ 01 $(sk_{\text{SIG}}, pk_{\text{SIG}}) \leftarrow_{\$} \text{SIG.gen}$ 02 $pk \leftarrow (pk_{\text{KEM}}, pk_{\text{SIG}})$ 03 $sk \leftarrow (sk_{\text{KEM}}, sk_{\text{SIG}}, H(pk))$ 04 Return (sk, pk) Proc AK.enc $(sk, \perp, pk, ad)$ 05 $(\_, sk_{\text{SIG}}, h) \leftarrow sk$ 06 $(pk_{\text{KEM}}, \_) \leftarrow pk$ 07 $(k_{\text{KEM}}, c_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$ 08 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_{\text{SIG}}, (c_{\text{KEM}}, h, ad))$ 09 $k_{\text{otp}} \leftarrow F(k_{\text{KEM}}, \text{otp})$ 10 $c \leftarrow (c_{\text{KEM}}, (h, \sigma) \oplus k_{\text{otp}})$ 11 $k \leftarrow F(k_{\text{KEM}}, (\text{key}, \mu(sk), pk, c, ad))$ 12 Return (k, c)	Proc AK.dec $(sk, PK, \perp, ad, c)$ 13 $(sk_{\text{KEM}}, \_) \leftarrow sk$ 14 $(c_{\text{KEM}}, c') \leftarrow c$ 15 $k_{\text{KEM}} \leftarrow_{\$} \text{KEM.dec}(sk_{\text{KEM}}, c_{\text{KEM}})$ 16 $k_{\text{otp}} \leftarrow F(k_{\text{KEM}}, \text{otp})$ 17 $(h, \sigma) \leftarrow c' \oplus k_{\text{otp}}$ 18 $pk \leftarrow \{pk' \in PK \mid H(pk') = h\}$ $\parallel$ If $pk \in PK$ is ordered by hash, this is fast 19 $(\_, pk_{\text{SIG}}) \leftarrow pk$ 20 $b \leftarrow \text{SIG.vfy}(pk_{\text{SIG}}, (c_{\text{KEM}}, h, ad), \sigma)$ 21 If $b = \text{fal} \vee pk = \emptyset$: Return $\perp^3$ 22 $k \leftarrow F(k_{\text{KEM}}, (\text{key}, pk, \mu(sk), c, ad))$ 23 Return $(pk, \perp, k)$
---	--

Fig. 7. AKEM construction $\text{AK}_{\text{a-a}}$ with $\text{CF} = \text{AU} = \text{a}$.

- A fully redundant AKEM $\text{AK}_{\text{sa-sa}}$ (Section 5.3), where both properties are provided by pre-shared secrets and public/secret keys. The construction carefully combines the ideas from the first two schemes and, for this, mixes the derived key material and entangles the authentication mechanisms.

An overview of the required security assumptions is given in Table 2. We discuss all remaining combinations (and their irrelevance) in Section 5.4.

5.1 Purely Asymmetric AKEM

The idea of this first construction, which we call $\text{AK}_{\text{a-a}}$ and present in Figure 7, is to have a purely asymmetric anonymous AKEM (i.e., $\text{CF} = \text{AU} = \text{a}$). It uses a KEM, a signature scheme, a PRF (domain-separated with two different labels) and a hash function. A public key consists of a KEM public key (for sending) and a signature public key (for receiving). Apart from the corresponding secret keys, the final secret key additionally includes the hash of the public keys. The construction then follows the Encaps-then-Sign-then-PRF approach, however, we modify the standard generic construction to additionally provide anonymity. Sending the signature in plain would immediately break anonymity. Instead, we encrypt the signature using a one-time pad derived from the KEM key. Since the receiver will not know which sender public key to use to verify the decrypted signature, the sender additionally encrypts (the hash of) their public key.

Correctness. The construction $\text{AK}_{\text{a-a}}$ is correct if KEM and SIG are correct and if H is collision-resistant, where the last property ensures that the set of public keys (cf. line 18) contains at most one element.

Remark 3. We can define the secret key space as $\mathcal{SK}_{\text{KEM}} \times \mathcal{SK}_{\text{SIG}} \times \{0,1\}^\ell$. In all games, when the adversary provides a secret key in our game and it checks whether $sk \in \mathcal{SK}$, we then implicitly assume it also checks validity. Therefore, an adversarially chosen secret key would only be registered if $h =$

$H(\mu_{\text{KEM}}(sk_{\text{KEM}}), \mu_{\text{SIG}}(sk_{\text{SIG}}))$, where μ_{KEM} and μ_{SIG} map secret keys to their respective public key.

Theorem 1. *The AKEM construction AK_{a-a} from Figure 7 offers key indistinguishability. More precisely, for every adversary $\mathcal{A}$ against AK_{a-a} in game IND from Figure 3, there exist a reduction $\mathcal{R}_{\text{KEM}}$ in game IND from Figure 13 against the KEM KEM and a reduction $\mathcal{R}_{\text{F}}$ in game IND from Figure 11 against the PRF F, such that*

$$\text{Adv}_{\text{AK}_{a-a}}^{\text{IND}_{a-a}}(\mathcal{A}) \leq 2 \cdot \left(\text{Adv}_{\text{KEM}}^{\text{IND}}(\mathcal{R}_{\text{KEM}}) + \text{Adv}_{\text{F}}^{\text{IND}}(\mathcal{R}_{\text{F}}) \right)$$

and the running time of $\mathcal{R}_{\text{KEM}}$ and $\mathcal{R}_{\text{F}}$ is about that of $\mathcal{A}$.

The full proof is given in Appendix C.1. Roughly, we first replace the KEM keys which are encapsulated in challenge queries with random ones, ensuring that the PRF's key is uniformly random. We reduce this change to the IND security of the KEM. Then, we replace the PRF in line 11 with a random function, making challenge outputs independent of b , reducing this to the IND security of the PRF.

Theorem 2. *The AKEM construction AK_{a-a} from Figure 7 offers anonymity. More precisely, for every adversary $\mathcal{A}$ against AK_{a-a} in game ANON from Figure 4, there exist a reduction $\mathcal{R}_{\text{KEM},1}$ in game IND from Figure 13, a reduction $\mathcal{R}_{\text{KEM},2}$ in game ANON from Figure 1, both against the KEM KEM, and a reduction $\mathcal{R}_{\text{F}}$ in game IND from Figure 11 against the PRF F, such that*

$$\text{Adv}_{\text{AK}_{a-a}}^{\text{ANON}_{a-a}}(\mathcal{A}) \leq 2 \cdot \left(\text{Adv}_{\text{KEM}}^{\text{IND}}(\mathcal{R}_{\text{KEM},1}) + \text{Adv}_{\text{KEM}}^{\text{ANON}}(\mathcal{R}_{\text{KEM},2}) + \text{Adv}_{\text{F}}^{\text{IND}}(\mathcal{R}_{\text{F}}) \right),$$

where the running time of $\mathcal{R}_{\text{KEM},1}$, $\mathcal{R}_{\text{KEM},2}$ and $\mathcal{R}_{\text{F}}$ is about that of $\mathcal{A}$.

The full proof is given in Appendix C.2. Roughly, we first use the IND security of KEM KEM to switch k_{KEM} to random and its ANON security to switch c_{KEM} to random. Then, we can use the security of the PRF to switch k_{otp} to random. This way, the second ciphertext component is also randomly distributed.

Theorem 3. *The AKEM construction AK_{a-a} from Figure 7 offers authenticity. More precisely, for every adversary $\mathcal{A}$ against AK_{a-a} in game AUTH from Figure 5, there exists a reduction $\mathcal{R}_{\text{SIG}}$ in game SUF from Figure 12 against the signature scheme SIG, such that*

$$\text{Adv}_{\text{AK}_{a-a}}^{\text{AUTH}_{a-a}}(\mathcal{A}) \leq \text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{R}_{\text{SIG}}),$$

where the running time of $\mathcal{R}_{\text{SIG}}$ is about that of $\mathcal{A}$.

The full proof is given in Appendix C.3. We prove this statement by a direct reduction to strong unforgeability, showing that the adversary either must come up with a new message/signature pair or a new signature on a previous ciphertext in order to forge a valid ciphertext for an uncorrupted sender.

5.2 Symmetric Signcryption with Redundant Authenticity

The next construction, denoted by $\text{AK}_{\text{s-sa}}$ and presented in Figure 8, provides confidentiality and anonymity based on a pre-shared key, and it achieves authenticity if either the pre-shared key or the sender's secret key remains uncorrupted ($\text{CF} = \text{s}; \text{AU} = \{\text{s}, \text{a}\}$). This variant of AKEM basically captures the key encapsulation variant of what has been introduced as Symmetric Signcryption by Jaeger et al. [JKS24, JK25]. The initial motivation to study this primitive was to isolate the encryption core of group messaging protocols: Every sender uses the shared group key to authentically encrypt their messages to the group; additionally, the sender signs the message or the resulting ciphertext in order to prevent impersonation attacks by malicious group members.

Our construction is inspired by the generic Sign-then-Encrypt design in [JKS24] and can be considered a special case thereof. As a warm-up for the more complex fully redundant AKEM construction in Section 5.3, we present our scheme here; furthermore, our proofs show that our construction provides anonymity, which is not covered by Jaeger et al. [JKS24]. Finally, using our Tagging Mechanism, decapsulation can be performed efficiently even under anonymity.

Proc AK.gen 00 $(sk_{\text{SIG}}, pk) \leftarrow_{\text{s}} \text{SIG.gen}$ 01 $sk \leftarrow (sk_{\text{SIG}}, H(pk))$ 02 Return (sk, pk) Proc AK.enc $(sk, psk, \perp, ad)$ 03 $(sk_{\text{SIG}}, h) \leftarrow sk$ 04 $n \leftarrow_{\text{s}} \mathcal{N}$ 05 $k_{\text{tag}} \leftarrow F(psk, \text{tag})$ 06 $c_T \leftarrow_{\text{s}} T.\text{tag}(k_{\text{tag}})$ 07 $\sigma \leftarrow_{\text{s}} \text{SIG.sig}(sk_{\text{SIG}}, (n, c_T, h, ad))$ 08 $k_{\text{otp}} \leftarrow F(psk, (\text{otp}, n))$ 09 $k_{\tau} \leftarrow F(psk, \text{mac})$ 10 $\tau \leftarrow F(k_{\tau}, (n, c_T, h, \sigma, ad))$ 11 $c \leftarrow (n, c_T, (h, \sigma, \tau) \oplus k_{\text{otp}})$ 12 $k \leftarrow F(psk, (\text{key}, c, ad))$ 13 Return (k, c)	Proc AK.prec (PSK) 14 $K \leftarrow \emptyset; P[\cdot] \leftarrow \perp$ 15 For all $psk \in PSK$: 16 $k_{\text{tag}} \leftarrow F(psk, \text{tag})$ 17 $K \leftarrow \{k_{\text{tag}}\}$ 18 $P[k_{\text{tag}}] \leftarrow psk$ 19 $st' \leftarrow_{\text{s}} T.\text{prec}(K)$ 20 $st \leftarrow (st', P)$ 21 Return st Proc AK.dec $(\perp, PK, st, ad, c)$ 22 $(st', P) \leftarrow st; (n, c_T, c') \leftarrow c$ 23 $k_T \leftarrow T.\text{det}(st', c_T)$ 24 $psk \leftarrow P[k_T]$ 25 Require $psk \neq \perp$ 26 $k_{\text{otp}} \leftarrow F(psk, (\text{otp}, n))$ 27 $(h, \sigma, \tau) \leftarrow c' \oplus k_{\text{otp}}$ 28 $pk \leftarrow \{pk \in PK \mid H(pk) = h\}$ 29 $b \leftarrow \text{SIG.vfy}(pk, (n, c_T, h, ad), \sigma)$ 30 $k_{\tau} \leftarrow F(psk, \text{mac})$ 31 $\tau' \leftarrow F(k_{\tau}, (n, c_T, h, \sigma, ad))$ 32 If $b = \text{fal} \vee \tau \neq \tau' \vee pk = \emptyset$: Return $\perp^3$ 33 $k \leftarrow F(psk, (\text{key}, c, ad))$ 34 Return (pk, psk, k)
--	---

Fig. 8. AKEM construction $\text{AK}_{\text{s-sa}}$ with $(\text{CF} = \text{s}; \text{AU} = \{\text{s}, \text{a}\})$.

The public key of our construction in Figure 8 is just a signature scheme's public verification key, and the secret key consists of the corresponding secret signing key and a hash h of the public key.

At encapsulation, the sender derives three keys from the pre-shared key: k_{tag} for tagging the latter in c_T via Tagging Mechanism T such that the receiver

can quickly identify the right pre-shared key at decapsulation; k_{otp} for encryption and k_τ for computing an authentication tag. The sender, then, signs a random nonce n , tag c_T for identification, hash h of the sender's public key, and associated data string ad . All of this, plus the signature itself, is then authenticated with k_τ in tag τ , and the final ciphertext consists of nonce n , identification tag c_T , and an encryption of hash h , signature σ , and authentication tag τ . Finally, the encapsulated key is derived from the pre-shared key as well as the ciphertext and the associated data string.

To efficiently detect the right pre-shared key at decapsulation, the receiver pre-computes a state with respect to all their candidate pre-shared keys. For this, a dedicated detection key k_{tag} is derived from each candidate pre-shared key. Taking the set of all these detection keys, the Tagging Mechanism pre-computes its state.

At decapsulation, the Tagging Mechanism first identifies the right pre-shared key using c_T ; after this, the encrypted tuple (h, σ, τ) is decrypted from ciphertext component c' , such that signature σ can be verified with the public key hashed in h , and the authentication tag τ can be verified, too. If both verifications succeed, the receiver derives the encapsulated key.

Confidentiality is provided by the secrecy of the pre-shared key and the freshness of the nonce; anonymity is achieved due to the Tagging Mechanism's privacy and the encryption of signature, public key hash, and authentication tag. For redundant authenticity, observe that we sign all probabilistic ciphertext components, and the authentication tag deterministically covers that signature and everything that is signed by it.

Below we state the security theorems for the three properties and we give their proofs in Appendices D.1 to D.3. For ANON security, we need to rely on the random oracle model due to the adversary's capability to ask for multiple challenges and later decide on which part to corrupt.

Theorem 4. *The AKEM construction $\text{AK}_{\text{s-sa}}$ from Figure 7 offers key indistinguishability. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{s-sa}}$ in game IND from Figure 3, there exists a reduction $\mathcal{R}_F$ in game IND from Figure 11 against the PRF F , such that*

$$\text{Adv}_{\text{AK}_{\text{s-sa}}}^{\text{IND}}(\mathcal{A}) \leq 2 \cdot \left(\frac{(q_{e+d})^2}{|\mathcal{N}|} + \text{Adv}_F^{\text{IND}}(\mathcal{R}_F) \right),$$

where q_{e+d} are the number of queries to the encapsulation and decapsulation oracle, $\mathcal{N}$ is the nonce space, and the running time of $\mathcal{R}_F$ is about that of $\mathcal{A}$.

Theorem 5. *The AKEM construction $\text{AK}_{\text{s-sa}}$ from Figure 7 offers anonymity. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{s-sa}}$ in game ANON from Figure 4 in the ROM, there exists a reduction $\mathcal{R}_T$ against IND security of tagging scheme T in game IND from Figure 6, such that*

$$\text{Adv}_{\text{AK}_{\text{s-sa}}}^{\text{ANON}}(\mathcal{A}) \leq 2 \cdot \left(\frac{q_{e+d}^2}{|\mathcal{N}|} + \frac{q_{\text{psk}} q_{\text{RO}}}{|\mathcal{PSK}|} + \frac{q_{\text{psk}} q_{\text{RO}}}{|\mathcal{K}_\tau|} + \text{Adv}_T^{\text{IND}}(\mathcal{R}_T) \right),$$

with variables as in Theorem 4, q_{RO} the number of adversarial RO queries, PSK the pre-shared key space, $\mathcal{K}_\tau$ the space of authentication tag keys, and the running time of $\mathcal{R}_T$ is about that of $\mathcal{A}$.

Theorem 6. *The AKEM construction AK_{s-sa} from Figure 8 offers authenticity. More precisely, for every adversary $\mathcal{A}$ against AK_{s-sa} in game AUTH from Figure 5, there exist two reductions $\mathcal{R}_{F,1}$ as well as $\mathcal{R}_{F,2}$ against the IND security of the PRF F and a reduction $\mathcal{R}_{SIG}$ against the SUF security against the signature scheme SIG , such that*

$$\text{Adv}_{AK_{s-sa}}^{\text{AUTH}}(\mathcal{A}) \leq \frac{q_{e+d}^2}{|\mathcal{N}|} + \text{Adv}_{SIG}^{\text{SUF}}(\mathcal{R}_{SIG}) + q_{psk} \left(\sum_{i=1}^2 \text{Adv}_F^{\text{IND}}(\mathcal{R}_{F,i}) + \frac{q_d}{|\mathcal{T}|} \right),$$

with variables as in Theorem 4 and 5, q_{psk} queries to the symmetric key generation oracle, tag space $\mathcal{T}$, and the running time of $\mathcal{R}_F$ and $\mathcal{R}_{SIG}$ is about that of $\mathcal{A}$.

5.3 AKEM with Full Redundancy

Our final AKEM construction, denoted by AK_{sa-sa} and presented in Figure 9, provides full redundancy for confidentiality, anonymity, and authenticity with respect to the sender's and receiver's key pairs, respectively, and the pre-shared key ($CF = \{s, a\}$; $AU = \{s, a\}$).

A key pair consists of a KEM key pair and a signature key pair, and the secret key, additionally, contains a hash of the full public key.

The sender begins by encapsulating the KEM public key of the receiver, which yields symmetric key k_{KEM} . From key k_{KEM} , the sender derives two further keys: k_{otp1} that is used for encrypting the Tagging Mechanism's tag and k_1 that is, later, mixed for redundant security. Subsequently, the sender derives two keys from the pre-shared key psk : k_{tag} for the Tagging Mechanism and k_2 that is mixed with k_1 . Key k_{tag} is directly used for tagging the pre-shared key in c_T such that the receiver can quickly identify the latter at decapsulation. Using the signing key, the sender signs KEM ciphertext c_{KEM} , a random nonce n , identification tag c_T , public key hash h , and associated data string ad . Finally, mixed key k_{mix} is derived from k_1 and k_2 via a dual pseudorandom generator, which is modeled as a random oracle; this key remains pseudorandom even if either the receiver's secret key or the pre-shared key—but not both—is ever corrupted. Using this key k_{mix} , an authentication tag is computed for the same values that are signed plus the signature itself; furthermore, a second encryption key k_{otp2} is derived from k_{mix} as well as the final output key k , which is contextualized by associated data string ad and the final ciphertext c . This final ciphertext consists of KEM ciphertext c_{KEM} , nonce n , identification tag c_T , which is encrypted under a key derived from the KEM's encapsulated key, and the encryption of public key hash h , signature σ , and authentication tag τ under k_{otp2} , which is derived from k_{mix} .

Proc AK.gen 00 $(sk_{\text{KEM}}, pk_{\text{KEM}}) \leftarrow_{\$} \text{KEM.gen}$ 01 $(sk_{\text{SIG}}, pk_{\text{SIG}}) \leftarrow_{\$} \text{SIG.gen}$ 02 $pk \leftarrow (pk_{\text{KEM}}, pk_{\text{SIG}})$ 03 $sk \leftarrow (sk_{\text{KEM}}, sk_{\text{SIG}}, H(pk))$ 04 Return (sk, pk) Proc AK.enc(sk, psk, pk, ad) 05 $(\_, sk_{\text{SIG}}, h) \leftarrow sk$ 06 $(pk_{\text{KEM}}, \_) \leftarrow pk$ 07 $(k_{\text{KEM}}, c_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$ 08 $n \leftarrow_{\$} \mathcal{N}$ 09 $k_{\text{tag}} \leftarrow F(psk, \text{tag})$ 10 $c_{\text{T}} \leftarrow_{\$} \text{T.tag}(k_{\text{tag}})$ 11 $k_{\text{otp1}} \leftarrow F(k_{\text{KEM}}, \text{otp1})$ 12 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_{\text{SIG}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, ad))$ 13 $k_1 \leftarrow F(k_{\text{KEM}}, \text{mix})$ 14 $k_2 \leftarrow F(psk, \text{mix})$ 15 $k_{\text{mix}} \leftarrow G_d(k_1, k_2)$ 16 $\tau \leftarrow F(k_{\text{mix}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, \sigma, ad))$ 17 $k_{\text{otp2}} \leftarrow F(k_{\text{mix}}, \text{otp2})$ 18 $c \leftarrow (c_{\text{KEM}}, n, c_{\text{T}} \oplus k_{\text{otp1}}, (h, \sigma, \tau) \oplus k_{\text{otp2}})$ 19 $k \leftarrow F(k_{\text{mix}}, (\text{key}, c, ad))$ 20 Return (k, c)	Proc AK.prec(PSK) 21 $K \leftarrow \emptyset; P[\cdot] \leftarrow \perp$ 22 For all $psk \in PSK$: 23 $k_{\text{tag}} \leftarrow F(psk, \text{tag})$ 24 $K \stackrel{\cup}{\leftarrow} \{k_{\text{tag}}\}$ 25 $P[k_{\text{tag}}] \leftarrow psk$ 26 $st' \leftarrow_{\$} \text{T.prec}(K)$ 27 $st \leftarrow (st', P)$ 28 Return st Proc AK.dec(sk, PK, st, ad, c) 29 $(sk_{\text{KEM}}, \_, \_) \leftarrow sk$ 30 $(\_, pk_{\text{SIG}}) \leftarrow pk$ 31 $(st', P) \leftarrow st$ 32 $(c_{\text{KEM}}, n, c', c'') \leftarrow c$ 33 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, c_{\text{KEM}})$ 34 $k_{\text{otp1}} \leftarrow F(k_{\text{KEM}}, \text{otp1})$ 35 $c_{\text{T}} \leftarrow c' \oplus k_{\text{otp1}}$ 36 $k_{\text{T}} \leftarrow \text{T.det}(st', c_{\text{T}})$ 37 $psk \leftarrow P[k_{\text{T}}]$ 38 $k_1 \leftarrow F(k_{\text{KEM}}, \text{mix})$ 39 $k_2 \leftarrow F(psk, \text{mix})$ 40 $k_{\text{mix}} \leftarrow G_d(k_1, k_2)$ 41 $k_{\text{otp2}} \leftarrow F(k_{\text{mix}}, \text{otp2})$ 42 $(h, \sigma, \tau) \leftarrow c'' \oplus k_{\text{otp2}}$ 43 $pk \leftarrow \{pk' \in PK \mid H(pk') = h\}$ 44 $b \leftarrow \text{SIG.vfy}(pk_{\text{SIG}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, ad), \sigma)$ 45 $\tau' \leftarrow F(k_{\text{mix}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, \sigma, ad))$ 46 If $b = \text{fal} \vee \tau \neq \tau' \vee pk = \emptyset$: Return $\perp^3$ 47 $k \leftarrow F(k_{\text{mix}}, (\text{key}, c, ad))$ 48 Return (pk, psk, k)
---	---

Fig. 9. AKEM construction with $\text{CF} = \text{AU} = \{\mathbf{s}, \mathbf{a}\}$.

The pre-computation of a state from a set of candidate pre-shared keys works as for our Symmetric Signcryption construction from Section 5.2, and the decapsulation here also follows the same pattern as in Section 5.2.

The main improvements of our fully redundant construction are, however, that: (i) the Tagging Mechanism's tag is additionally encrypted under a key derived from the used KEM encapsulation, (ii) a mixed key is derived from both the pre-shared key and KEM's encapsulated key, (iii) this mixed key redundantly authenticates the ciphertext, anonymizes components therein, and seeds the final derivation of the output key.

Proof intuitions. The main challenge for the proofs is achieving redundant security under adaptive corruptions. When the adversary requests a challenge encapsulation with both primitives uncorrupted, we must produce a ciphertext that depends on the security of both while still allowing later corruption of one.

That means, for the AKEM's IND notion, we cannot directly rely on standard IND games of the KEM and PRF, since we may not corrupt the games' secret keys after a challenge query. We solve this problem by using the RO with lazy sampling: When generating a challenge where both primitives are uncorrupted, we omit the KEM/PRF keys in the subsequent RO queries. Now, distinguishing

the challenge requires querying the RO on the hidden encapsulated key or pre-shared key, which would break OW-KCA security or guess the pre-shared key. If a primitive is later corrupted, we retroactively program the RO to remain consistent.

Analogously, for the AKEM's ANON notion, we additionally require KEM EXP-ANON security and lazy sampling instead of the standard KEM ANON notion.

For the AKEM's AUTH notion, the adversary wins immediately if they produce a valid forgery (i.e., there is no post-challenge corruption). We prove that our scheme provides authenticity by reducing security first to the unforgeability of the signature scheme; then, the adversary only wins if they produce a valid authentication tag which we bound by the IND security of the PRF.

We state the security below and provide full proofs in Appendices E.1 to E.3.

Theorem 7. *The AKEM construction $\text{AK}_{\text{sa-sa}}$ from Figure 9 offers key indistinguishability. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{sa-sa}}$ in game IND from Figure 3, there exists a reduction $\mathcal{R}_{\text{KEM}}$ in game OW-KCA from Figure 14 against the KEM scheme KEM in the ROM, such that*

$$\text{Adv}_{\text{AK}_{\text{sa-sa}}}^{\text{IND}}(\mathcal{A}) \leq \frac{2(q_{e+d})^2}{|\mathcal{N}|} + 2 \left(\text{Adv}_{\text{KEM}}^{\text{OW-KCA}}(\mathcal{R}_{\text{KEM}}) + \frac{q_{\text{RO}} q_{\text{psk}}}{|\mathcal{PSK}|} + \frac{2q_{\text{RO}} q_{e+d}}{|\mathcal{K}|} \right),$$

where q_{e+d} are the number of queries to the encapsulation and decapsulation oracle, $\mathcal{N}$ is the nonce space, q_{RO} the number of adversarial RO queries, $\mathcal{PSK}$ the pre-shared key space, $\mathcal{K}$ the space of intermediate keys output by the RO, where the running time of $\mathcal{R}_{\text{KEM}}$ is about that of $\mathcal{A}$.

Theorem 8. *The AKEM construction $\text{AK}_{\text{sa-sa}}$ from Figure 9 offers anonymity. More precisely, for every adversary $\mathcal{A}$ against AK in game ANON from Figure 4, there exist a reduction $\mathcal{R}_{\text{T,IND}}$ against the IND security of the tagging scheme T, a reduction $\mathcal{R}_{\text{KEM,EXP-ANON}}$ against the EXP-ANON security of the KEM scheme KEM, and a reduction $\mathcal{R}_{\text{KEM,OW-KCA}}$ against the OW-KCA security of the KEM scheme KEM in the ROM, such that*

$$\begin{aligned} \text{Adv}_{\text{AK}_{\text{sa-sa}}}^{\text{ANON}}(\mathcal{A}) &\leq \frac{(q_{e+d})^2}{|\mathcal{N}|} + 2 \left(\frac{q_{\text{RO}} q_{\text{psk}}}{|\mathcal{PSK}|} + \text{Adv}_{\text{KEM}}^{\text{OW-KCA}}(\mathcal{R}_{\text{KEM,OW-KCA}}) \right. \\ &\quad \left. + 2\text{Adv}_{\text{KEM}}^{\text{EXP-ANON}}(\mathcal{R}_{\text{KEM,EXP-ANON}}) + \text{Adv}_{\text{T}}^{\text{IND}}(\mathcal{R}_{\text{T,IND}}) \right), \end{aligned}$$

with variables as in Theorem 7 and the running time of the reductions is about that of $\mathcal{A}$.

Theorem 9. *The AKEM construction $\text{AK}_{\text{sa-sa}}$ from Figure 9 offers authenticity. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{sa-sa}}$ in game AUTH from Figure 5, there exist two reductions $\mathcal{R}_{\text{F},1}$ as well as $\mathcal{R}_{\text{F},3}$ against the IND security of the PRF F, a reduction $\mathcal{R}_{\text{F},2}$ against the IND security of the dual PRG*

scheme modeled as a PRF, and a reduction $\mathcal{R}_{\text{SIG}}$ against the SUF security of the underlying signature scheme SIG, such that

$$\text{Adv}_{\text{AK}_{\text{sa-sa}}}^{\text{AUTH}}(\mathcal{A}) \leq \frac{q_e^2 + d}{|\mathcal{N}|} + \text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{R}_{\text{SIG}}) + q_{\text{psk}} \left(\sum_{i=1}^3 \text{Adv}_{\text{F}}^{\text{IND}}(\mathcal{R}_{\text{F},i}) + \frac{q_d}{|\mathcal{T}|} \right),$$

with variables as in Theorem 7, tag space $\mathcal{T}$, and the running time of the reductions is about that of $\mathcal{A}$.

5.4 Further Variants

We only cover three out of multiple configurations of redundant security for AKEM based on asymmetric sender and receiver key pairs and symmetric pre-shared keys. For all remaining cases, we briefly explain why studying them formally is either irrelevant or already captured by the constructions in this section:

- $\text{CF} = \mathbf{s}; \text{AU} = \mathbf{s}$: This is covered by authenticated encryption.
- $\text{CF} = \mathbf{s}; \text{AU} = \mathbf{a}$: If a symmetric key is already used for encapsulation, we believe that it is unreasonable to not also use it to redundantly authenticate the ciphertext.
- $\text{CF} = \mathbf{a}; \text{AU} = \mathbf{s}$: If a symmetric key is already used for authentication, we believe that it is unreasonable to not also use it to redundantly protect the ciphertext's secrecy.
- $\text{CF} = \mathbf{a}; \text{AU} = \{\mathbf{s}, \mathbf{a}\}$: See case $(\text{CF} = \mathbf{a}; \text{AU} = \mathbf{s})$.
- $\text{CF} = \{\mathbf{s}, \mathbf{a}\}; \text{AU} = \mathbf{s}$: This case is a simplification of $\text{CF} = \text{AU} = \{\mathbf{s}, \mathbf{a}\}$, which we build in Section 5.3; the only meaningful performance improvement for this simpler case can be gained by skipping all signature operations. For this reason, we refrain from studying this simplification explicitly.
- $\text{CF} = \{\mathbf{s}, \mathbf{a}\}; \text{AU} = \mathbf{a}$: See case $(\text{CF} = \mathbf{s}; \text{AU} = \mathbf{a})$.

6 Performance Evaluation

This section presents encapsulation and decapsulation benchmarks for our three AKEM constructions from Figures 7 to 9. Our Python 3 implementation [Tho26] instantiates the PRF and variable-length one-time-pad derivation with AES-256-CMAC in counter mode, the KEM with X25519 and HKDF-SHA-256, the signature scheme with Ed25519, and the hash function and dual PRG with SHA-256. We ran the benchmarks on an Apple M2 MacBook Air (Mac14,2) with 24 GB of unified memory and macOS Tahoe 26.2.

We benchmark receiver key sets of size 1, 4, 16, 64, and 256, using independently generated public-key pairs and uniformly sampled pre-shared keys. Each point in Figure 10 is the mean of 100 timed runs after 10 untimed warm-up runs; we distribute the position of the correct key evenly over each set and randomize the measurement order. We exclude key generation, hash-table construction, and one-time tagging precomputation from the timed measurements. For the purely

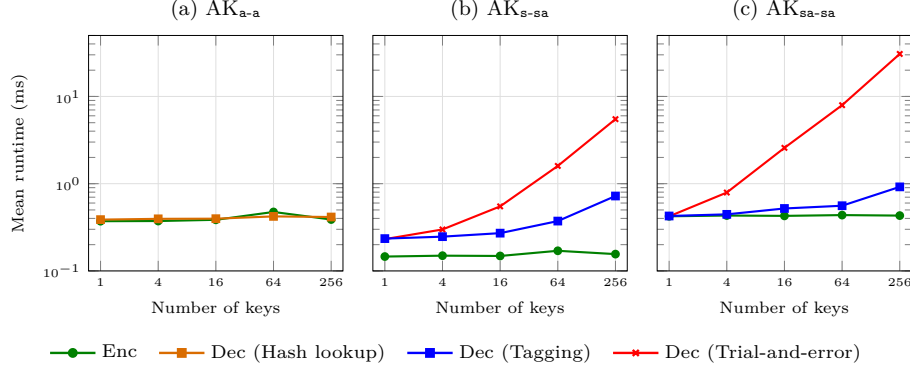


Fig. 10. Mean encapsulation (Enc) and decapsulation (Dec) runtimes as the receiver key-set size grows. Panel (a) shows AK_{a-a} from Figure 7 with hash-table lookup; panels (b) and (c) show AK_{s-sa} from Figure 8 and AK_{sa-sa} from Figure 9, respectively, with tagging and trial-and-error key identification. Each point averages 100 runs.

asymmetric construction AK_{a-a} , decapsulation recovers the sender’s public-key hash from the ciphertext. Since public keys and their hashes are public, the receiver can precompute a table mapping each hash to its public key, identify the sender by an expected constant-time lookup, and then perform one signature verification. Accordingly, mean decapsulation time remains essentially constant, increasing only from 0.39 ms for one key to 0.41 ms for 256 keys.

The other two constructions, AK_{s-sa} and AK_{sa-sa} , additionally rely on secret pre-shared keys. As a result, we cannot use the hash-table approach for key identification. We therefore compare two key-identification strategies: trial-and-error and our Tagging Mechanism from Figure 6. Trial-and-error performs a complete decapsulation with each pre-shared key until one succeeds, whereas tagging evaluates an inexpensive PRF-based tag for each key and completes decapsulation only once for the identified key. For 256 keys, tagging reduces mean decapsulation time from 5.49 ms to 0.72 ms for AK_{s-sa} and from 30.71 ms to 0.92 ms for AK_{sa-sa} , yielding speedups of approximately $7.6\times$ and $33\times$, respectively. These results confirm the practical benefit of tagging anticipated in Section 4.

7 A Lower Bound on the Efficiency of Tagging Mechanisms

Recall the tagging construction of Section 4 constructed from a pseudorandom function F deriving tags of the form $t = (n, F(k, n)) \leftarrow_{\$} T.tag(k)$. Given a set $st = K$ consisting of $s \in \mathbb{N}$ candidate keys, $T.det$ identifies the sender of a tag by trial and error and thus needs to evaluate F about s times. The goal of this section is to show that this is, to some degree, inherent for schemes using PRFs as a building block.

Intuitively, this is due to the two requirements on tagging mechanisms being seemingly at odds with each other: Tag indistinguishability IND together with the fact that tag generation is stateless implies that the tags, even if generated under the same key, must exhibit a sufficient amount of entropy. The correctness condition, on the other hand, requires to map tags, which for the class of considered tagging mechanisms will consist of PRF evaluations on points determined by $T.\text{tag}$, to the key they were generated with. If $T.\text{det}$ is supposed to do this more efficiently than by just recomputing the tags, intuitively, it must have pre-computed (some of) these values as part of running $T.\text{prec}$ and stored them as part of the state. This, however, should not be possible, if $T.\text{tag}$ generates tags consisting of PRF evaluations on points with a sufficient amount of randomness, as is implied by IND.

Theorem 11, the main result of this section, makes the above formal in the following sense. We first define a large natural class **hashTAG** of tagging mechanisms based on PRFs, which in this section we will model as a random oracle RO. We then observe that already a weak form of tag indistinguishability implies that tags generated by $T.\text{tag}$ must exhibit super-polynomial randomness. We rewrite the correctness condition of any tagging scheme in **hashTAG** as a game COR in the pre-processing random-oracle model (AI-ROM). COR first allows the adversary to perform *unbounded* preprocessing (corresponding to $T.\text{prec}$) on RO, the description of T , and a set $\{k_1, \dots, k_s\}$ of candidate keys. The main adversary (corresponding to $T.\text{det}$) given the pre-computed advice has to identify the sender of a tag $t \leftarrow_{\$} T.\text{tag}(k_{i^*})$ for random i^* . Our lower bound essentially states that for schemes with tags exhibiting super-polynomial randomness and $T.\text{det}$ making at most T random oracle evaluations the success probability of correctly identifying i^* is at most $(T+1)/s + \text{negl}$. Note that the trivial construction from Section 4 matches this bound.

Throughout this section we use λ to denote the security parameter.

7.1 Random-Oracle Based Tagging

We define a natural class **hashTAG** of tagging mechanisms deriving tags through a random oracle. In particular, it includes the construction of Section 4 if PRF F is modeled as an random oracle RO. In short, **hashTAG** contains all schemes producing tags that consist of arbitrary random oracle evaluations, i.e., evaluations of RO on inputs that may consist of the key k , the used randomness, some constants defined by the scheme, or other random oracle evaluations of this form. Additionally we allow the scheme to communicate arbitrary information on the randomness r used to generate a tag by additionally sending $f(r)$ for some arbitrary function f . This is to capture schemes like the one of Section 4 which sends the used nonce n in the clear.

Throughout this section we will often make the randomness used by algorithms explicit, writing, e.g., $t \leftarrow T.\text{tag}(k; r)$ for running $T.\text{tag}$ on input k with randomness r sampled uniformly at random.

Definition 1. Let $\text{RO} : \{0, 1\}^{d(\lambda)} \rightarrow \{0, 1\}^\lambda$ be a random oracle with input length $d(\lambda) \in \mathbb{N}$. The class `hashTAG` contains all tagging mechanisms T such that there exist $m_r, m_t \in \mathbb{N}$ and some function f such that for every key k and randomness $r = (r_1, \dots, r_{m_r})$ tags $t \leftarrow T.\text{tag}(k; r)$ are of the form

$$t = (t_1, \dots, t_{m_t}, f(r)),$$

and for every $i \in [m_t]$ the value t_i belongs to the set $\mathcal{T}^{\text{hT}} = \bigcup_{m \geq 1} \mathcal{T}_m^{\text{hT}}$ defined recursively as follows.

$$\begin{aligned} \mathcal{T}_1^{\text{hT}} &= \{t' = \text{RO}(z_1, \dots, z_\ell) \mid \forall j \in [\ell] : z_j \in \{k, r_1, \dots, r_{m_r}, c_1, \dots, c_{m_c}\}\}, \text{ and} \\ \mathcal{T}_m^{\text{hT}} &= \{t' = \text{RO}(z_1, \dots, z_\ell) \mid \forall j \in [\ell] : z_j \in \mathcal{T}_{m-1}^{\text{hT}} \cup \{k, r_1, \dots, r_{m_r}, c_1, \dots, c_{m_c}\}\} \end{aligned}$$

for $m > 1$. In the above, $\ell \in \mathbb{N}$ can be arbitrary (and in particular does not have to be the same for different t') and $c_1, \dots, c_{m_c}$ are some constants that are part of T 's description. We will sometimes refer to the values t_i as partial tags.

Note that in Definition 1, while the function f can be arbitrary, we require it to be independent of the key k , reflected by it only taking the randomness as input. We make this choice to capture the fact that we restrict ourselves to tagging schemes in which k is encoded *through the used random oracle*. We consider the construction of tagging schemes with more efficient key-detection by making use of more structured primitives (which would correspond to the use of a function f dependent on k) an interesting open problem.

In the following we will consider tags $t = (t_1, \dots, t_{m_t}, f(r))$ generated by $T.\text{tag} \in \text{hashTAG}$. By the definition above, we have that for every $i \in [m_t]$ the value t_i must satisfy $t_i = \text{RO}(z)$ for some $z = (z_1, \dots, z_\ell)$. We will denote this *last preimage* involved in the computation of t_i by $\text{preIm}_{\text{last}}(t_i) = z$.

Preprocessing models for random oracles. We prove our lower bound in the random oracle model with preprocessing a.k.a. the *auxiliary-input ROM* (AI-ROM) recalled below. Essentially, in games defined in this model a two-stage adversary (for our application think $(T.\text{prec}, T.\text{det})$) first performs unbounded preprocessing on the random oracle RO to generate a state of bounded size S . The second stage receives as input the state and a challenge to solve as specified by the game. Typically one aims to bound any such adversary's advantage in terms of S and the amount T of queries made to RO in the second stage.

Our proof relies on a pre-sampling technique introduced by Unruh [Unr07] and later refined by Coretti et al. [CDGS18]. That is, we first prove a bound in the *bit-fixing ROM* (BF-ROM), in which the adversary is allowed to program the oracle behavior on a bounded number of inputs, while RO 's outputs are set to be independently uniformly random on all remaining points, after the preprocessing phase ends. By the main result of [CDGS18] we will be able to carry over our bound in the BF-ROM to the AI-ROM.

Formally, a game **Game** in the AI-ROM (BF-ROM) is defined with respect to two-stage adversaries $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{main}})$. At the beginning of the game $\mathcal{A}_{\text{pre}}$ receives some input and is allowed to make a single call to an oracle PRE (to

be defined below). After unbounded computation $\mathcal{A}_{\text{pre}}$ returns a state st . Then $\mathcal{A}_{\text{main}}$ is invoked on st and the game's challenge, and receives access to RO (as defined by PRE during the preprocessing phase) and additional oracles as specified by **Game**. Depending on which preprocessing model is used, PRE is defined as follows.

AI-ROM: PRE samples RO uniformly at random from the set of all functions from $\{0, 1\}^{d(\lambda)}$ to $\{0, 1\}^\lambda$ and returns the function table of RO to $\mathcal{A}_{\text{pre}}$.

BF-ROM(P): The bit fixing model is parameterized by $P \in \mathbb{N}$. Oracle PRE takes as input a list of P input-output pairs (x, y) in $\{0, 1\}^{d(\lambda)} \times \{0, 1\}^\lambda$. It samples RO uniformly at random from the set of all functions from $\{0, 1\}^{d(\lambda)}$ to $\{0, 1\}^\lambda$ and overwrites the oracle's behavior to $\text{RO}(x) = y$ for all pairs it received. It does *not* provide $\mathcal{A}_{\text{pre}}$ with a description of RO.

We say a game **Game** is $((S, T), \varepsilon)$ -secure in the AI-ROM (BF-ROM(P)) if every adversary $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{main}})$ such that the state st output by $\mathcal{A}_{\text{pre}}$ is of bit-size at most S and $\mathcal{A}_{\text{main}}$ makes at most T queries to RO has an advantage bounded by ε . We recall the central result by Coretti et al. allowing us to carry over bounds in the BF-ROM to the AI-ROM.

Theorem 10 ([CDGS18], Theorem 5). *For every $P \in \mathbb{N}$ and every $\gamma > 0$, if a game **Game** is $((S, T), \varepsilon')$ -secure in the BF-ROM(P) with respect to random oracle RO, then it is $((S, T), \varepsilon)$ -secure in the AI-ROM with*

$$\varepsilon \leq \varepsilon' + \frac{2(S + \log(\gamma^{-1})) \cdot T_{\text{Game}}^{\text{comb}}}{P} + 2\gamma,$$

where $T_{\text{Game}}^{\text{comb}}$ is the combined query complexity of $\mathbf{G}$ and $\mathcal{A}$, i.e., the sum of the queries made to RO by $\mathcal{A}$ and by **Game**.

7.2 A Lower Bound on Tag Detection

Weak Indistinguishability. Consider a tagging mechanism $\mathbf{T} = (\mathbf{T}.\text{tag}, \mathbf{T}.\text{prec}, \mathbf{T}.\text{det})$ as defined in Section 4. We define *weak indistinguishability* (wIND), a relaxed version of tag IND as defined in Figure 6. It essentially states that without access to the used keys tags do not leak the identity of the key they were generated with. Formally, for tagging mechanism $\mathbf{T} = (\mathbf{T}.\text{tag}, \mathbf{T}.\text{prec}, \mathbf{T}.\text{det})$, key $k \leftarrow_{\S} \mathcal{K}(1^\lambda)$, and adversary $\mathcal{A}$ we define

$$\text{Adv}_{\mathbf{T}, \mathcal{A}, s}^{\text{wIND}}(\lambda) := |2 \Pr[b^* \leftarrow_{\S} \mathcal{O}_{b^*, k}^{\mathbf{T}, \text{tag}}(1^\lambda) \mid b^* \leftarrow_{\S} \{0, 1\}] - 1|,$$

where $\mathcal{O}_{b^*, k}^{\mathbf{T}, \text{tag}}$ is a tagging oracle that takes no input and if $b^* = 1$ returns a tag $t \leftarrow_{\S} \mathbf{T}.\text{tag}(k)$, and in the case $b^* = 0$ returns uniformly random tags $t \in \mathcal{T}$. We say that $\mathbf{T}$ satisfies weak indistinguishability if for every PPT adversary $\mathcal{A}$ it holds that $\text{Adv}_{\mathbf{T}, \mathcal{A}}^{\text{wIND}}(\lambda)$ is negligible.

The lower bound. We now prove the main result of this section, essentially stating that `hashTAG` does not contain any weakly indistinguishable tagging mechanisms which key detection algorithm is more efficient than trial decryption.

Theorem 11. *Let $T = (T.\text{tag}, T.\text{prec}, T.\text{det}) \in \text{hashTAG}$ be a tagging mechanism defined with respect to a hash function RO modeled as a random oracle such that $T.\text{tag}$ and $T.\text{det}$ run in polynomial time. Assume that $T.\text{prec}$ on input RO and a set K of keys performs unbounded preprocessing and generates a state st of polynomial size, and that $T.\text{det}$ makes at most T queries to RO .*

For $s \in \mathbb{N}$ and $i \in \{1, \dots, s\}$ let $k_i \leftarrow_{\$} \mathcal{K}(1^\lambda)$ and define $K = \{k_1, \dots, k_s\}$. Further, let $st \leftarrow_{\$} T.\text{prec}(K)$ and for $i^* \leftarrow_{\$} \{1, \dots, s\}$ generate a tag $t \leftarrow_{\$} T.\text{tag}(k_{i^*})$. If T satisfies `wIND` then the probability that $k_{i^*} = T.\text{det}(st, t)$ is at most

$$\Pr[k_{i^*} = T.\text{det}(st, t)] \leq \frac{T+1}{s} + \text{negl}.$$

We split the theorem's proof into two steps, first essentially establishing that tags must have sufficiently large min entropy, the second computing a bound on $T.\text{det}$'s success probability in terms of the min entropy of tags.

Unsurprisingly, by weak anonymity of T the tags generated by T , and in turn their last preimages must exhibit super-polynomial randomness. Before proving the pre-processing lower bound, for completeness, we make this formal in the following lemma.

Lemma 2. *Let $T \in \text{hashTAG}$ be satisfy `wIND`. Then with overwhelming probability with respect to the choice $k \leftarrow_{\$} \mathcal{K}(1^\lambda)$ tags $t = (t_1, \dots, t_{m_t}, f(r)) \leftarrow T.\text{tag}(k; r)$ satisfy that $2^{H_\infty(\text{preIm}_{\text{last}}(t_i))}$ is super-polynomial for all $i \in [m_t]$.*

Proof (Lemma 2). Let $k \in \mathcal{K}$, fix $i \in [m_t]$ and consider the adversary $\mathcal{A}$ against game `wIND` that queries $\mathcal{O}_{b^*,k}^{\text{T.tag}}$ for two tags $(t_1, \dots, t_{m_t}, r)$ and $(t'_1, \dots, t'_{m_t}, r')$ and returns 1 if $t_i = t'_i$. Its advantage conditioned on the used key being k is $|\Pr[t_i = t'_i] - 2^{-\lambda}|/2$. Further,

$$\begin{aligned} \Pr[t_i = t'_i] &\geq \Pr[\text{preIm}_{\text{last}}(t_i) = \text{preIm}_{\text{last}}(t'_i)] \\ &= 2^{-H_2(\text{preIm}_{\text{last}}(t_i))} \geq 2^{-2H_\infty(\text{preIm}_{\text{last}}(t_i))}, \end{aligned}$$

where H_2 denotes the Rényi entropy. Thus, if $2^{H_\infty(\text{preIm}_{\text{last}}(t_i))}$ was polynomial, $\mathcal{A}$ would have a noticeable advantage with respect to k . In turn, if $2^{H_\infty(\text{preIm}_{\text{last}}(t_i))}$ was polynomial for a noticeable fraction of keys, $\mathcal{A}$ would have a noticeable advantage overall. $\square$

To prove the lower bound on $T.\text{det}$'s success probability we first rephrase the correctness condition from Section 4 as a game in the `AI-ROM/BF-ROM` with respect to two-stage adversary $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{main}})$. Essentially, $\mathcal{A}_{\text{pre}}$ can be thought of as $T.\text{prec}$ and accordingly receives as input a set of s candidate keys, the description of the tagging mechanism and additionally gets access to RO , as defined by `AI-ROM` or `BF-ROM`. After unlimited preprocessing it generates a state st . Then a tag t is generated with respect to one of the candidate keys and $\mathcal{A}_{\text{main}}$, to be thought of as $T.\text{det}$, on input (st, t) has to determine the used key.

Definition 2. Let $s \in \mathbb{N}$, and consider $T = (T.\text{tag}, T.\text{prec}, T.\text{det}) \in \text{hashTAG}$ defined with respect to random oracle $\text{RO} : \{0, 1\}^{d(\lambda)} \rightarrow \{0, 1\}^\lambda$. Further, let $K = (k_1, \dots, k_s)$ be a vector of pairwise-distinct keys $k_i \in \mathcal{K}(1^\lambda)$. Tagging game $\text{COR}_{\text{PP-MOD}}$ parameterized by λ, s, T , and K is defined with respect to $\text{PP-MOD} \in \{\text{AI-ROM}, \text{BF-ROM}\}$ and preprocessing adversary $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{main}})$ as follows.

1. $\mathcal{A}_{\text{pre}}$ receives as input (s, K, T) , interacts with RO as defined in PP-MOD , and after unlimited preprocessing returns state st .
2. Sample challenge index $i^* \leftarrow_{\$} [s]$ and a tag $t = (t_1, \dots, t_{m_t}, f(r)) \leftarrow T.\text{tag}(k_{i^*}; r)$ for $r \leftarrow_{\$} \mathcal{R}$.
3. $\mathcal{A}_{\text{main}}$ receives input (st, t) , is given oracle access to RO and returns a guess k' .
4. If $k = k_{i^*}$ the game returns 1 and if not it returns 0.

We define the advantage of preprocessing adversary $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{main}})$ as

$$\text{Adv}_{\text{PP-MOD}}^{\text{COR}_{s,K,T}}(\mathcal{A}) =: \Pr[\text{COR}_{\text{PP-MOD}}^{s,K,T}(\mathcal{A}) \Rightarrow 1].$$

Lemma 3. Let s, K, T be as in Definition 2. Let $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{main}})$ be an such that $\mathcal{A}_{\text{pre}}$ returns a state of bounded size $|st| \leq S$ and $\mathcal{A}_{\text{main}}$ makes at most T calls to RO . If κ is such that the min entropy of last preimages satisfies $H_\infty(\text{preIm}_{\text{last}}(t_j)) \geq \kappa$ for every $j \in [m_t]$, where $(t_1, \dots, t_{m_t}, f(r)) \leftarrow T.\text{tag}(k_{i^*}; r)$, then $\mathcal{A}$ has an advantage in game $\text{COR}_{s,K,T}$ bounded by

$$\text{Adv}_{\text{AI-ROM}}^{\text{COR}_{s,K,T}}(\mathcal{A}) \leq 2\sqrt{\frac{2m_t(S + \lambda)T_{\text{Game}}^{\text{comb}}}{2^{-\kappa}}} + \frac{T + 1}{s} + \text{negl}(\lambda). \quad (1)$$

Proof. We first prove a lower bound for an adversary $\mathcal{A}$ against COR in the bit-fixing random oracle model and then move it to the standard auxiliary-input random oracle model. Let $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{main}})$ be an (S, T) -attacker. Let $\mathcal{P} = ((z'_1, t'_1), \dots, (z'_P, t'_P))$ be the list of P points of RO fixed by $\mathcal{A}_{\text{pre}}$ as $t'_j = \text{RO}(z'_j)$ where $z'_j \in \{0, 1\}^{d(\lambda)}$ and $t'_j \in \{0, 1\}^\lambda$ for $1 \leq j \leq P$.

Now consider adversary $\mathcal{A}_{\text{main}}$. Let $(t_1, \dots, t_{m_t}, f(r))$ be the challenge tag. We may assume, without loss of generality, that $m_t > 0$ and $f(r) = r$: Indeed, if $m_t = 0$ the adversary's input is independent of i^* , so it's winning probability is bounded by $1/s$, and if our lower bound holds in the case $f(r) = r$, providing the adversary the full randomness used, then it must also hold for a tagging mechanism where the adversary receives less information.

Intuitively, the prefixed coordinates $\mathcal{P}$ only help $\mathcal{A}_{\text{main}}$ find i^* if there exists $j \in [m_t], k \in K$ such that $t_j = T.\text{tag}(k, r)$ and $(\text{preIm}_{\text{last}}(t_j), t_j) \in \mathcal{P}$: If $k = k_{i^*}$, it follows that the adversary immediately knows i^* . If $k \neq k_{i^*}$, then $\mathcal{A}_{\text{pre}}$ can exclude k as potential k_{i^*} .⁵ To make this formal, we define the event

⁵ This means that our lower bound is not optimal. However, this is no problem since this already gives a negligible advantage.

$E := \{\exists j \in [m_t] : (\text{preIm}_{\text{last}}(t_j), t_j) \in \mathcal{P}\}$. Conditioning on E we can bound the adversary's advantage by

$$\text{Adv}_{\text{BF-ROM}}^{\text{COR}_{s,K,T}}(\mathcal{A}) \leq \Pr[E] + \Pr[\text{COR}_{s,K,T}(\mathcal{A}) \Rightarrow 1 \mid \neg E]. \quad (2)$$

Below, by bounding each summand we will show that

$$\text{Adv}_{\text{BF-ROM}}^{\text{COR}_{s,K,T}}(\mathcal{A}) \leq \frac{m_t P}{2^\kappa} + \frac{T+1}{s} + \text{negl}(\lambda). \quad (3)$$

Before moving to the proof of Equation (3) we show that it implies the lemma's statement. Indeed, by Theorem 10 for any P and any $\gamma > 0$ any adversary $\mathcal{A}$'s advantage in the AI-ROM is bounded by

$$\text{Adv}_{\text{AI-ROM}}^{\text{COR}_{s,K,T}}(\mathcal{A}) \leq \frac{2(S + \log(\gamma^{-1})) \cdot T_{\text{Game}}^{\text{comb}}}{P} + \frac{m_t P}{2^\kappa} + \frac{T+1}{s} + 2\gamma + \text{negl}(\lambda).$$

By choosing $\gamma = 2^{-\lambda}$ and then balancing the two first summand by setting $P = \sqrt{2^{\kappa+1}(S + \lambda)T_{\text{Game}}^{\text{comb}}/m_t}$ we obtain Equation (1), as desired.

It remains to show that Equation (3) holds. For one partial tag t_j the probability that there exists some z such that $(z, t_j) \in \mathcal{P}$ is

$$\Pr[\exists z : (z, t_j) \in \mathcal{P}] \leq \frac{P}{2^{H_\infty(\text{preIm}_{\text{last}}(t_j))}} \leq \frac{P}{2^\kappa}.$$

Using a union bound we can bound the probability that $(\text{preIm}_{\text{last}}(t_j), t_j) \in \mathcal{P}$ for any $j \in [m_t]$.

$$\begin{aligned} \Pr[E] &= \Pr[\exists j \in [m_t] : (\text{preIm}_{\text{last}}(t_j), t_j) \in \mathcal{P}] \\ &\leq \sum_{j=1}^{m_t} \Pr[(\text{preIm}_{\text{last}}(t_j), t_j) \in \mathcal{P}] \leq \frac{m_t P}{2^\kappa}. \end{aligned}$$

To bound the second term of Equation (2), note that conditioned on $\neg E$ from the adversary's view every partial tag t_j with $j \in [m_t]$ is a uniformly random element of $\{0, 1\}^\lambda$ and, in particular, independent from i^* and the set of candidate keys K , unless adversary $\mathcal{A}_{\text{main}}$ queries the random oracle RO on input $\text{preIm}_{\text{last}}(t_j)$. Consider the T queries to the random oracle made by $\mathcal{A}_{\text{main}}$. We now define the set of indices $i \in [s]$ for which the adversary after making its T random oracle queries knows whether computing a tag with key k_i and randomness r will correspond to the challenge it received. I.e., we define

$$\begin{aligned} I &= \{i \in [s] \mid \exists j \in [m_t] : (\hat{t}_1, \dots, \hat{t}_{m_t}, r) = \text{T.tag}(k_i; r) \\ &\quad \wedge \exists j \in [m_t] : \mathcal{A}_{\text{main}} \text{ queries for RO}(\text{preIm}_{\text{last}}(\hat{t}_j))\}. \end{aligned}$$

If the computation of all $\text{T.tag}(k_i; r)$ for $i \in s$ does not contain any random-oracle collisions, which happens with overwhelming probability, we have $|I| \leq \min(T, s)$. If $i^* \in I$ then $\mathcal{A}_{\text{main}}$ might have recomputed a partial tag of $\text{T.tag}(k_{i^*}; r)$

and seeing that it matches the corresponding element t_j of challenge learned i^* . On the other hand, if $i^* \notin I$ the adversary still might be able to exclude particular indices; for $i \in I$ it might have recomputed $T.\text{tag}(k_i; r)$ and observed that it differs from the corresponding t_i . Importantly, if $i^* \notin I$, then from the adversary's view it must be still be distributed uniformly random in $[s] \setminus I$, and in turn any remaining candidate for k_{i^*} is equally probable. Thus, denoting the event that game $\text{COR}_{s,K,T}(\mathcal{A})$ outputs 1 conditioned on $\neg E$ by win we obtain

$$\begin{aligned} \Pr[\text{win}] &\leq \Pr[i^* \in I] + \Pr[i^* \notin I] \cdot \Pr[\text{win} \mid i^* \notin I] \\ &\leq \frac{|I|}{s} + \left(1 - \frac{|I|}{s}\right) \frac{1}{s - |I|} + \text{negl}(\lambda) \\ &= \frac{|I| + 1}{s} + \text{negl}(\lambda) \leq \frac{T + 1}{s} + \text{negl}(\lambda), \end{aligned}$$

where for the first summand we used that i^* is sampled uniformly random from $[s]$. This concludes the proof of Equation (3). $\square$

Proof (Theorem 11). Observe that with overwhelming probability the keys in K are pairwise distinct and we thus by Lemma 2 can apply Lemma 3 with super-polynomial 2^κ . $\square$

8 Redundancy vs. Combiners

Our definitions provide redundant security by combining symmetric and asymmetric primitives, maintaining security as long as (the secret of) either component remains uncorrupted. The setting resembles that of a combiner, however, there is a subtle difference, which makes our definitions stronger. We elaborate further below and first recall the goals of traditional combiners.

Classical/post-quantum combiners. For the transition to post-quantum security, the conservative approach is to combine two schemes in a hybrid fashion, where one scheme is based on classical and one is based on post-quantum assumptions. Then, if any of the two schemes fails, security is still guaranteed by the other one. More generally, such combiners have been studied for KEMs [GHP18], authenticated encryption [PR20], ordinary AKEMs [GHJ25] and other primitives. Such combiners typically aim to construct the same primitive as their underlying components, even though sometimes in a non-blackbox way. For combined security guarantees, we then want to show that the advantage of an adversary in breaking the combined scheme is bounded by the minimum of the advantages of breaking any of the underlying schemes.

It is important to note that in this setting, the final construction is oblivious of the underlying structure. For example, the result of a KEM combiner is still a KEM and not syntactically aware of the two components. Most importantly, the secret key is the combination of the two individual secret keys; therefore, the corruption of the secret key will always trivially break security of the scheme. Even if we allowed the adversary to corrupt only one component, such a model

seems to undermine the purpose of the combiner because it does not allow us to bound the advantage by the minimum. To see this, note that the adversary can simply corrupt the secret key of the secure scheme and then leverage the insecurity of the other scheme to break the combined scheme.

A recent work by Millerjord, Stebila and Steckel [MSS26] studies how to strengthen security notions for combiners. They consider stronger notions for KEM combiners specifically, where the primitive (called a compound KEM) and security notions explicitly capture the hybrid structure. They then study security in the presence of an individual decryption oracle for each underlying KEM. Due to the above, it seems unlikely that one can extend this further to consider corruptions in a meaningful way.

Redundancy. For redundant security guarantees, we also combine two (or more) components, in our case symmetric and asymmetric primitives. However, the goal is not to prove that security should hold as long as at least one component is secure, but rather that security should hold as long as one secret (asymmetric/symmetric) remains uncorrupted. This is conceptually different to the traditional combiner and requires the strong security notion that we define.

More concretely, we want to explicitly capture the case where for one user the asymmetric secret is corrupted and for another user the symmetric secret could be corrupted. We note that this resembles the definitional approach taken in papers on symmetric signcryption, e.g., [JKS24]. While it is indeed stronger than combiner security, it is reasonable for the applications that we and prior work target, e.g., for messaging applications, where symmetric keys are shared across users and may be corrupted independently of other secrets such as a party’s long-term secret key. At the same time, if a long-term secret is corrupted, we still hope to get security from the symmetric key. Compared to the traditional combiner, we inherently want to capture a multi-user setting where we want to protect against different (adaptive) corruption scenarios.

It is possible to relax our security definitions to only let the adversary win if both symmetric and asymmetric secrets of the challenge remain uncorrupted. However, this does not seem to be sufficient for the targeted applications.

References

- AAP⁺23. Shashank Agrawal, Shweta Agrawal, Manoj Prabhakaran, Rajeev Raghunath, and Jayesh Singla. CASE: A new frontier in public-key authenticated encryption. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part II*, volume 14370 of *LNCS*, pages 189–219. Springer, Cham, November / December 2023.
- ABH⁺21. Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the HPKE standard. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 87–116. Springer, Cham, October 2021.
- ABR01. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, edi-

- tor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Berlin, Heidelberg, April 2001.
- AJKL23. Joël Alwen, Jonas Janneck, Eike Kiltz, and Benjamin Lipp. The pre-shared key modes of HPKE. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 329–360. Springer, Singapore, December 2023.
- ARR⁺26. Benedikt Auerbach, Doreen Riepel, Paul Rösler, Lea Thiemt, and Julian Thomas. Symmetric and asymmetric anonymous authenticated kem. In *Advances in Cryptology - ASIACRYPT 2026 - 32th International Conference on the Theory and Application of Cryptology and Information Security*, Lecture Notes in Computer Science, 2026.
- BBLW22. Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. Hybrid Public Key Encryption. RFC 9180, February 2022.
- BCP⁺23. Hugo Beguinet, Céline Chevalier, David Pointcheval, Thomas Ricosset, and Mélissa Rossi. GeT a CAKE: Generic transformations from key encapsulation mechanisms to password authenticated key exchanges. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 2023, Part II*, volume 13906 of *LNCS*, pages 516–538. Springer, Cham, June 2023.
- BFG⁺20. Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. Towards post-quantum security for Signal’s X3DH handshake. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O’Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 404–430. Springer, Cham, October 2020.
- BLMQ05. Paulo S. L. M. Barreto, Benoît Libert, Noel McCullagh, and Jean-Jacques Quisquater. Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 515–532. Springer, Berlin, Heidelberg, December 2005.
- Boy03. Xavier Boyen. Multipurpose identity-based signcryption (a swiss army knife for identity-based cryptography). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 383–399. Springer, Berlin, Heidelberg, August 2003.
- BRT23. Alexander Bienstock, Paul Rösler, and Yi Tang. ASMesh: Anonymous and secure messaging in mesh networks using stronger, anonymous double ratchet. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1–15. ACM Press, November 2023.
- BS20. Mihir Bellare and Igors Stepanovs. Security under message-derived keys: Signcryption in iMessage. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 507–537. Springer, Cham, May 2020.
- CCH25. Daniel Collins, Simone Colombo, and Loïs Huguenin-Dumittan. Real-world deniability in messaging. *PoPETs*, 2025(1):320–340, January 2025.
- CDGS18. Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 227–258. Springer, Cham, April / May 2018.
- CM05. Liqun Chen and John Malone-Lee. Improved identity-based signcryption. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 362–379. Springer, Berlin, Heidelberg, January 2005.

- FDC⁺07. Hal Finney, Lutz Donnerhacke, Jon Callas, Rodney L. Thayer, and Daphne Shaw. OpenPGP Message Format. RFC 4880, November 2007.
- FG25. Rune Fiedler and Felix Günther. Security analysis of Signal’s PQXDH handshake. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part II*, volume 15675 of *LNCS*, pages 137–169. Springer, Cham, May 2025.
- FGPZ25. Rune Fiedler, Felix Günther, Jiaxin Pan, and Runzhi Zeng. XHMQV: Better efficiency and stronger security for signal’s initial handshake based on HMQV. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VIII*, volume 16007 of *LNCS*, pages 109–140. Springer, Cham, August 2025.
- FKMS20. Scott Fluhrer, Panos Kampanakis, David McGrew, and Valery Smyslov. Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security. RFC 8784, June 2020.
- FO99. Eiichiro Fujisaki and Tatsuoaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999.
- GHJ25. Phillip Gajland, Vincent Hwang, and Jonas Janneck. Shadowfax: A deniability-preserving AKEM combiner. Cryptology ePrint Archive, Report 2025/154, 2025.
- GHP18. Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 190–218. Springer, Cham, March 2018.
- GV22. Rishab Goyal and Vinod Vaikuntanathan. Locally verifiable signature and key aggregation. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 761–791. Springer, Cham, August 2022.
- HHK17. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Cham, November 2017.
- HKKP22. Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. An efficient and generic construction for Signal’s handshake (X3DH): Post-quantum, state leakage secure, and deniable. *Journal of Cryptology*, 35(3):17, July 2022.
- HKW25. Keitaro Hashimoto, Shuichi Katsumata, and Thom Wiggers. Bundled authenticated key exchange: A concrete treatment of signal’s handshake protocol and post-quantum security. In Lujo Bauer and Giancarlo Pellegrino, editors, *USENIX Security 2025*, pages 6777–6796. USENIX Association, August 2025.
- IT21. Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.
- JK25. Joseph Jaeger and Akshaya Kumar. Analyzing group chat encryption in MLS, session, signal, and matrix. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VIII*, volume 15608 of *LNCS*, pages 272–301. Springer, Cham, May 2025.
- JKS24. Joseph Jaeger, Akshaya Kumar, and Igors Stepanovs. Symmetric sign-encryption and E2EE group messaging in keybase. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*, pages 283–312. Springer, Cham, May 2024.

- LGR22. Julia Len, Paul Grubbs, and Thomas Ristenpart. Authenticated encryption with key identification. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 181–209. Springer, Cham, December 2022.
- LP25. Jacob Leiken and Sunoo Park. On the credibility of deniable communication in court. Cryptology ePrint Archive, Report 2025/1949, 2025.
- LQ04. Benoît Libert and Jean-Jacques Quisquater. Efficient signcryption with key privacy from gap Diffie-Hellman groups. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *PKC 2004*, volume 2947 of *LNCS*, pages 187–200. Springer, Berlin, Heidelberg, March 2004.
- Lun18. Joshua Lund. Sealed sender for signal. <https://signal.org/blog/sealed-sender/>, 2018.
- MP16. Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol. <https://signal.org/docs/specifications/x3dh/>, 2016.
- MSS26. Lise Millerjord, Douglas Stebila, and Camryn Steckel. Split-key PRFs and extended hybrid security for KEM combiners. *IACR Communications in Cryptology*, 2(4), 2026.
- Per18. Trevor Perrin. The noise protocol framework. <http://www.noiseprotocol.org/noise.html>, 2018.
- PR20. Bertram Poettering and Paul Rösler. Combiners for AEAD. *IACR Trans. Symm. Cryptol.*, 2020(1):121–143, 2020.
- Rog15. Phillip Rogaway. The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, 2015.
- Smy25. Valery Smyslov. Mixing Preshared Keys in the IKE_INTERMEDIATE and CREATE_CHILD_SA Exchanges of the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-Quantum Security. RFC 9867, November 2025.
- SRT19. Jim Schaad, Blake C. Ramsdell, and Sean Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification. RFC 8551, April 2019.
- Tho26. Julian Thomas. Anonymous akem reference implementation. <https://github.com/3Komma1415/sym-asym-anon-auth-kem-prototype>, 2026.
- TRB⁺25. Lea Thiemt, Paul Rösler, Alexander Bienstock, Rolfe Schmidt, and Yevgeniy Dodis. Generic anonymity wrapper for messaging protocols. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025*, pages 350–364. ACM Press, October 2025.
- Unr07. Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 205–223. Springer, Berlin, Heidelberg, August 2007.
- WHWY24. Paul Wouters, Daniel Huigens, Justus Winter, and Niibe Yutaka. OpenPGP. RFC 9580, July 2024.
- WMAS13. Yang Wang, Mark Manulis, Man Ho Au, and Willy Susilo. Relations among privacy notions for signcryption and key invisible “Sign-then-Encrypt”. In Colin Boyd and Leonie Simpson, editors, *ACISP 13*, volume 7959 of *LNCS*, pages 187–202. Springer, Berlin, Heidelberg, July 2013.
- Zhe97. Yuliang Zheng. Digital signcryption or how to achieve $\text{cost}(\text{signature} \ \& \ \text{encryption}) \ll \text{cost}(\text{signature}) + \text{cost}(\text{encryption})$. In Burton S. Kaliski, Jr., editor, *CRYPTO’97*, volume 1294 of *LNCS*, pages 165–179. Springer, Berlin, Heidelberg, August 1997.

A Additional Preliminaries

Pseudorandom Functions. We define a pseudorandom function (PRF) $F : \mathcal{K} \times (\mathcal{L} \cup \{\perp\}) \times \{0, 1\}^* \rightarrow \mathcal{OUT}$ as a keyed function that maps from a key space $\mathcal{K}$, a (possibly empty) label space $\mathcal{L}$ and an arbitrary long input bitstring to an output space $\mathcal{OUT}$. In our constructions, labels (written in typewriter font $\mathbf{tt}$) are used for domain-separation. The output space is usually another key space, therefore mostly also denoted by $\mathcal{K}$. In our constructions, we sometimes use pre-shared keys as input keys for the same PRF, in which case we implicitly assume that the key spaces match. In some instances, a longer output is required when using F with a specific label. For simplicity, we do not make that explicit, but we note that this can be easily be done by additionally applying a pseudorandom generator.

Pseudorandomness of a PRF F is defined via an indistinguishability game in which an adversary $\mathcal{A}$ can generate multiple keys and query an oracle on pairs (i, m) , where i is the index to a PRF key and m is a message. (We include labels inside messages here.) We define the advantage against the pseudorandomness of F as

$$\text{Adv}_F^{\text{IND}}(\mathcal{A}) := |\Pr[\text{IND}_F^0(\mathcal{A}) \Rightarrow 1] - \Pr[\text{IND}_F^1(\mathcal{A}) \Rightarrow 1]|.$$

Game $\text{IND}_F^b(\mathcal{A})$	Oracle $\text{Eval}(i, m)$
00 $n \leftarrow 0$	16 Require $i \in [n]$
01 $E, CH, X \leftarrow \emptyset^3$	17 Require $(i, m) \notin CH$
02 $b' \leftarrow_{\$} \mathcal{A}$	18 $k \leftarrow F(k_i, m)$
03 For all $(i, m) \in CH$:	19 $E \leftarrow \{(i, m)\}$
04 Require $i \notin X$	20 Return k
05 Stop with b'	
Oracle $\text{Gen}(k)$	Oracle $\text{Chall}(i, m)$
06 $n \leftarrow n + 1$	21 Require $i \in [n]$
07 If $k \neq \perp$:	22 Require $(i, m) \notin CH \cup E$
08 Require $\nexists i \in X : k_i = k$	23 $k_0 \leftarrow F(k_i, m)$
09 $k_n \leftarrow k$	24 $k_1 \leftarrow_{\$} \mathcal{K}$
10 $X \leftarrow \{n\}$	25 $CH \leftarrow \{(i, m)\}$
11 Else: $k_n \leftarrow_{\$} \mathcal{K}$	26 Return k_b
12 Return	
Oracle $\text{Corrupt}(i)$	
13 Require $i \in [n]$	
14 $X \leftarrow \{i\}$	
15 Return k_i	

Fig. 11. Game IND for pseudo-random function F .

Game $\text{SUF}_{\text{SIG}}(\mathcal{A})$	Oracle $\text{Sig}(i, m)$
00 $n \leftarrow 0$	10 Require $i \in [n]$
01 $MS, X \leftarrow \emptyset^2$	11 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_i, m)$
02 Invoke $\mathcal{A}$	12 $MS \leftarrow MS \cup \{(i, m, \sigma)\}$
03 Stop with 0	13 Return σ
Oracle Gen	Oracle $\text{Vfy}(i, m, \sigma)$
04 $n \leftarrow n + 1$	14 $b \leftarrow \text{SIG.vfy}(vk_i, m, \sigma)$
05 $(sk_n, vk_n) \leftarrow_{\$} \text{SIG.gen}$	15 If $(i, m, \sigma) \notin MS \wedge i \notin X \wedge b$:
06 Return vk_n	16 Stop with 1
Oracle $\text{Corrupt}(i)$	17 Return b
07 Require $i \in [n]$	
08 $X \leftarrow \{i\}$	
09 Return sk_i	

Fig. 12. Game SUF for signature scheme SIG .

Signature Schemes. A signature scheme SIG with secret key space $\mathcal{SK}$, public key space $\mathcal{PK}$, message space $\mathcal{M}$, signature space Σ , and randomness space $\mathcal{R}$ consists of three algorithms SIG.gen , SIG.sig , and SIG.vfy with the following syntax:

Game $\text{IND}_{\text{KEM}}^b(\mathcal{A})$	Oracle $\text{Enc}(i)$
00 $n \leftarrow 0; XP \leftarrow \emptyset$	11 Require $i \in [n]$
01 $b' \leftarrow_{\$} \mathcal{A}$	12 Require $i \notin XP$
02 Stop with b'	13 $(k_0, c) \leftarrow_{\$} \text{KEM.enc}(pk_i)$
Oracle Gen	14 $k_1 \leftarrow_{\$} \mathcal{K}$
03 $n \leftarrow n + 1$	15 $CK_i \leftarrow^{\perp} \{c\}$
04 $(pk_n, sk_n) \leftarrow_{\$} \text{KEM.gen}$	16 Return (k_b, c)
05 $CK_n \leftarrow \emptyset$	Oracle Dec(i, c)
06 Return pk_n	17 Require $i \in [n]$
Oracle Expose(i)	18 Require $c \notin CK_i$
07 Require $i \in [n]$	19 $k \leftarrow \text{KEM.dec}(sk_i, c)$
08 Require $CK_i = \emptyset$	20 Return k
09 $XP \leftarrow^{\perp} \{i\}$	
10 Return sk_i	

Fig. 13. Game IND for KEM scheme KEM.

Game $\text{OW-KCA}_{\text{KEM}}(\mathcal{A})$	Oracle $\text{Enc}(i)$
00 $n \leftarrow 0; XP \leftarrow \emptyset$	10 Require $i \in [n]$
01 Invoke $\mathcal{A}$	11 $(k, c) \leftarrow_{\$} \text{KEM.enc}(pk_i)$
02 Stop with 0	12 $CK_i \leftarrow^{\perp} \{(c, k)\}$
Oracle Gen	13 Return c
03 $n \leftarrow n + 1$	Oracle Check(i, c, k)
04 $(pk_n, sk_n) \leftarrow_{\$} \text{KEM.gen}$	14 Require $i \in [n]$
05 $CK_n \leftarrow \emptyset$	15 If $\exists k' : (c, k') \in CK_i \wedge k = k'$:
06 Return pk_n	16 If $i \notin XP$: Stop with 1
Oracle Expose(i)	17 $k' \leftarrow \text{KEM.dec}(sk_i, c)$
07 Require $i \in [n]$	18 Return $\llbracket k = k' \rrbracket$
08 $XP \leftarrow^{\perp} \{i\}$	
09 Return sk_i	

Fig. 14. Game OW-KCA for KEM scheme KEM.

- $\text{SIG.gen}: \emptyset \rightarrow_{\$} \mathcal{SK} \times \mathcal{PK}$: Outputs a secret key sk and a public key pk .
- $\text{SIG.sig}: \mathcal{SK} \times \mathcal{M} \times \mathcal{R} \rightarrow \Sigma$ resp. $\text{SIG.sig}: \mathcal{SK} \times \mathcal{M} \rightarrow_{\$} \Sigma$: Outputs a signature σ for message m .
- $\text{SIG.vfy}: \mathcal{PK} \times \mathcal{M} \times \Sigma \rightarrow \{\text{tru}, \text{fal}\}$: Outputs **tru** if σ is a valid signature for m , and **fal** otherwise.

For the correctness of a signature scheme SIG, we require that for all $(sk, pk) \leftarrow \text{SIG.gen}$ and all $m \in \mathcal{M}$, it holds that $\Pr[\text{SIG.vfy}(pk, m, \text{SIG.sig}(sk, m)) = \text{tru}] = 1$. We use the signature security notion of strong unforgeability under chosen message attacks suf. In this game, an adversary $\mathcal{A}$ is given a public key pk and access to a signing oracle $\text{SIG.sig}(sk, \cdot)$. The adversary wins if it can output a forgery (m, σ) such that $\text{SIG.vfy}(pk, m, \sigma) = \text{tru}$ and the pair (m, σ) was not previously returned by the signing oracle. Following this, for all $\mathcal{A}$ in game SUF from Figure 12, we define the advantage as

$$\text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{A}) := \Pr[\text{SUF}_{\text{SIG}}(\mathcal{A}) \Rightarrow 1].$$

Omitted KEM Security Notions. Security game IND capturing multi-user security under chosen ciphertext attacks and corruptions can be found in Figure 13. Game OW-KCA, which captures one-wayness under key checking attacks in Figure 14. We define the advantage of an adversary $\mathcal{A}$ as

$$\text{Adv}_{\text{KEM}}^{\text{IND}}(\mathcal{A}) := |\Pr[\text{IND}_{\text{KEM}}^0(\mathcal{A}) \Rightarrow 1] - \Pr[\text{IND}_{\text{KEM}}^1(\mathcal{A}) \Rightarrow 1]|,$$

and

$$\text{Adv}_{\text{KEM}}^{\text{OW-KCA}}(\mathcal{A}) := \Pr[\text{OW-KCA}_{\text{KEM}}(\mathcal{A}) \Rightarrow 1].$$

B Proof of Lemma 1

Proof. We prove IND (see Figure 6) of our construction from Figure 6 with reduction towards the IND security of the underlying PRF (see Figure 11). Before replacing the PRF, we first exclude nonce collisions via a standard bad-event hop. We do this, since the adversary sees the nonces, and in case of a collision during challenges, the probability of having the same output (under a nonce collision) for $b = 0$ is 1, whereas for $b = 1$ (fresh random) this probability is negligible. We denote the games as G_i for $i \in [3]$.

Game G_1^b is the original $\text{IND}_T^b(\mathcal{A})$ experiment instantiated with Tagging Mechanism T from Figure 6. In particular $\Pr[G_1^b \Rightarrow 1] = \Pr[\text{IND}_T^b(\mathcal{A}) \Rightarrow 1]$.

Game G_2^b is identical to G_1^b except that it aborts if a nonce repeats. Let q be the number of tag generations, and let E_n be the bad event of collision. Since n is sampled uniformly at random from $\mathcal{K}$, we have the standard birthday bound $\Pr[E_n] \leq q^2/|\mathcal{K}|$. Since G_1^b and G_2^b are identical until E_n occurs, for each $b \in \{0, 1\}$ we have

$$|\Pr[G_1^b \Rightarrow 1] - \Pr[G_2^b \Rightarrow 1]| \leq \frac{q^2}{|\mathcal{K}|}.$$

Game G_3 is identical to G_2^0 except that in every uncorrupted computation of $T.\text{tag}$ the computation of $x \leftarrow F(k, (\text{tag}, n))$ is replaced with the evaluation of a uniformly random function (per key).

G_2^0 to G_3 : We construct a reduction $\mathcal{R}_F$ against IND_F^b (cf. Figure 11). Reduction $\mathcal{R}_F$ runs $\mathcal{A}$ and simulates the IND_T experiment as follows. $\mathcal{R}_F$ internally stores each key index created in the T game as the corresponding key index in its F game by forwarding $T.\text{Gen}$ and $T.\text{Corrupt}$ queries to $F.\text{Gen}$ and $F.\text{Corrupt}$, respectively. Furthermore, $\mathcal{R}_F$ maintains a table T . On any query of $\mathcal{A}$ that requires a tag under some (uncorrupted) key index i , $\mathcal{R}_F$ samples $n \leftarrow_{\$} \mathcal{K}$ and computes $x \leftarrow F.\text{Chall}(\{i\}, (\text{tag}, n))$, sets $t \leftarrow (n, x)$ and returns t , and stores $T[(i, n)] \leftarrow x$.

All other computations are done as in Figure 6 with the following specification: when $T.\text{det}$ needs a value $x' = F(k_i, (\text{tag}, n))$, $\mathcal{R}_F$ first checks if $T[(i, n)]$ is defined and in that case sets $x' \leftarrow T[(i, n)]$; otherwise it uses $F.\text{Eval}(\{i\}, (\text{tag}, n))$ for the relevant indices i (or computes locally if the key is known due to corruption).

Eventually, $\mathcal{A}$ outputs a bit b' and $\mathcal{R}_F$ outputs the same bit. If the IND_F bit is 0, then $F.\text{Chall}$ returns $F(k_i, (\text{tag}, n))$ and the simulation is as in G_2^0 . Otherwise, $F.\text{Chall}$ returns uniform values (with consistency ensured by T) and the simulation is as in G_3 . Consequently,

$$|\Pr[G_2^0 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_F).$$

Finally, note that G_3 differs from G_2^1 only in that G_3 answers tags via the PRF IND challenger. Conditioned on not aborting (i.e., on $\neg E_n$), all inputs (tag, n) used in tag generations are distinct, hence in the random case the outputs are distributed as fresh uniform values, exactly as in the $b = 1$ tagging experiment. Absorbing the two b -dependent for the bad event hop, yields the stated bound. $\square$

C Security of the Purely Asymmetric AKEM

C.1 Proof of Theorem 1

Theorem 1. *The AKEM construction AK_{a-a} from Figure 7 offers key indistinguishability. More precisely, for every adversary $\mathcal{A}$ against AK_{a-a} in game IND from Figure 3, there exist a reduction $\mathcal{R}_{\text{KEM}}$ in game IND from Figure 13 against the KEM KEM and a reduction $\mathcal{R}_F$ in game IND from Figure 11 against the PRF F, such that*

$$\text{Adv}_{\text{AK}_{a-a}}^{\text{IND}_{a-a}}(\mathcal{A}) \leq 2 \cdot \left(\text{Adv}_{\text{KEM}}^{\text{IND}}(\mathcal{R}_{\text{KEM}}) + \text{Adv}_F^{\text{IND}}(\mathcal{R}_F) \right)$$

and the running time of $\mathcal{R}_{\text{KEM}}$ and $\mathcal{R}_F$ is about that of $\mathcal{A}$.

Proof. We prove the key indistinguishability of Figure 7 with a sequence of games, as shown in Figure 15. The first hop uses a reduction to IND security (see Figure 13) of the KEM scheme, allowing us to replace the computation of the KEM key k_{KEM} with uniform sampling. In the second hop, we replace the PRF computation (for which we can now assume the key k_{KEM} is sampled uniformly at random) with uniform random sampling from the same output space utilizing a reduction towards the IND security (see Figure 11). As a result, both keys that the adversary has to distinguish between them and are drawn uniformly at random in the final game. We denote the games as G_i for $i \in [3]$.

Game G_1^b-G_3^b	Oracle $\text{Dec}(r, PK, ad, c)$
00 $(n, m) \leftarrow 0^2$; $sk, pk \leftarrow \perp^2$	30 Require $(r, PK) \in [n] \times \mathcal{P}(PK)$
01 $CH, XA \leftarrow \emptyset^2$	31 If $\exists (s', r, ad, c) \in CH : (s' = \perp \vee pk_{s'} \in PK)$: Return $\perp$
02 $T[\cdot, \cdot] \leftarrow \perp$	G_2 - G_3 32 If $\exists pk \in PK, k' : (pk, r, c, ad, k') \in D$: Return k' G_3
03 $D \leftarrow \emptyset$	G_2 - G_3 33 $(sk_{\text{KEM}}, \_, \_) \leftarrow sk_r$; $(ck_{\text{KEM}}, c') \leftarrow c$
04 $b' \leftarrow_{\$} \mathcal{A}$	34 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, ck_{\text{KEM}})$ G_1
05 For all $(s, r, ad, c) \in CH$:	35 If $T[r, ck_{\text{KEM}}] = \perp$: G_2 - G_3
06 Require $\perp \neq r \notin XA$	36 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, ck_{\text{KEM}})$ G_2 - G_3
07 Stop with b'	37 Else: $k_{\text{KEM}} \leftarrow T[r, ck_{\text{KEM}}]$ G_2 - G_3
Oracle $\text{Enc}(s, pk, ad, ch)$	38 $k_{\text{otp}} \leftarrow F(k_{\text{KEM}}, \text{otp})$
08 Require $(s, pk) \in [n] \times \mathcal{PK}$	39 $(h, \sigma) \leftarrow c' \oplus k_{\text{otp}}$
09 $(\_, sk_{\text{SIG}}, h) \leftarrow sk_s$	40 $pk \leftarrow \{pk' \in PK \mid H(pk') = h\}$
10 $(pk_{\text{KEM}}, \_) \leftarrow pk$	41 $(\_, pk_{\text{SIG}}) \leftarrow pk$
11 If $\exists r' \in [n] : pk_{r'} = pk$: $r \leftarrow r'$; Else: $r \leftarrow \perp$	42 $b \leftarrow \text{SIG.vfy}(pk_{\text{SIG}}, (ck_{\text{KEM}}, h, ad), \sigma)$
12 $(k_{\text{KEM}}, ck_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$	43 If $b = \text{fail} \vee pk = \emptyset$: Return $\perp^3$
13 If $ch \wedge r \in [n] \setminus XA$:	44 $k \leftarrow F(k_{\text{KEM}}, (\text{key}, pk, \mu(sk_r), c, ad))$ G_1 - G_2
14 $k_{\text{KEM}} \leftarrow_{\$} \mathcal{K}$	G_2 - G_3 45 If $T[r, ck_{\text{KEM}}] = \perp$: G_3
15 $T[r, ck_{\text{KEM}}] \leftarrow k_{\text{KEM}}$	G_2 - G_3 46 $k \leftarrow F(k_{\text{KEM}}, (\text{key}, pk, \mu(sk_r), c, ad))$ G_3
16 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_{\text{SIG}}, (ck_{\text{KEM}}, h, ad))$	G_2 - G_3 47 Else: G_3
17 $k_{\text{otp}} \leftarrow F(k_{\text{KEM}}, \text{otp})$	48 $k \leftarrow_{\$} \mathcal{K}$ G_3
18 $c \leftarrow (ck_{\text{KEM}}, (h, \sigma) \oplus k_{\text{otp}})$	49 $D \leftarrow \{(pk, r, c, ad, k)\}$ G_3
19 $k_0 \leftarrow F(k_{\text{KEM}}, (\text{key}, \mu(sk_s), pk, c, ad))$	50 Return (pk, k)
20 If $ch \wedge r \in [n] \setminus XA$:	G_3 Oracle $\text{Gen}(sk)$
21 $k_0 \leftarrow_{\$} \mathcal{K}$	G_3 51 Require $\nexists u \in [n] : pk_u = \mu(sk)$
22 $k_1 \leftarrow_{\$} \mathcal{K}$	52 $n \leftarrow n + 1$
23 If ch :	53 If $sk \in \mathcal{SK}$: $\backslash$ this already checks hash
24 $CH \leftarrow \{(s, r, ad, c)\}$	54 $(sk_n, pk_n) \leftarrow (sk, \mu(sk))$
25 Return (k_b, c)	55 $XA \leftarrow \{n\}$
26 Return (k_0, c)	56 Else:
Oracle $\text{CorruptA}(u)$	57 $pk_{\text{KEM}} \leftarrow \text{IND.Gen}$
27 Require $u \in [n]$	58 $(sk_{\text{SIG}}, pk_{\text{SIG}}) \leftarrow_{\$} \text{SIG.gen}$
28 $XA \leftarrow \{u\}$	59 $pk_n \leftarrow (pk_{\text{KEM}}, pk_{\text{SIG}})$
29 Return sk_u	60 $sk_n \leftarrow (sk_{\text{KEM}}, sk_{\text{SIG}}, H(pk))$
	61 Return pk_n

Fig. 15. Games for the proof of Theorem 1.

Game G_1 is the original $\text{IND}_{\text{AK}_{\text{a-a}}}^{b, \text{a}, \text{a}}$ game instantiated with our construction from Figure 7, where we have removed the parts that are only necessary for $\mathbf{s} \in \text{CF} \cup \text{AU}$, including the index i . In particular,

$$\Pr[G_1^b \Rightarrow 1] = \Pr[\text{IND}_{\text{AK}_{\text{a-a}}}^{b, \text{a}, \text{a}}(\mathcal{A}) \Rightarrow 1]$$

and

$$\text{Adv}_{\text{AK}_{\text{a-a}}}^{\text{IND}_{\text{a-a}}}(\mathcal{A}) = |\Pr[G_1^0 \Rightarrow 1] - \Pr[G_1^1 \Rightarrow 1]|.$$

Game G_2 is identical to G_1 except that we now sample the KEM key k_{KEM} uniformly at random for challenge queries to Enc whenever $r \in [n] \notin XA$. Note that by the end of the game we require exactly this for challenges. We store k_{KEM} along with the index r in a table T to change the decryption oracle accordingly.

G_1 to G_2 : We construct a reduction $\mathcal{R}_{\text{KEM}}^b$ against the IND security of the KEM scheme (cf. Figure 13), where $\mathcal{R}_{\text{KEM}}^b$ is parameterized via b , just as the games. $\mathcal{R}_{\text{KEM}}^b$ runs adversary $\mathcal{A}$ as a subroutine and simulates Figure 3 as follows:

- On query from $\mathcal{A}$ to Gen , where $\mathcal{A}$ asks to generate an honest key pair, $\mathcal{R}_{\text{KEM}}^b$ queries $\text{IND}_{\text{KEM}}.\text{Gen}$ for the KEM part and samples the signature key pair itself via SIG.gen .

Technically, this creates a mismatch between indices in the KEM game and those in the AKEM game, because in the latter, the adversary may also create dishonest users. However, we note that the corresponding mapping can be easily implemented by the reduction, so in the following, we assume that it always converts indices before forwarding them to the KEM oracles.

- On query from $\mathcal{A}$ to $\text{CorruptA}(u)$, reduction $\mathcal{R}_{\text{KEM}}^b$ forwards to $\text{IND}_{\text{KEM}}.\text{Expose}(u)$ to obtain sk_{KEM} for u , retrieves the locally stored sk_{SIG} , and returns $sk_u = (sk_{\text{KEM}}, sk_{\text{SIG}})$.
- On query from $\mathcal{A}$ to Enc , reduction $\mathcal{R}_{\text{KEM}}^b$ parses $pk = (pk_{\text{KEM}}, pk_{\text{SIG}})$, determines r and queries its challenge oracle whenever the condition from G_2 holds, in which case we simulate as in G_2 using the table T to store the KEM challenge key. Otherwise, it runs the encapsulation locally and completes $\text{AK}_{a-a}.\text{enc}$ honestly.
- On query from $\mathcal{A}$ to Dec , reduction $\mathcal{R}_{\text{KEM}}^b$ first applies the same reject condition as in G_2 . If the KEM part of the ciphertext is the same as one that was output by Enc in a challenge query for the same r , then it uses the key stored in T . Otherwise, it computes k_{KEM} by either querying $\text{IND}_{\text{KEM}}.\text{Dec}$ (if r is an honestly generated user) or by decapsulating locally (if r 's secret key is known to $\mathcal{R}_{\text{KEM}}^b$).

Eventually, $\mathcal{A}$ outputs b' and $\mathcal{R}_{\text{KEM}}^b$ outputs b' . If $\mathcal{R}_{\text{KEM}}^b$ is in game $\text{IND}_{\text{KEM}}^0$, then challenges that the reduction receives are real encapsulated keys and the simulation is as in G_1^b ; if it is in game $\text{IND}_{\text{KEM}}^1$, it receives uniformly random keys and the simulation is as G_2^b . Moreover, the corruption restrictions are compatible in the sense that whenever $\mathcal{A}$ respects them, $\mathcal{R}_{\text{KEM}}^b$ never makes an invalid query. Hence, for each $b \in \{0, 1\}$ it holds that

$$|\Pr[\text{G}_1^b \Rightarrow 1] - \Pr[\text{G}_2^b \Rightarrow 1]| \leq \text{Adv}_{\text{KEM}}^{\text{IND}}(\mathcal{R}_{\text{KEM}}^b).$$

Game G_3 is identical to G_2 except for challenge queries to $\text{IND}.\text{Enc}$. Instead of computing $k_0 \leftarrow \text{F}(k_{\text{KEM}}, (\text{key}, c, ad))$ for challenge queries, we sample a random key from the output space. We also change the decryption oracle accordingly, also keeping repeated queries consistent.

G_2 to G_3 : We construct a reduction $\mathcal{R}_{\text{F}}^b$ against IND security of F (cf. Figure 11) that uses $\mathcal{A}$ as a subroutine. Reduction $\mathcal{R}_{\text{F}}^b$ has access to the four oracles $\text{IND}_{\text{F}}.\{\text{Gen}, \text{Corrupt}, \text{Eval}, \text{Chall}\}$. It runs $\mathcal{A}$ and simulates as follows. Whenever $\mathcal{R}_{\text{F}}^b$ encounters a fresh pair (r, c_{KEM}) arising from a challenge query with $r \neq \perp$ and $r \notin XA$, it queries $\text{F}.\text{Gen}(\perp)$ to generate a fresh (uncorrupted) PRF key index j and stores $T[r, c_{\text{KEM}}] \leftarrow j$ (instead of the actual key). $\mathcal{R}_{\text{F}}^b$ memorizes all answers returned by $\text{F}.\text{Eval}$ and $\text{F}.\text{Chall}$ and replays them on repeated inputs, so it never queries the PRF challenger twice on the same pair (I, m) . The simulation is as follows:

- On query to Gen or CorruptA , reduction $\mathcal{R}_{\text{F}}^b$ simulates as in G_3^b .
- On query from $\mathcal{A}$ to Enc , if $\mathcal{A}$ asks for a (valid) challenge, $\mathcal{R}_{\text{F}}^b$ calls $\text{IND}_{\text{F}}.\text{Gen}(\perp)$ and stores the current counter value $T[r, c_{\text{KEM}}] \leftarrow j$ (instead of the actual key). It computes the signature and ciphertext (by querying $\text{IND}_{\text{F}}.\text{Eval}$ to derive k_{otp}) and then queries $\text{IND}_{\text{F}}.\text{Chall}$ on j and the corresponding input to compute the final key. For non-challenge queries, $\mathcal{R}_{\text{F}}^b$ simulates locally.
- On query to Dec , $\mathcal{R}_{\text{F}}^b$ works as in G_3^b . If $T[r, c_{\text{KEM}}]$ is defined, it queries $\text{IND}_{\text{F}}.\text{Eval}$ for k_{otp} and $\text{IND}_{\text{F}}.\text{Chall}$ to derive the final session key. Otherwise, $\mathcal{R}_{\text{F}}^b$ simulates locally.

Eventually, $\mathcal{A}$ outputs a bit b' and $\mathcal{R}_{\text{F}}^b$ outputs the same bit b' . If $\mathcal{R}_{\text{F}}^b$ is in game IND_{F}^0 , then it perfectly simulates G_2^b ; if $\mathcal{R}_{\text{F}}^b$ is in game IND_{F}^1 , then it perfectly simulates G_3^b . Thus, $\text{Adv}_{\text{F}}^{\text{IND}}(\mathcal{R}_{\text{F}}^b)$ bound the gap between G_2^b and G_3^b and we have

$$|\Pr[\text{G}_2^b \Rightarrow 1] - \Pr[\text{G}_3^b \Rightarrow 1]| \leq \text{Adv}_{\text{F}}^{\text{IND}}(\mathcal{R}_{\text{F}}^b).$$

Finally, note that G_3^0 and G_3^1 are exactly the same, therefore

$$\Pr[\text{G}_3^0 \Rightarrow 1] = \Pr[\text{G}_3^1 \Rightarrow 1].$$

For each b , the first difference is bounded by the KEM reduction and the second by the PRF reduction. Absorbing the two b -dependent reductions into single reductions $\mathcal{R}_{\text{KEM}}$ and $\mathcal{R}_{\text{F}}$ gives the final bound of the theorem. $\square$

C.2 Proof of Theorem 2

Theorem 2. *The AKEM construction $\text{AK}_{\text{a-a}}$ from Figure 7 offers anonymity. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{a-a}}$ in game ANON from Figure 4, there exist a reduction $\mathcal{R}_{\text{KEM},1}$ in game IND from Figure 13, a reduction $\mathcal{R}_{\text{KEM},2}$ in game ANON from Figure 1, both against the KEM KEM , and a reduction $\mathcal{R}_{\text{F}}$ in game IND from Figure 11 against the PRF F , such that*

$$\text{Adv}_{\text{AK}_{\text{a-a}}}^{\text{ANON}_{\text{a-a}}}(\mathcal{A}) \leq 2 \cdot \left(\text{Adv}_{\text{KEM}}^{\text{IND}}(\mathcal{R}_{\text{KEM},1}) + \text{Adv}_{\text{KEM}}^{\text{ANON}}(\mathcal{R}_{\text{KEM},2}) + \text{Adv}_{\text{F}}^{\text{IND}}(\mathcal{R}_{\text{F}}) \right),$$

where the running time of $\mathcal{R}_{\text{KEM},1}$, $\mathcal{R}_{\text{KEM},2}$ and $\mathcal{R}_{\text{F}}$ is about that of $\mathcal{A}$.

Proof. We prove anonymity of Figure 7 with a sequence of games, as shown in Figure 16. In the first game hop, we replace the computation of the KEM key k_{KEM} with uniform random sampling. As a consequence, the keys for the PRF in line 09 as well as line 11 are uniform, and we can replace both PRF evaluations by random sampling in one game hop. Finally, we replace the computation of the KEM ciphertext c_{KEM} by random sampling, which allows us to replace the whole ciphertext c_0 with random sampling. We denote the games as G_i for $i \in [4]$.

Game G_1^b-G_5^b	Oracle $\text{Dec}(r, PK, ad, c)$
00 $(n, m) \leftarrow 0^2$; $sk, pk \leftarrow \perp^2$; $CH, XA \leftarrow \emptyset^2$	32 Require $(r, PK) \in [n] \times \mathcal{P}(PK)$
01 $T[\cdot, \cdot] \leftarrow \perp$	G_2 - G_5 33 If $\exists (s', r, ad, c) \in CH : pk_{s'} \in PK$: Return $\perp$
02 $T_{\text{otp}}[\cdot, \cdot, \cdot] \leftarrow \perp$; $D \leftarrow \emptyset$	G_3 - G_5 34 If $\exists pk \in PK, k' : (pk, r, c, ad, k') \in D$: Return k' G_3 - G_5
03 $b' \leftarrow_{\$} \mathcal{A}$	35 $(sk_{\text{KEM}}, \_, \_) \leftarrow sk_r$; $(ck_{\text{KEM}}, c') \leftarrow c$
04 For all $(s, r, ad, c) \in CH$:	36 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, ck_{\text{KEM}})$ G_1
05 Require $\perp \neq r \notin XA$	37 If $T[r, ck_{\text{KEM}}] = \perp$: G_2 - G_5
06 Stop with b'	38 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, ck_{\text{KEM}})$ G_2 - G_5
Oracle $\text{Enc}(s, pk, ad, ch)$	39 Else: $k_{\text{KEM}} \leftarrow T[r, ck_{\text{KEM}}]$ G_2 - G_5
07 Require $(s, pk) \in [n] \times \mathcal{PK}$	40 $k_{\text{otp}} \leftarrow \text{F}(k_{\text{KEM}}, \text{otp})$ G_1 - G_2
08 $(\_, sk_{\text{SIG}}, h) \leftarrow sk_s$; $(pk_{\text{KEM}}, \_) \leftarrow pk$	41 If $T_{\text{otp}}[r, ck_{\text{KEM}}] = \perp$: G_3 - G_5
09 If $\exists r' \in [n] : pk_{r'} = pk$: $r \leftarrow r'$; Else: $r \leftarrow \perp$	42 $k_{\text{otp}} \leftarrow \text{F}(k_{\text{KEM}}, \text{otp})$ G_3 - G_5
10 $(k_{\text{KEM}}, ck_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$	43 Else: $k_{\text{otp}} \leftarrow T_{\text{otp}}[r, ck_{\text{KEM}}]$ G_3 - G_5
11 If $ch \wedge r \in [n] \setminus XA$	44 $(h, \sigma) \leftarrow c' \oplus k_{\text{otp}}$ G_2 - G_5
12 $k_{\text{KEM}} \leftarrow_{\$} \mathcal{K}$	45 $pk \leftarrow \{pk' \in PK \mid H(pk') = h\}$; $(\_, pk_{\text{SIG}}) \leftarrow pk$ G_2 - G_5
13 $ck_{\text{KEM}} \leftarrow_{\$} \mathcal{CKEM}$	46 $b \leftarrow \text{SIG.vfy}(pk_{\text{SIG}}, (ck_{\text{KEM}}, h, ad), \sigma)$ G_4 - G_5
14 $T[r, ck_{\text{KEM}}] \leftarrow k_{\text{KEM}}$	47 If $b = \text{fal} \vee pk = \emptyset$: Return $\perp^2$ G_2 - G_5
15 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_{\text{SIG}}, (ck_{\text{KEM}}, h, ad))$	48 $k \leftarrow \text{F}(k_{\text{KEM}}, (\text{key}, pk, \mu(sk_r), c, ad))$ G_3 - G_5
16 $k_{\text{otp}} \leftarrow \text{F}(k_{\text{KEM}}, \text{otp})$	49 If $T[r, ck_{\text{KEM}}] = \perp$: G_3 - G_5
17 If $ch \wedge r \in [n] \setminus XA$	50 $k \leftarrow \text{F}(k_{\text{KEM}}, (\text{key}, pk, \mu(sk_r), c, ad))$ G_3 - G_5
18 $k_{\text{otp}} \leftarrow_{\$} \mathcal{K}$	51 Else: G_3 - G_5
19 $T_{\text{otp}}[r, ck_{\text{KEM}}] \leftarrow k_{\text{otp}}$	52 $k \leftarrow_{\$} \mathcal{K}$; $D \leftarrow \perp \cup \{(pk, r, c, ad, k)\}$ G_3 - G_5
20 $c' \leftarrow (h, \sigma) \oplus k_{\text{otp}}$	53 Return (pk, k) G_1 - G_4
21 $c' \leftarrow_{\$} \mathcal{K}$	G_5 Oracle $\text{Gen}(sk)$
22 $c_0 \leftarrow (ck_{\text{KEM}}, c')$	54 Require $\nexists u \in [n] : pk_u = \mu(sk)$
23 $k \leftarrow \text{F}(k_{\text{KEM}}, (\text{key}, \mu(sk_s), pk, c_0, ad))$	55 $n \leftarrow n + 1$
24 If $ch \wedge r \in [n] \setminus XA$:	G_3 - G_5 56 If $sk \in \mathcal{SK}$:
25 $k \leftarrow_{\$} \mathcal{K}$	G_3 - G_5 57 $(sk_n, pk_n) \leftarrow (sk, \mu(sk))$
26 $c_1 \leftarrow_{\$} \mathcal{CKEM} \times \mathcal{K}$	58 $XA \leftarrow \perp \cup \{n\}$
27 If ch : $CH \leftarrow \perp \cup \{(s, r, ad, c_0)\}$; Return (k, c_0)	59 Else:
28 Return (k, c_0)	60 $(sk_{\text{KEM}}, pk_{\text{KEM}}) \leftarrow_{\$} \text{KEM.gen}$
Oracle $\text{CorruptA}(u)$	61 $(sk_{\text{SIG}}, pk_{\text{SIG}}) \leftarrow_{\$} \text{SIG.gen}$
29 Require $u \in [n]$	62 $pk_n \leftarrow (pk_{\text{KEM}}, pk_{\text{SIG}})$
30 $XA \leftarrow \perp \cup \{u\}$	63 $sk_n \leftarrow (sk_{\text{KEM}}, sk_{\text{SIG}}, H(pk))$
31 Return sk_u	64 Return pk_n

Fig. 16. Games for the proof of Theorem 2.

Game G_1 is the original $\text{ANON}_{\text{AK}_{\text{a-a}}}^{b, \text{a}, \text{a}}$ game instantiated with our construction from Figure 7, where we have removed the parts that are only necessary for $s \in \text{CF} \cup \text{AU}$, including the index i . In particular,

$$\Pr[G_1^b \Rightarrow 1] = \Pr[\text{ANON}_{\text{AK}_{\text{a-a}}}^{b, \text{a}, \text{a}}(\mathcal{A}) \Rightarrow 1]$$

and

$$\text{Adv}_{\text{AK}_{\text{a-a}}}^{\text{ANON}_{\text{a-a}}}(\mathcal{A}) = |\Pr[G_1^0 \Rightarrow 1] - \Pr[G_1^1 \Rightarrow 1]|.$$

Game G_2 is identical to G_1 except that we now sample k_{KEM} uniformly at random for challenge queries to Enc whenever $r \in [n] \notin XA$. If $r = \perp$ or $r \in XA$, G_2 behaves as G_1 and runs the encapsulation honestly. We store k_{KEM} along with the index r in a table T to change the decryption oracle accordingly.

G_1 to G_2 : We construct a reduction $\mathcal{R}_{\text{KEM},1}^b$ against the IND security of the KEM scheme (cf. Figure 13). This reduction is the same as the corresponding reduction for the hop from G_1 to G_2 in the proof of Theorem 7. Hence, for each $b \in \{0,1\}$ it holds that

$$|\Pr[G_1^b \Rightarrow 1] - \Pr[G_2^b \Rightarrow 1]| \leq \text{Adv}_{\text{KEM}}^{\text{IND}}(\mathcal{R}_{\text{KEM},1}^b).$$

Game G_3 is identical to G_2 except for challenge queries to Enc and for queries to Dec that get as input a ciphertext containing c_{KEM} from a challenge query. More precisely, for those queries, G_3 samples $k_{\text{otp}} \leftarrow_{\$} \mathcal{K}$ uniformly at random, stores it associated to r, c_{KEM} , and uses it for forming c . It also samples the final key k uniformly at random. In Dec, whenever the ciphertext has the same pair (r, c_{KEM}) , G_3 uses the stored k_{otp} instead of evaluating the PRF. Instead of computing the final output with the PRF, it also samples a random key k for those queries. It records the whole query in a set D , as in the proof of Theorem 1.

G_2 to G_3 : We construct a reduction $\mathcal{R}_F$ against IND_F^b (cf. Figure 11) that uses $\mathcal{A}$ as a subroutine. $\mathcal{R}_F$ memorizes all answers returned by F.Eval and F.Chall and replays them on repeated inputs. The simulation is as follows:

- On query from $\mathcal{A}$ to Gen or CorruptA, reduction $\mathcal{R}_F$ simulates as in G_3^b .
- On query from $\mathcal{A}$ to Enc, if $\mathcal{A}$ asks for a (valid) challenge, $\mathcal{R}_F^b$ calls $\text{IND}_F.\text{Gen}(\perp)$ and stores the current counter value $T[r, c_{\text{KEM}}] \leftarrow j$ (instead of the actual key). It derives k_{otp} and k by querying $\text{IND}_F.\text{Eval}$ or computing it locally (for non-challenges). Note that the inputs are domain-separated using the labels.
- On query from $\mathcal{A}$ to Dec, reduction $\mathcal{R}_F^b$ works as in G_3^b . If $T[r, c_{\text{KEM}}]$ is defined, it uses the recorded k_{otp} , otherwise it queries $\text{IND}_F.\text{Eval}$. Then, it also runs $\text{IND}_F.\text{Chall}$ to derive the final session key if $T[r, c_{\text{KEM}}] \neq \perp$. Otherwise, it simulates locally.

Eventually, $\mathcal{A}$ outputs a bit b' and $\mathcal{R}_F^b$ outputs the same bit b' . If $\mathcal{R}_F^b$ is in game IND_F^0 , then it perfectly simulates G_2^b ; if $\mathcal{R}_F^b$ is in game IND_F^1 , then it perfectly simulates G_3^b . Thus, $\text{Adv}_F^{\text{IND}}(\mathcal{R}_F^b)$ bound the gap between G_2^b and G_3^b and for each $b \in \{0,1\}$ we have

$$|\Pr[G_2^b \Rightarrow 1] - \Pr[G_3^b \Rightarrow 1]| \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_F).$$

Game G_4 is identical to G_3 except that we switch c_{KEM} to a randomly sampled ciphertext for challenge queries. Since k_{KEM} is not needed anymore due to the previous game hop, we can bound the difference by the anonymity of the KEM.

G_3 to G_4 : We construct a reduction $\mathcal{R}_{\text{KEM},2}^b$ against $\text{ANON}_{\text{KEM}}^b$ (cf. Figure 1) that uses $\mathcal{A}$ as a subroutine. The simulation is as follows:

- On query from $\mathcal{A}$ to Gen or CorruptA, reduction $\mathcal{R}_{\text{KEM},2}^b$ simulates as $\mathcal{R}_{\text{KEM}}^b$ from the proof of Theorem 1.
- On query from $\mathcal{A}$ to Enc, if $\mathcal{A}$ asks for a challenge, $\mathcal{R}_{\text{KEM},2}^b$ calls $\text{ANON}_{\text{KEM}}.\text{Enc}(r)$ to receive c_{KEM} which is either real or random. It can then simulate the remaining operations as in G_4^b . For non-challenges, it just simulates locally.
- On query from $\mathcal{A}$ to Dec, reduction $\mathcal{R}_{\text{KEM},2}^b$ works as in G_4^b . In order to respond to queries with c_{KEM} not output by Enc, it can query the decryption oracle. For the other ones, k_{KEM} is not needed because k_{otp} and the final output k is already switched to random.

Eventually, $\mathcal{A}$ outputs a bit b' and $\mathcal{R}_{\text{KEM},2}^b$ outputs the same bit b' . If $\mathcal{R}_{\text{KEM},2}^b$ is in game $\text{ANON}_{\text{KEM}}^0$, then it perfectly simulates G_3^b ; if $\mathcal{R}_{\text{KEM},2}^b$ is in game $\text{ANON}_{\text{KEM}}^1$, then it perfectly simulates G_4^b . Thus, we have

$$|\Pr[G_3^b \Rightarrow 1] - \Pr[G_4^b \Rightarrow 1]| \leq \text{Adv}_{\text{KEM}}^{\text{ANON}}(\mathcal{R}_{\text{KEM},2}^b).$$

Game G_5 is identical to G_4 except that we now sample c' at random.

G_4 to G_5 : Since k_{otp} is already random and k from decapsulation queries with the same c_{KEM} does not leak anything beyond a random key, this change is only conceptual. Thus, the gap between G_4 and G_5 is zero and for bit b we have

$$\Pr[G_4^b \Rightarrow 1] = \Pr[G_5^b \Rightarrow 1].$$

Finally, note that in G_5 every challenge ciphertext c_0 is sampled uniformly from $\mathcal{C}$ and is distributed as c_1 . Hence the view of $\mathcal{A}$ in G_6 is independent of the challenge bit and we have

$$\Pr[G_5^0 \Rightarrow 1] = \Pr[G_5^1 \Rightarrow 1].$$

For each b , the first difference is bounded by the KEM reduction and the second by the PRF reduction. Absorbing the two b -dependent reductions into single reductions $\mathcal{R}_{\text{KEM}}$ and $\mathcal{R}_{\text{F}}$ gives the final bound of the theorem. $\square$

C.3 Proof of Theorem 3

Theorem 3. *The AKEM construction $\text{AK}_{\text{a-a}}$ from Figure 7 offers authenticity. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{a-a}}$ in game AUTH from Figure 5, there exists a reduction $\mathcal{R}_{\text{SIG}}$ in game SUF from Figure 12 against the signature scheme SIG, such that*

$$\text{Adv}_{\text{AK}_{\text{a-a}}}^{\text{AUTH}_{\text{a-a}}}(\mathcal{A}) \leq \text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{R}_{\text{SIG}}),$$

where the running time of $\mathcal{R}_{\text{SIG}}$ is about that of $\mathcal{A}$.

Proof. We prove the authenticity of AKEM $\text{AK}_{\text{a-a}}$ via a direct reduction to SUF_{SIG} (cf. Figure 12), showing that

$$\Pr[\text{AUTH}_{\text{AK}_{\text{a-a}}}^{\text{a,a}}(\mathcal{A}) \Rightarrow 1] \leq \text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{R}_{\text{SIG}}),$$

which yields the bound of the theorem statement. Note that the game outputs 1 if the adversary makes a query to Dec such that the ciphertext decapsulates to a key not equal to $\perp$, and the ciphertext was not previously output by the Enc oracle (for the same sender and receiver) and the sender was not corrupted. We will show that this translates exactly to a signature forgery in the sense of strong unforgeability.

Reduction $\mathcal{R}_{\text{SIG}}$ 00 $(n, m) \leftarrow 0^2$; $sk, pk, psk \leftarrow \perp^3$ 01 $C, XS, XA \leftarrow \emptyset^3$ 02 Invoke $\mathcal{A}$ 03 Stop with 0 Oracle Gen(sk) 04 Require $\exists u \in [n] : pk_u = \mu(sk)$ 05 $n \leftarrow n + 1$ 06 If $sk \in \mathcal{SK}$: 07 $(sk_n, pk_n) \leftarrow (sk, \mu(sk))$ 08 $XA \leftarrow \{n\}$ 09 Else: 10 $(sk_{\text{KEM}}, pk_{\text{KEM}}) \leftarrow \text{KEM.gen}$ 11 $pk_{\text{SIG}} \leftarrow \text{SUF.Gen}$ 12 $pk_n \leftarrow (pk_{\text{KEM}}, pk_{\text{SIG}})$ 13 $sk_n \leftarrow (sk_{\text{KEM}}, \perp, H(pk))$ 14 Return pk_n Oracle CorruptA(u) 15 Require $u \in [n]$ 16 $XA \leftarrow \{u\}$ 17 $sk_{\text{SIG}} \leftarrow \text{SUF.Corrupt}(u)$ 18 $sk_u \leftarrow (\_, sk_{\text{SIG}}, \_)$ 19 Return sk_u	Oracle Enc(s, pk, ad) 20 Require $(s, pk) \in [n] \times \mathcal{PK}$ 21 $(\_, \_, h) \leftarrow sk_s$ 22 $(pk_{\text{KEM}}, \_) \leftarrow pk$ 23 $(k_{\text{KEM}}, ck_{\text{KEM}}) \leftarrow \text{KEM.enc}(pk_{\text{KEM}})$ 24 $\sigma \leftarrow \text{SUF.Sig}(s, (ck_{\text{KEM}}, h, ad))$ 25 $k_{\text{otp}} \leftarrow F(k_{\text{KEM}}, \text{otp})$ 26 $c \leftarrow (ck_{\text{KEM}}, (h, \sigma) \oplus k_{\text{otp}})$ 27 $k \leftarrow F(k_{\text{KEM}}, (\text{key}, \mu(sk_s), pk, c, ad))$ 28 If $\exists r \in [n] : pk_r = pk$: $CH \leftarrow \{(s, r, ad, c)\}$ 29 Return (k, c) Oracle Dec(r, PK, ad, c) 30 Require $(r, PK) \in [n] \times \mathcal{P}(\mathcal{PK})$ 31 $(sk_{\text{KEM}}, \_, \_) \leftarrow sk_r$; $(ck_{\text{KEM}}, c') \leftarrow c$ 32 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, ck_{\text{KEM}})$ 33 $k_{\text{otp}} \leftarrow F(k_{\text{KEM}}, \text{otp})$ 34 $(h, \sigma) \leftarrow c' \oplus k_{\text{otp}}$ 35 $pk \leftarrow \{pk' \in PK \mid H(pk') = h\}$ 36 $(\_, pk_{\text{SIG}}) \leftarrow pk$ 37 $b \leftarrow \text{SIG.vfy}(pk_{\text{SIG}}, (ck_{\text{KEM}}, h, ad), \sigma)$ 38 If $b = \text{fal} \vee pk = \emptyset$: Return $\perp^3$ 39 $k \leftarrow F(k_{\text{KEM}}, (\text{key}, pk, \mu(sk_r), c, ad))$ 40 If $\exists s' \in [n] : pk_{s'} = pk$: $s \leftarrow s'$; Else: $s \leftarrow \perp$ 41 If $k \neq \perp \wedge (s, r, ad, c) \notin C \wedge \perp \neq s \notin XA$ 42 $\text{SUF.Vfy}(s, (ck_{\text{KEM}}, h, ad), \sigma)$ 43 Return (pk, k)
---	--

Fig. 17. Reduction for the proof of Theorem 3. For simplicity, we assume that $\mathcal{R}_{\text{SIG}}$'s challenger uses the same indices as in the simulation.

Reduction $\mathcal{R}_{\text{SIG}}$: In Figure 17, we construct a reduction $\mathcal{R}_{\text{SIG}}$ against SUF_{SIG} (cf. Figure 12) that runs $\mathcal{A}$ and simulates $\text{AUTH}_{\text{AK}_{\text{a-a}}}^{\text{a,a}}$. The simulation is as follows:

- On query from $\mathcal{A}$ to Gen, $\mathcal{R}_{\text{SIG}}$ simulates as in the original game if $\mathcal{A}$ asks to generate a user with secret key sk . If $sk = \perp$, it generates $(sk_{\text{KEM}}, pk_{\text{KEM}})$ locally, queries $pk_{\text{SIG}} \leftarrow \text{SUF}_{\text{SIG}}.\text{Gen}$ and sets $pk_n = (pk_{\text{KEM}}, pk_{\text{SIG}})$. Similar to earlier proofs, we simplify the mapping of indices. Here, we assume that the SUF game keeps the same indices. Technically, the reduction would have to define a mapping, which we omit from Figure 17.
- On query from $\mathcal{A}$ to Enc, $\mathcal{R}_{\text{SIG}}$ simulates $\text{AK}_{\text{a-a}}.\text{enc}$ as in Figure 7, except that it computes the signature by querying the signing oracle.
- On query from $\mathcal{A}$ to Dec, reduction $\mathcal{R}_{\text{SIG}}$ simulates $\text{AK}_{\text{a-a}}.\text{dec}$ exactly as in the original game. If it gets to the check of the winning condition, this means that the signature has verified. If further the other conditions hold, then $\mathcal{R}_{\text{SIG}}$ queries the verification oracle from the SUF game. We explain below why this call will set the win flag to true.
- On query from $\mathcal{A}$ to CorruptA, $\mathcal{R}_{\text{SIG}}$ forwards to $\text{SUF}_{\text{SIG}}.\text{Expose}$ to obtain sk_{SIG} for this user, retrieves the locally stored sk_{KEM} and h , and returns the full secret key.

It remains to show that whenever $\mathcal{A}$ wins, then $\mathcal{R}_{\text{SIG}}$ wins as well. When the reduction runs the checks in line 41, then we already know that the signature encrypted inside the ciphertext verified for some sender public key, which is determined in the line before and cannot be $\perp$ (otherwise, the adversary could have trivially created a forgery). We also know that the pair (c_{KEM}, c') was not previously output by the Enc oracle. This means that at least one of the components needs to be new. We get the following case distinction:

- If c_{KEM} was never output by Enc, then, since c_{KEM} is part of the message signed, σ is a valid forgery for (c_{KEM}, h, ad) and sender s .
- If c_{KEM} has been output by Enc, then it decapsulates to the same k_{KEM} and therefore results in the same k_{otp} . This means that either h or σ or both must be different from the ones used in the encapsulation (since c' must be different). If at least σ is different, then this constitutes a valid forgery because we are reducing to strong unforgeability. If at least h is different, then it must be a valid forgery for another sender s' .

This concludes the description of $\mathcal{R}_{\text{SIG}}$ and the proof. $\square$

D Security of Symmetric Signcryption with Redundant Authenticity

D.1 Proof of Theorem 4

Theorem 4. *The AKEM construction $\text{AK}_{\text{s-sa}}$ from Figure 7 offers key indistinguishability. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{s-sa}}$ in game IND from Figure 3, there exists a reduction $\mathcal{R}_{\text{F}}$ in game IND from Figure 11 against the PRF F , such that*

$$\text{Adv}_{\text{AK}_{\text{s-sa}}}^{\text{IND}}(\mathcal{A}) \leq 2 \cdot \left(\frac{(q_e + d)^2}{|\mathcal{N}|} + \text{Adv}_{\text{F}}^{\text{IND}}(\mathcal{R}_{\text{F}}) \right),$$

where $q_e + d$ are the number of queries to the encapsulation and decapsulation oracle, $\mathcal{N}$ is the nonce space, and the running time of $\mathcal{R}_{\text{F}}$ is about that of $\mathcal{A}$.

Proof. We prove this with a sequence of games. The first hop catches the bad event in which the uniformly random sampled nonce (line 11) has a collision. In the next game hop, we replace the PRF in line 12 by sampling the key uniformly at random. We define the games as G_i for $i \in [3]$.

Game G_1 is the original $\text{IND}_{\text{AK}_{\text{s-sa}}}^{b, \text{s}, \{\text{s}, \text{a}\}}(\mathcal{A})$ game instantiated with our construction. In particular,

$$\Pr[\text{G}_1^b \Rightarrow 1] = \Pr[\text{IND}_{\text{AK}_{\text{s-sa}}}^{b, \text{s}, \{\text{s}, \text{a}\}}(\mathcal{A}) \Rightarrow 1]$$

and

$$\text{Adv}_{\text{AK}_{\text{s-sa}}}^{\text{IND}}(\mathcal{A}) = |\Pr[\text{G}_1^0 \Rightarrow 1] - \Pr[\text{G}_1^1 \Rightarrow 1]|.$$

Game G_2 is identical to G_1 except that it aborts if a nonce repeats among encapsulation and decapsulation queries.

Game G_1^b-G_3^b 00 $(n, m) \leftarrow 0^2; sk, pk, psk \leftarrow \perp^3$ 01 $CH, XS, XA \leftarrow \emptyset^3$ 02 $b' \leftarrow_{\$} \mathcal{A}$ 03 $N \leftarrow \emptyset$ 04 For all $(s, i, r, ad, c) \in CH$: 05 Require $s \in CF \wedge i \notin XS$ 06 $\forall a \in CF \wedge \perp \neq r \notin XA$ 07 Stop with b' Oracle Enc (s, i, pk, ad, ch) 07 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$ 08 Require $a \in AU \iff s \in [n]$ 09 Require $s \in CF \cup AU \iff i \in [m]$ 10 Require $a \in CF \iff pk \in \mathcal{PK}$ 11 $(sk_{SIG}, h) \leftarrow sk_s$ 12 $n \leftarrow_{\$} \mathcal{N}$ 13 If $n \in N$: Abort 14 $N \leftarrow \{n\}$ 15 $k_{tag} \leftarrow F(pk_i, \text{tag})$ 16 $c_T \leftarrow_{\$} T.\text{tag}(k_{tag})$ 17 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_{SIG}, (n, c_T, h, ad))$ 18 $k_{otp} \leftarrow F(pk_i, (\text{otp}, n))$ 19 $k_r \leftarrow F(pk_i, \text{mac})$ 20 $\tau \leftarrow F(k_r, (n, c_T, h, \sigma, ad))$ 21 $c \leftarrow (n, c_T, (h, \sigma, \tau) \oplus k_{otp})$ 22 $k_0 \leftarrow F(pk_i, (\text{key}, c, ad))$ 23 If $ch \wedge r \in [n] \setminus XA: k_0 \leftarrow_{\$} \mathcal{K}$ 24 $k_1 \leftarrow_{\$} \mathcal{K}$ 25 If $\exists r' \in [n] : pk_{r'} = pk: r \leftarrow r'$ 26 Else: $r \leftarrow \perp$ 27 If ch : 28 $CH \leftarrow \{(s, i, r, ad, c)\}$ 29 Return (k_b, c) 30 Return (k_0, c) Oracle CorruptS (i) 31 Require $i \in [m]$ 32 $XS \leftarrow \{i\}$ 33 Return psk_i Oracle CorruptA (u) 34 Require $u \in [n]$ 35 $XA \leftarrow \{u\}$ 36 Return sk_u	Oracle Dec (r, PK, I, ad, c) 37 Require $(r, PK, I) \in ([n] \times \mathcal{P}(\mathcal{PK}) \times \mathcal{P}([m])) \cup \{\perp^3\}$ 38 Require $a \in AU \iff PK \subseteq \mathcal{PK}$ 39 Require $s \in CF \cup AU \iff I \subseteq [m]$ 40 Require $a \in CF \iff r \in [n]$ 41 If $\exists (s', i', r, ad, c) \in CH : i' \in I \cup \{\perp\}$ 42 $\wedge (s' = \perp \vee pk_{s'} \in PK)$: Return $\perp$ 43 $K' \leftarrow \emptyset; P'[\cdot] \leftarrow \perp$ 44 For all $psk \in (psk_i)_{i \in I}$: 45 $k_{tag} \leftarrow F(pk, \text{tag})$ 46 $K' \leftarrow K' \cup \{k_{tag}\}$ 47 $P'[k_{tag}] \leftarrow psk$ 48 $st' \leftarrow_{\$} T.\text{prec}(K')$ 49 $(n, c_T, c') \leftarrow c$ 50 $N \leftarrow \{n\}$ 51 $k_T \leftarrow T.\text{det}(st', c_T); psk \leftarrow P'[k_T]$ 52 Require $psk \neq \perp$ 53 $k_{otp} \leftarrow F(pk, (\text{otp}, n))$ 54 $(h, \sigma, \tau) \leftarrow c' \oplus k_{otp}$ 55 $pk \leftarrow \{pk \in PK \mid H(pk) = h\}$ 56 $b \leftarrow \text{SIG.vfy}(pk, (n, c_T, h, ad), \sigma)$ 57 $k_r \leftarrow F(pk, \text{mac})$ 58 $\tau' \leftarrow F(k_r, (n, c_T, h, \sigma, ad))$ 59 If $b = \text{fal} \vee \tau \neq \tau' \vee pk = \emptyset$: Return $\perp^3$ 60 $k \leftarrow F(pk, (\text{key}, c, ad))$ 61 If $\exists i' \in [m] : psk_{i'} = psk: i \leftarrow i'$ 62 Else: $i \leftarrow psk$ 63 Return (pk, i, k) Oracle Gen (sk) 63 Require $\nexists u \in [n] : pk_u = \mu(sk)$ 64 $n \leftarrow n + 1$ 65 If $sk \in SK$: 66 $(sk_n, pk_n) \leftarrow (sk, \mu(sk)); XA \leftarrow \{n\}$ 67 Else: $(sk_n, pk_n) \leftarrow_{\$} \text{AK.gen}$ 68 Return pk_n Oracle Psk (psk) 69 Require $\nexists i \in XS : psk_i = psk$ 70 $m \leftarrow m + 1$ 71 If $psk \in \mathcal{PSK}$: 72 $psk_m \leftarrow psk; XS \leftarrow \{m\}$ 73 Else: $psk_m \leftarrow_{\$} \mathcal{PSK}$ 74 Return
---	--

Fig. 18. Games for the proof of Theorem 4 for IND security of AKEM construction from Figure 8 with $CF = \mathbf{s}; AU = \{\mathbf{s}, \mathbf{a}\}$.

G_1 to G_2 : A nonce is sampled uniformly at random in each encapsulation query. Moreover, the adversary can guess a new nonce with each decapsulation query. The probability of nonce collisions is thus bounded by the birthday bound in q_{e+d} number of encapsulation and decapsulation queries and nonce space $\mathcal{N}$. Since G_1 and G_2 are identical until nonce collision event occurs, the gap between the games $b \in \{0, 1\}$ is

$$|\Pr[G_1^b \Rightarrow 1] - \Pr[G_2^b \Rightarrow 1]| \leq \frac{(q_{e+d})^2}{|\mathcal{N}|}.$$

Game G_3 is identical to G_2 except that we replace the PRF output k in line 12 by the output of a random function for each challenge query to IND.Enc. Through this change, the output key k is a random value and the adversary has no advantage in distinguishing the real from the random world. We do not need to keep track of the random replacement values for the PRF key k_0 : First, the input into encapsulation queries is always new (since we have a fresh nonce each time via the previous game). Second, in decapsulation queries, the adversary can never request the decapsulation of a challenge ciphertext and replaying parts of challenge ciphertexts also does not help as this results in a non-challenge input to the PRF.

We now show that the output of the PRF is indistinguishable from the output of a random function through the following reduction.

G_2 to G_3 : We construct a reduction $\mathcal{R}_F^b$ against IND_F^b (cf. Figure 11) that runs $\mathcal{A}$ as a subroutine and simulates $\text{IND}_{\text{AK}_{\text{s-sa}}}^{b, \text{CF}, \text{AU}}$ for $\mathcal{A}$. Reduction $\mathcal{R}_F^b$ maps PRF challengers by pre-shared key indices. On each call to the pre-shared key generation oracle $\text{Psk}(psk)$ creating a fresh i , the reduction calls the PRF challenger's oracle $\text{Gen}(psk)$ if $psk \in \mathcal{PSK}$ and otherwise $\text{Gen}()$, and assigns the resulting index to i . As explained above, the same challenge query to a PRF is never repeated. The simulation is as follows:

- On query from $\mathcal{A}$ to $\text{Psk}(psk)$, reduction $\mathcal{R}_F^b$ creates a new index i . If $psk = \perp$, this index is mapped to a new PRF challenger in game IND_F^b . Otherwise, psk is set by the adversary, the reduction stores it and performs the respective computations internally.
- On query from $\mathcal{A}$ to Enc , reduction $\mathcal{R}_F^b$ computes c exactly as in the construction, but obtains all values of the form $F(psk_i, x)$ by querying $F.\text{Eval}(\{i\}, x)$ except for the challenge encapsulations: In this case, i.e., $ch = \text{tru}$, the reduction forwards the generation of k to the PRF challenger as $k \leftarrow F.\text{Chall}(i, (\text{key}, c, ad))$. Finally, the reduction returns (k, c) (and stores the tuple in CH as in Figure 3).
- On query $\text{IND}.\text{CorruptS}(i)$, if i does not correspond to an adversarially set pre-shared key, the reduction forwards corrupts the corresponding PRF challenger $F.\text{Corrupt}(i)$, stores, and returns the key.
- On any query from $\mathcal{A}$ to an oracle other than $\text{IND}.\text{Enc}$, $\mathcal{R}_F^b$ simulates exactly as in Figure 18.

Eventually, $\mathcal{A}$ outputs a bit b' and $\mathcal{R}_F^b$ outputs the same bit b' for its IND_F^b experiment. Thus $\text{Adv}_F^{\text{IND}}(\mathcal{R}_F^b)$ bounds the gap between G_2 and G_3 and for each $b \in \{0, 1\}$ it holds that

$$|\Pr[G_2^b \Rightarrow 1] - \Pr[G_3^b \Rightarrow 1]| \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_F^b).$$

In G_3 , for every challenge query to Enc the game samples $k_0 \leftarrow_{\$} \mathcal{K}$ and $k_1 \leftarrow_{\$} \mathcal{K}$ and returns k_b . Hence, the challenge key returned to $\mathcal{A}$ is uniformly distributed over $\mathcal{K}$ independent of b . All other oracle answers are identically distributed in G_3^0 and G_3^1 . As a result, the view of $\mathcal{A}$ is identically distributed in both experiments, and we have

$$\Pr[G_3^0 \Rightarrow 1] = \Pr[G_3^1 \Rightarrow 1].$$

Absorbing the two b -dependent reductions into a single reduction costs a factor 2. Combining the bounds from the game hops, we obtain the bound from the theorem. $\square$

D.2 Proof of Theorem 5

Theorem 5. *The AKEM construction $\text{AK}_{\text{s-sa}}$ from Figure 7 offers anonymity. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{s-sa}}$ in game ANON from Figure 4 in the ROM, there exists a reduction $\mathcal{R}_T$ against IND security of tagging scheme T in game IND from Figure 6, such that*

$$\text{Adv}_{\text{AK}_{\text{s-sa}}}^{\text{ANON}}(\mathcal{A}) \leq 2 \cdot \left(\frac{q_{e+d}^2}{|\mathcal{N}|} + \frac{q_{psk} q_{\text{RO}}}{|\mathcal{PSK}|} + \frac{q_{psk} q_{\text{RO}}}{|\mathcal{K}_\tau|} + \text{Adv}_T^{\text{IND}}(\mathcal{R}_T) \right),$$

with variables as in Theorem 4, q_{RO} the number of adversarial RO queries, $\mathcal{PSK}$ the pre-shared key space, $\mathcal{K}_\tau$ the space of authentication tag keys, and the running time of $\mathcal{R}_T$ is about that of $\mathcal{A}$.

Proof. We prove that $\text{AK}_{\text{s-sa}}$ satisfies the ANON security with a sequence of games modeling the PRF as a random oracle (RO) which uses lazy sampling. We require the random oracle model because of commitment problems when replacing a pre-shared key with a random value: One step towards showing that the returned ciphertext is anonymous is to show that the tagging function's ciphertext is indistinguishable from random. To show this, we need to replace k_{tag} with a random value. Now, consider the case where the adversary generates a new psk and queries an encapsulation. At this point, we do not know whether the adversary corrupts the psk later or asks for a challenge with the psk (disallowing a later corruption). Yet, the input to the tagging function, k_{tag} needs to be consistent across challenge and non-challenge encapsulation queries, so the reduction needs to decide immediately whether to use a random value or not. Using a random oracle, the simulation can internally use a consistent k_{tag} , which allows a consistent simulation across encapsulations and challenges. Note that we could avoid the ROM by guessing for each key whether it is corrupted

Game $G_1^b - G_6^b$ 00 $(n, m) \leftarrow 0^2; sk, pk, psk \leftarrow \perp^3$ 01 $CH, XS, XA \leftarrow \emptyset^3$ 02 $N \leftarrow \emptyset$ 03 $H[\cdot, \cdot, \cdot] \leftarrow \perp$ 04 $b' \leftarrow_{\$} \mathcal{A}$ 05 For all $(s, i, r, ad, c) \in CH$: 06 Require $s \in CF \wedge i \notin XS$ 07 $\vee a \in CF \wedge \perp \neq r \notin XA$ 07 Stop with b' Oracle Gen(sk) 08 Require $\nexists u \in [n] : pk_u = \mu(sk)$ 09 $n \leftarrow n + 1$ 10 If $sk \in SK$ 11 $(sk_n, pk_n) \leftarrow (sk, \mu(sk))$ 12 $XA \leftarrow \{n\}$ 13 Else: $(sk_n, pk_n) \leftarrow_{\$} AK.gen$ 14 Return pk_n Oracle Psk(psk) 15 Require $\nexists i \in XS : psk_i = psk$ 16 $m \leftarrow m + 1$ 17 If $psk \in PSK$: 18 $psk_m \leftarrow psk$ 19 $XS \leftarrow \{m\}$ 20 Else: $psk_m \leftarrow_{\$} PSK$ 21 $T_m \leftarrow \emptyset$ 22 Return Oracle CorruptS(i) 23 Require $i \in [m]$ 24 $XS \leftarrow \{i\}$ 25 Return psk_i Oracle CorruptA(u) 26 Require $u \in [n]$ 27 $XA \leftarrow \{u\}$ 28 Return sk_u Oracle RO(k, l, a) 29 Require $(k, l, a) \in PSK \times \mathcal{L} \times \{0, 1\}^*$ 30 $H[k, l, a] \leftarrow_{(s)} H[k, l, a] ? \mathcal{K}$ 31 Return $H[k, l, a]$ Oracle RO^A(k, l, a) 32 Require $(k, l, a) \in PSK \times \mathcal{L} \times \{0, 1\}^*$ 33 If $\exists i \in [m] \setminus XS : k = psk_i$: Abort 34 If $\exists i \in [m] \setminus XS : k = H[psk_i, mac]$: Abort 35 $H[k, l, a] \leftarrow_{(s)} H[k, l, a] ? \mathcal{K}$ 36 Return $H[k, l, a]$	 Oracle Enc(s, i, pk, ad, ch) 37 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$ 38 Require $a \in AU \iff s \in [n]$ 39 Require $s \in CF \cup AU \iff i \in [m]$ 40 $(sk_{SIG}, h) \leftarrow sk_s$ 41 $n \leftarrow_{\$} \mathcal{N}$ 42 If $n \in N$: Abort 43 $N \leftarrow \{n\}$ 44 $k_{tag} \leftarrow RO(pk_{SIG}, tag)$ 45 $c_T \leftarrow_{\$} T.tag(k_{tag})$ 46 $c_{T,0} \leftarrow c_T$ 47 If ch : $c_T \leftarrow_{\$} \mathcal{C}_T$ 48 $\sigma \leftarrow_{\$} SIG.sig(sk_{SIG}, (n, c_T, h, ad))$ 49 $k_{otp} \leftarrow RO(pk_{SIG}, (otp, n))$ 50 $k_\tau \leftarrow RO(pk_{SIG}, mac)$ 51 $\tau \leftarrow RO(k_\tau, (n, c_T, h, \sigma, ad))$ 52 $c_0 \leftarrow (n, c_T, (h, \sigma, \tau) \oplus k_{otp})$ 53 If ch : $c_0 \leftarrow_{\$} \mathcal{C}$ 54 $k \leftarrow RO(pk_{SIG}, (key, c_0, ad))$ 55 $c_1 \leftarrow_{\$} \mathcal{C}$ 56 If $\exists r' \in [n] : pk_{r'} = pk$: $r \leftarrow r'$; Else: $r \leftarrow \perp$ 57 If ch : 58 $(\_, c_{T,b}, \_) \leftarrow c_b$ 59 $T_i \leftarrow_{\$} \{c_{T,0}, c_{T,b}\}$ 60 $CH \leftarrow \{(s, i, r, ad, c_b)\}$ 61 Return (k, c_b) 62 Return (k, c_0) Oracle Dec(r, PK, I, ad, c) 63 Require $(r, PK, I) \in ([n] \times \mathcal{PK} \times \mathcal{P}([m])) \cup \{\perp^3\}$ 64 Require $a \in AU \iff PK \subseteq \mathcal{PK}$ 65 Require $s \in CF \cup AU \iff I \subseteq [m]$ 66 If $\exists (s', i', r, ad, c) \in CH : i' \in I \cup \{\perp\}$ 67 $\wedge (s' = \perp \vee pk_{s'} \in PK)$: Return $\perp$ 68 $K' \leftarrow \emptyset; P'[\cdot] \leftarrow \perp$ 69 For all $psk \in (psk_i)_{i \in I}$: 70 $k_{tag} \leftarrow RO(pk, tag)$ 71 $K' \leftarrow \{k_{tag}\}; P'[k_{tag}] \leftarrow psk$ 72 $st' \leftarrow_{\$} T.prec(K')$ 73 $(n, c_T, c') \leftarrow c$ 74 If $\exists i, c_{T,0} : (c_{T,0}, c_T) \in T$: 75 $c_T \leftarrow c_{T,0}$ 76 $k_T \leftarrow T.det(st', c_T)$ 77 $psk \leftarrow P'[k_T]$; Require $psk \neq \perp$ 78 $k_{otp} \leftarrow RO(pk, (otp, n)); (h, \sigma, \tau) \leftarrow c' \oplus k_{otp}$ 79 $pk \leftarrow \{pk \in PK \mid H(pk) = h\}$ 80 $k_\tau \leftarrow RO(pk, mac)$ 81 $\tau' \leftarrow RO(k_\tau, (n, c_T, h, \sigma, ad))$ 82 $b \leftarrow SIG.vfy(pk, (n, c_T, h, ad), \sigma)$ 83 If $b = \text{fal} \vee \tau' \neq \tau \vee pk = \emptyset$: Return $\perp^3$ 84 $k \leftarrow RO(pk, (key, c, ad))$ 85 If $\exists i' \in [m] : psk_{i'} = psk$: $i \leftarrow i'$; Else: $i \leftarrow psk$ 86 Return (pk, i, k)
--	---

Fig. 19. Games for the proof of Theorem 5 for ANON security of AKEM construction from Figure 8 with $CF = \mathbf{s}; AU = \{\mathbf{s}, \mathbf{a}\}$.

or not, which, however, would result in an exponential blow-up in the number of pre-shared keys (one binary decision per key).

Again, the sequence of games starts with catching collisions for the nonce. Then, we prevent the adversary from querying the random oracle on any uncorrupted honestly generated pre-shared key (i.e., we abort on such queries). As a result, all RO evaluations on secret PSKs remain information-theoretically hidden until corruption. Then, for challenge queries, we replace the tag from the tagging mechanism with a uniform random tag utilizing a reduction towards its IND security. Finally, we make a conceptual game hop in which we then sample the whole output ciphertext c_0 from random. We define the games as G_i for $i \in [5]$. Furthermore, for better readability, we provide the adversary with its own interface to the random oracle RO^A , which shares an internal state with the game's RO.

Game G_1 is the original $\text{ANON}_{\text{AK}_{s\text{-sa}}}^{b,s,\{s,a\}}(\mathcal{A})$ game instantiated with our construction. In particular,

$$\Pr[G_1^b \Rightarrow 1] = \Pr[\text{ANON}_{\text{AK}_{s\text{-sa}}}^{b,s,\{s,a\}}(\mathcal{A}) \Rightarrow 1]$$

and

$$\text{Adv}_{\text{AK}_{s\text{-sa}}}^{\text{ANON}_{s\text{-sa}}}(\mathcal{A}) = |\Pr[G_1^0 \Rightarrow 1] - \Pr[G_1^1 \Rightarrow 1]|.$$

Game G_2 is identical to G_1 except that it aborts if a nonce repeats among encapsulation and decapsulation queries.

G_1 to G_2 : A nonce is sampled uniformly at random in each encapsulation query. Moreover, the adversary can guess with each decapsulation query a new nonce. The probability of nonce collisions is thus bounded by the birthday bound in q_{e+d} number of encapsulation and decapsulation queries and nonce space $\mathcal{N}$. Since G_1 and G_2 are identical until nonce collision event E_n occurs, $\Pr[E_n]$ bounds the gap between the games and for each $b \in \{0, 1\}$ we have

$$|\Pr[G_1^b \Rightarrow 1] - \Pr[G_2^b \Rightarrow 1]| \leq \frac{(q_{e+d})^2}{|\mathcal{N}|}.$$

Game G_3 is identical to G_2 except that G_3 aborts if the adversary queries its random oracle $\text{RO}^{\mathcal{A}}$ for an uncorrupted pre-shared key. Through this hop, we ensure that all outputs of the RO look random to the adversary as long as the corresponding pre-shared key is not exposed or queried. In particular, the resulting key $k \leftarrow \text{RO}(psk_i, (\text{key}, c_0, ad))$ is thereby random and independent of the output ciphertext.

G_2 to G_3 : First, note that $\mathcal{A}$ may query RO on any pre-shared key that is adversarially chosen or already corrupted, and these queries are answered exactly as in G_2 . For an uncorrupted honestly sampled key psk_i , however, psk_i is uniform in $\mathcal{PSK}$ and information-theoretically hidden from $\mathcal{A}$ until corruption, so the only way to trigger the abort is to guess its value. A union bound over at most q_{psk} honestly generated keys and q_{RO} adversarial RO queries bounds the probability $\Pr[E_{psk}]$ of this abort event E_{psk} . Note that this bound is conservative, since all RO queries whose first argument matches an honest uncorrupted psk_i cause abort, while only a subset of these RO queries are actually used by the game in challenge queries. Since G_2 and G_3 are identical until the adversary, using q_{RO} queries to the RO, guesses one of the q_{psk} uncorrupted pre-shared keys, i.e., E_{psk} occurs, $\Pr[E_{psk}]$ bounds the gap between the games and for each $b \in \{0, 1\}$ we have

$$|\Pr[G_2^b \Rightarrow 1] - \Pr[G_3^b \Rightarrow 1]| \leq \frac{q_{psk}q_{\text{RO}}}{|\mathcal{PSK}|}.$$

Game G_4 is identical to G_3 except that we sample a random tagging ciphertext in challenge encapsulations.

G_3 to G_4 : We build a reduction $\mathcal{R}_T^b$ against the IND_T security of the tagging scheme (Figure 6). If the adversary detects this change between the games, $\mathcal{R}_T^b$, we use this to break the IND_T security of the tagging scheme (Figure 6). First, note that the pre-shared key used to derive the tagging key k_{tag} in a challenge is unknown to the adversary via the previous game hop. That means, for challenge queries, we can outsource the tag computation to the IND_T . The following reduction maps the pre-shared key indices to tagging challenger indices; all computations involving a pre-shared key set by the adversary are performed by the reduction internally. For readability, we omit this bookkeeping and implicitly associate every query to a tagging oracle with the corresponding pre-shared key index (or set of indices).

- On query from $\mathcal{A}$ to Enc (where here $pk = \perp$), reduction $\mathcal{R}_T^b$ simulates $\text{AK}_{s\text{-sa}}.\text{enc}$ as in the original construction except that the tag component is obtained by querying T.Tag if $ch = \text{fal}$ and otherwise T.Chall .
- On query from $\mathcal{A}$ to Dec, reduction $\mathcal{R}_T^b$ simulates $\text{AK}_{s\text{-sa}}.\text{dec}$ but forwards tag detection to T.Det (which internally performs the preprocessing via T.prec). It then continues with the remaining checks and outputs as in the experiment.
- On query from $\mathcal{A}$ to $\text{CorruptS}(i)$, if i corresponds to an uncorrupted pre-shared key, the reduction corrupts the tagging challenger to obtain k_T . It then embeds this value in the RO as $H[psk_i, \text{tag}] \leftarrow k_T$.

– All other queries are simulated internally as described by the construction (see Figure 19). Eventually $\mathcal{A}$ outputs a bit b' , and $\mathcal{R}_T^b$ outputs the same bit b' as its guess for the challenger bit in IND_T . If the tagging challenger uses $b = 0$, the simulation is as in $\mathcal{G}_3^b$; otherwise as in $\mathcal{G}_4^b$. Therefore, for each $b \in \{0, 1\}$ we have

$$|\Pr[\mathcal{G}_3^b \Rightarrow 1] - \Pr[\mathcal{G}_4^b \Rightarrow 1]| \leq \text{Adv}_T^{\text{IND}}(\mathcal{R}_T^b).$$

Game $\mathcal{G}_5$ is identical to $\mathcal{G}_4$ except that $\mathcal{G}_5$ aborts if the adversary queries its random oracle RO^A with a correct k_τ (prior output by the RO) in an uncorrupted case (honest psk_i). To understand the necessity of this hop, consider this toy example: At this point, we have no guarantees for the signature scheme. Consequently, we assume the adversary can forge new valid signatures. Further, if the adversary knows k_τ , it is possible to adapt the ciphertext in such a way that it holds $\tau' = \tau$ (while having a new ciphertext with a valid signature). As a result, decryption would not fail. However, this is only possible in the case where Enc outputs the real ciphertext c_0 . In the other case, where Enc outputs a uniformly sampled ciphertext c_1 , the same adaptation strategy would fail with overwhelming probability, since the c_1 (in contrast to c_0) has no exploitable structure.

$\mathcal{G}_4$ to $\mathcal{G}_5$: First, note that $\mathcal{A}$ may query the RO on any value that does not equal an honest uncorrupted key k_τ , and these queries are answered exactly as in $\mathcal{G}_4$. For an uncorrupted honestly sampled case, however, the corresponding value k_τ was previously output by a random oracle and is therefore uniform in $\mathcal{K}$. As a consequence, the only way to trigger the abort is to guess its value. A union bound over at most q_{psk} such honest values k_τ and q_{RO} adversarial queries to the RO bounds the probability $\Pr[E_\tau]$ of this abort event E_τ . Since $\mathcal{G}_4$ and $\mathcal{G}_5$ are identical until the adversary, using q_{RO} queries to the RO, guesses one of the at most q_{psk} honest hidden values k_τ , i.e., E_τ occurs, $\Pr[E_\tau]$ bounds the gap between the games and for each $b \in \{0, 1\}$ we have

$$|\Pr[\mathcal{G}_4^b \Rightarrow 1] - \Pr[\mathcal{G}_5^b \Rightarrow 1]| \leq \frac{q_{psk}q_{\text{RO}}}{|\mathcal{K}|}.$$

Game $\mathcal{G}_6$ is identical to $\mathcal{G}_5$ except that in challenge encapsulations, the computation of c_0 is replaced by an independent uniform sample from its ciphertext space.

$\mathcal{G}_5$ to $\mathcal{G}_6$: Consider any challenge encapsulation in $\mathcal{G}_5$ with nonce n and pre-shared key index i . By $\mathcal{G}_2$, n is fresh among all previous encapsulations, hence the RO input $(psk_i, (\text{otp}, n))$ is fresh for RO. Moreover, by $\mathcal{G}_3$ the adversary cannot query RO^A on the uncorrupted challenge key, and thus does not obtain $k_{\text{otp}} = \text{RO}(psk_i, (\text{otp}, n))$. Therefore k_{otp} is uniform and information-theoretically hidden from $\mathcal{A}$, and the one-time pad makes $c' = (h, \sigma, \tau) \oplus k_{\text{otp}}$ uniform over its domain. Replacing c_0 by an independent uniformly randomly sampled value does not change the distribution of the adversary's view. Thus, for all $b \in \{0, 1\}$ we have

$$\Pr[\mathcal{G}_5^b \Rightarrow 1] = \Pr[\mathcal{G}_6^b \Rightarrow 1].$$

In $\mathcal{G}_6$, in every challenge query, c_0 as well as c_1 are sampled uniformly at random from the ciphertext space. By $\mathcal{G}_3$, $\mathcal{A}$ cannot query RO^A on the uncorrupted challenge pre-shared key, and thus never obtains the RO value at the challenge point. Therefore, conditioned on the absence of abort events, the returned challenge key is uniformly distributed over $\mathcal{K}$ and independent of b , and the returned challenge ciphertext is also independent of b . All other oracle answers are identically distributed in $\mathcal{G}_5^0$ and $\mathcal{G}_5^1$. As a result, the view of $\mathcal{A}$ is identically distributed in both experiments, and we have

$$\Pr[\mathcal{G}_6^0 \Rightarrow 1] = \Pr[\mathcal{G}_6^1 \Rightarrow 1].$$

Absorbing the two b -dependent reductions into a single reduction costs a factor 2. Combining the bounds from the game hops, we obtain the bound from the theorem. $\square$

D.3 Proof of Theorem 6

Theorem 6. *The AKEM construction $\text{AK}_{\text{s-sa}}$ from Figure 8 offers authenticity. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{s-sa}}$ in game AUTH from Figure 5, there exist two reductions*

Game $G_1^{s,a}; G_2^{s,a}; G_3^s - G_0^s; G_3^s$ 00 $(n, m) \leftarrow \mathcal{O}^2; sk, pk, psk \leftarrow \perp^3$ 01 $CH, XS, XA \leftarrow \emptyset^3$ 02 $N \leftarrow \emptyset$ 03 $H \leftarrow \perp$ 04 Invoke $\mathcal{A}$ 05 Stop with 0 Oracle $\text{Enc}(s, i, pk, ad)$ 06 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$ 07 Require $a \in \text{AU} \iff s \in [n]$ 08 Require $s \in \text{CF} \cup \text{AU} \iff i \in [m]$ 09 Require $a \in \text{CF} \iff pk \in \mathcal{PK}$ 10 $(sk_{\text{SIG}}, h) \leftarrow sk_s$ 11 $n \leftarrow_{\$} \mathcal{N}$ 12 If $n \in N$: Abort 13 $N \leftarrow \{n\}$ 14 $k_{\text{tag}} \leftarrow F(psk_i, \text{tag})$ 15 $c_T \leftarrow_{\$} T.\text{tag}(k_{\text{tag}})$ 16 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_{\text{SIG}}, (n, c_T, h, ad))$ 17 $k_{\text{otp}} \leftarrow F(psk_i, (\text{otp}, n))$ 18 $k_\tau \leftarrow F(psk_i, \text{mac})$ 19 If $i = q_k^*$: 20 $k_\tau \leftarrow_{(\$)} H[\text{mac}, n] ? \mathcal{K}$ 21 $H[\text{mac}, n] \leftarrow k_\tau$ 22 $\tau \leftarrow F(k_\tau, (n, c_T, h, \sigma, ad))$ 23 If $i = q_k^*$: 24 $\tau \leftarrow_{\$} \mathcal{K}$ 25 $H[\text{mac2}, n, c_T, h, \sigma, ad] \leftarrow \tau$ 26 $c \leftarrow (n, c_T, (h, \sigma, \tau) \oplus k_{\text{otp}})$ 27 $k \leftarrow F(psk_i, (\text{key}, c, ad))$ 28 If $i \in [m] \vee \exists r \in [n] : pk_r = pk$: 29 $C \leftarrow \{(s, r, i, ad, c)\}$ 30 Return (k, c) Oracle $\text{Gen}(sk)$ 31 Require $\nexists u \in [n] : pk_u = \mu(sk)$ 32 $n \leftarrow n + 1$ 33 If $sk \in \mathcal{SK}$: 34 $(sk_n, pk_n) \leftarrow (sk, \mu(sk)); XA \leftarrow \{n\}$ 35 Else: 36 $(sk_{\text{SIG}}, pk_n) \leftarrow_{\$} \text{SIG.gen}$ 37 $sk_n \leftarrow (sk_{\text{SIG}}, H(pk))$ 38 Return pk_n Oracle $\text{CorruptA}(u)$ 39 Require $u \in [n]$ 40 $XA \leftarrow \{u\}$ 41 Return sk_u	Oracle $\text{Dec}(r, PK, I, ad, c)$ 42 Require $(r, PK, I) \in ([n] \times \mathcal{P}(\mathcal{PK}) \times \mathcal{P}([m])) \cup \{\perp^3\}$ 43 Require $a \in \text{AU} \iff PK \subseteq \mathcal{PK}$ 44 Require $s \in \text{CF} \cup \text{AU} \iff I \subseteq [m]$ 45 $K' \leftarrow \emptyset; P'[\cdot] \leftarrow \perp$ 46 For all $psk \in (psk_i)_{i \in I}$: 47 $k_{\text{tag}} \leftarrow F(psk_i, \text{tag}); K' \leftarrow \{k_{\text{tag}}\}; P'[k_{\text{tag}}] \leftarrow psk$ 48 $st' \leftarrow_{\$} T.\text{prec}(K)$ 49 $(n, c_T, c') \leftarrow c$ 50 $N \leftarrow \{n\}$ 51 $k_T \leftarrow T.\text{det}(st', c_T); psk \leftarrow P'[k_T];$ Require $psk \neq \perp$ 52 $k_{\text{otp}} \leftarrow F(psk, (\text{otp}, n))$ 53 $(h, \sigma, \tau) \leftarrow c' \oplus k_{\text{otp}}$ 54 $pk \leftarrow \{pk \in PK \mid H(pk) = h\}$ 55 $b \leftarrow \text{SIG.vfy}(pk, (n, c_T, h, ad), \sigma)$ 56 $k_\tau \leftarrow F(psk, \text{mac})$ 57 If $i = q_k^*$: 58 $k_\tau \leftarrow_{(\$)} H[\text{mac}, n] ? \mathcal{K}$ 59 $H[\text{mac}, n] \leftarrow k_\tau$ 60 $\tau' \leftarrow F(k_\tau, (n, c_T, h, \sigma, ad))$ 61 If $i = q_k^*$: 62 $\tau' \leftarrow_{(\$)} H[\text{mac2}, n, c_T, h, \sigma, ad] ? \mathcal{K}$ 63 $H[\text{mac2}, n, c_T, h, \sigma, ad] \leftarrow \tau$ 64 If $b \wedge (s, r, i, ad, c) \notin C \wedge s \notin XA$ 65 $\vee \tau' = \tau \wedge (s, r, i, ad, c) \notin C \wedge i \notin XS$: 66 If $b \wedge (s, r, i, ad, c) \notin C \wedge s \notin XA$: Abort 67 If $i \neq q_k^*$: Abort 68 If $\tau' = \tau \wedge (s, r, i, ad, c) \notin C \wedge i \notin XS$: Abort 69 If $b = \text{fail} \vee \tau \neq \tau' \vee pk = \emptyset$: Return $\perp^3$ 70 $k \leftarrow F(psk, (\text{key}, c, ad))$ 71 If $\exists i' \in I : psk_{i'} = psk$: $i \leftarrow i'$; Else: $i \leftarrow \perp$ 72 If $\exists s' \in [n] : pk_{s'} = pk$: $s \leftarrow s'$; Else: $s \leftarrow \perp$ 73 If $k \neq \perp \wedge (s, r, i, ad, c) \notin C$ 74 $\wedge (a \in \text{AU} \wedge \perp \neq s \notin XA \vee s \in \text{AU} \wedge i \notin XS)$ 75 Stop with 1 76 Return (pk, i, k) Oracle $\text{CorruptS}(i)$ 77 Require $i \in [m]$ 78 $XS \leftarrow \{i\}$ 79 Return psk_i Oracle $\text{Psk}(psk)$ 80 Require $\nexists i \in XS : psk_i = psk$ 81 $m \leftarrow m + 1$ 82 If $psk \in \mathcal{PSK}$: 83 $psk_m \leftarrow psk; XS \leftarrow \{m\}$ 84 Else: $psk_m \leftarrow_{\$} \mathcal{PSK}$ 85 Return
--	--

Fig. 20. Games for the proof of Theorem 6 for AUTH security of AKEM construction from Figure 8 with $\text{CF} = \mathbf{s}$; $\text{AU} = \{\mathbf{s}, \mathbf{a}\}$.

$\mathcal{R}_{F,1}$ as well as $\mathcal{R}_{F,2}$ against the IND security of the PRF F and a reduction $\mathcal{R}_{\text{SIG}}$ against the SUF security against the signature scheme SIG, such that

$$\text{Adv}_{\text{AK}_{\mathbf{s}-\mathbf{sa}}}^{\text{AUTH}}(\mathcal{A}) \leq \frac{q_{e+d}^2}{|\mathcal{N}|} + \text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{R}_{\text{SIG}}) + q_{psk} \left(\sum_{i=1}^2 \text{Adv}_F^{\text{IND}}(\mathcal{R}_{F,i}) + \frac{q_d}{|\mathcal{T}|} \right),$$

with variables as in Theorem 4 and 5, q_{psk} queries to the symmetric key generation oracle, tag space $\mathcal{T}$, and the running time of $\mathcal{R}_F$ and $\mathcal{R}_{\text{SIG}}$ is about that of $\mathcal{A}$.

Proof. We prove that $\text{AK}_{\mathbf{s}-\mathbf{sa}}$ satisfies the AUTH security with a sequence of games. The AUTH definition essentially states that each (non-trivial) forgery attempt by the adversary fails because the asymmetric or the symmetric key that the receiver uses for decapsulation remains uncorrupted. Forgery attempts are the ciphertexts that were not produced by the game internally. For a trivial forgery attempt, both asymmetric and symmetric security primitives are corrupted; otherwise, the forgery attempt is non-trivial and we reduce security to the uncorrupted primitive. Here, our construction uses a pre-shared key psk and a signature scheme SIG for symmetric and asymmetric authenticity, respectively.

First, for the reduction we only consider successful adversaries, i.e., each adversary at some point produces a successful forgery after which the game terminates with 1. This requires a non-trivial forgery attempt where at least one of these keys needs to be uncorrupted, meaning the adversary needs to break one of the primitives. At the time of a forgery attempt, we know which primitive is uncorrupted. This allows us to immediately reduce the security to a valid forgery against the signature scheme or guessing a pre-shared key. In the case that the forgery breaks the signature scheme, the attack can be directly forwarded as a forgery against a multi-instance SUF-CMA challenger. Otherwise, if the pre-shared key is uncorrupted, we reduce the security to the fact that the adversary cannot create a valid, forged tag τ and the verification fails for forgery attempts. We reduce this change to the IND security of the underlying PRF: If the adversary could produce a valid tag, then it could distinguish if we replace the output value of the PRF with a random value. To prepare for this reduction, we need to know for which pre-shared key the forgery is, so that we can replace the key k_τ with a random value (this needs to be consistent across encapsulations and decapsulations). Having ensured that the game is independent of this value, we can then outsource the computations of τ to an external PRF IND challenger. We denote the games as G_i for $i \in [7]$.

Game G_1 is the original $\text{AUTH}_{\text{AK}_{s\text{-}sa}}^{s,\{s,a\}}(\mathcal{A})$ game instantiated with our construction. In particular,

$$\Pr[G_1 \Rightarrow 1] = \Pr[\text{AUTH}_{\text{AK}_{s\text{-}sa}}^{s,\{s,a\}}(\mathcal{A}) \Rightarrow 1]$$

and

$$\text{Adv}_{\text{AK}_{s\text{-}sa}}^{\text{AUTH}}(\mathcal{A}) = \Pr[G_1 \Rightarrow 1].$$

Game G_2 is identical to G_1 except that it aborts if a nonce repeats among encapsulation and decapsulation queries.

G_1 to G_2 : A nonce is sampled uniformly at random in each encapsulation query. Moreover, the adversary can guess with each decapsulation query a new nonce. The probability of nonce collisions is thus bounded by the birthday bound in q_{e+d}^2 number of encapsulation and decapsulation queries and nonce space $\mathcal{N}$. Since G_1 and G_2 are identical until nonce collision event E_n occurs, $\Pr[E_n]$ bounds the gap between the games by

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \frac{q_{e+d}^2}{|\mathcal{N}|}.$$

Game G_3 considers the asymmetric case. G_3 aborts if the adversary successfully forges a valid signature for an uncorrupted signing key sk_s (with $s \notin XA$). We call this event E_f .

G_2 to G_3 : We construct a reduction $\mathcal{R}_{\text{SIG}}$ against the multi-instance SUF_{SIG} security of the signature scheme (cf. Figure 12), which internally runs $\mathcal{A}$ and simulates $\text{AUTH}_{\text{AK}_{s\text{-}sa}}^{s,\{s,a\}}$. We show that if $\mathcal{A}$ triggers E_f , reduction $\mathcal{R}_{\text{SIG}}$ can output a valid forgery. The simulation is as in Figure 20, handling corrupted queries locally and with the following specifications:

- On query from $\mathcal{A}$ to Gen, if $sk = \perp$, reduction $\mathcal{R}_{\text{SIG}}$ queries $pk_{\text{SIG}} \leftarrow \text{SIG.Gen}$. Similar to earlier proofs, we assume the reduction handles index mapping internally, and we omit the specification from Figure 20.
- On query from $\mathcal{A}$ to Enc, reduction $\mathcal{R}_{\text{SIG}}$ simulates the construction locally but obtains $\sigma \leftarrow \text{SIG.Sig}(s, (n, c_T, h, ad))$ for honest senders $s \notin XA$.
- On query from $\mathcal{A}$ to Dec, reduction $\mathcal{R}_{\text{SIG}}$ simulates according to the specification of the game. If E_f is triggered for some honest query (i.e., $s \notin XA$), the reduction $\mathcal{R}_{\text{SIG}}$ aborts the simulation.
- On query from $\mathcal{A}$ to CorruptA(u), reduction $\mathcal{R}_{\text{SIG}}$ forwards the query to $\text{SIG.Corrupt}(u)$ and returns the full secret key.

Eventually, $\mathcal{A}$ triggers E_f , and the reduction aborted the simulation. If so, we have $(s, r, i, ad, c) \notin C$ with $s \notin XA$ and a signature σ , which verifies for some $M = (n, c_T, h, ad)$. Since (s, r, i, ad, c) is fresh, (M, σ) must be fresh. As a consequence, the reduction can win its game by querying its SIG.Vfy oracle on $(s, (n, c_T, h, ad))$. This leads to two cases for the success analysis of $\mathcal{R}_{\text{SIG}}$:

- If M was never signed for s , this is a standard forgery.

- If M was signed, then for fixed n and psk_i, k_{otp} is fixed. Since c' in the forgery must differ from the encapsulated c'_{enc} , the unmasked values $(h, \sigma, \tau) = c' \oplus k_{otp}$ must differ. If σ or h changed, it is a strong forgery.

Consequently, this bound the gap between the games, and we have

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{R}_{\text{SIG}}).$$

Game G_4 is the first game of the symmetric case, in which we assume the first forgery in the game is against an uncorrupted pre-shared key $psk_{q_k^*}$. In this game, we guess the index q_k^* to isolate the specific pre-shared key instance used for the eventual forgery attempt, and abort if our guess is wrong. This guessing step is necessary to reduce the security to a single-instance PRF challenger while allowing the adversary to adaptively corrupt any other pre-shared key instances.

G_3 to G_4 : We guess the index of $q_k^* \in [q_{psk}]$ where q_{psk} is the number of pre-shared key generation queries. We abort if the forgery attempt does not correspond to $psk_{q_k^*}$; note that because we prevented this forgery attempt to break the signature scheme, if we arrive here, the forgery must be against an uncorrupted pre-shared key. Since the choice of q_k^* is independent of the adversary's view, we lose a factor of q_{psk} in the advantage, which bounds the gap between the games as

$$\Pr[G_3 \Rightarrow 1] \leq q_{psk} \cdot \Pr[G_4 \Rightarrow 1].$$

Game G_5 replaces the output of k_τ from $F(psk_p, \text{mac})$ with uniformly random sampled values for the uncorrupted pre-shared key $psk_{q_k^*}$.

G_4 to G_5 : We construct a reduction $\mathcal{R}_{F,1}$ against the multi-instance IND security of the PRF scheme (cf. Figure 11). The reduction simulates the game for $\mathcal{A}$ by querying its PRF challenge oracle for the query by $\mathcal{A}$ on the guessed pre-shared key index q_k^* . Formally, the simulation is as in Figure 20 with the following specifications:

- In the q_k^* -th pre-shared key generation query, instantiate a new PRF challenger through F.Gen.
- On query to Enc and Dec, reduction $\mathcal{R}_{F,1}$ simulates as in the original construction from Figure 4 except that all PRF invocations with psk_p are forwarded to the PRF challenger. The decapsulation computations for k_τ are forwarded to F.Chall, and all other computations (i.e., the ones for k_{tag} and k) are forwarded to F.Eval.
- Per assumption, $psk_{q_k^*}$ is uncorrupted and thus there is no query $\text{CorruptS}(p)$.

If the PRF challenger computes the real PRF, the simulation is as in G_4 ; otherwise, as in G_5 . Finally, $\mathcal{R}_{F,1}$ outputs 1 if $\mathcal{A}$ triggers a successful forgery (the winning condition of the AUTH game) and 0 otherwise. The resulting advantage is then

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_F).$$

Game G_6 replaces the output τ from $F(k_\tau, (n, c_T, h, \sigma, ad))$ with a uniform random sampled value.

G_5 to G_6 : This game hop is similar to the previous hop from G_4 to G_5 . We construct a reduction $\mathcal{R}_{F,2}$ against the multi-instance IND security of the PRF scheme (cf. Figure 11). The reduction simulates the game for $\mathcal{A}$ by using its PRF challenger to evaluate the PRF keyed by the random value k_τ belonging to the guessed instance q_k^* . Formally, the simulation is as in Figure 20 with the following specifications:

- In the q_k^* -th pre-shared key generation query, instantiate a new PRF challenger through F.Gen.
- On query to Enc and Dec, $\mathcal{R}_{F,2}$ simulates as in the original construction from Figure 4 except that the computation for the tag τ is obtained by forwarding (n, c_T, h, σ, ad) to the challenger's F.Chall oracle. All other values (including k_{otp} and k_τ) are handled locally as in G_4 , where k_τ serves as the PRF key.
- Since k_τ is an internal value derived from the uncorrupted $psk_{q_k^*}$ and was replaced by a uniform random value in G_5 , it remains unknown to the adversary, satisfying the security requirements of the PRF challenger. Consequently, there is no query from $\mathcal{A}$ to $\text{CorruptS}(p)$

If the PRF challenger computes the real PRF, the simulation is as in G_5 ; otherwise, as in G_6 . Finally, $\mathcal{R}_{F,2}$ outputs 1 if $\mathcal{A}$ triggers a successful forgery (the winning condition of the AUTH game) and 0 otherwise. The resulting advantage is then

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1]| \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_{F,2}).$$

Game G_7 is the final game, which aborts if the adversary creates a valid forgery where $\tau = \tau'$ under an uncorrupted pre-shared key.

G_6 to G_7 : To bound the advantage of detecting this change, consider the following. The forgery attempt is a new ciphertext, which means that the tuple of ciphertext components $(n, c_T, c' = ((h, \sigma) \oplus k_{\text{otp}}), ad)$ must be different from those of all existing encapsulations. Observe that partial ciphertext replays would count as a forgery attempt. The argument now is that, in order to produce a forgery, the adversary needs to be able to achieve $\tau = \tau'$ in the decapsulation, which happens with negligible probability.

First, k_{otp} is a random value (via the previous games), and since we excluded nonce collisions, the adversary never sees two ciphertexts that use the same one-time pad. That means, if the adversary adapts the ciphertext c' , the resulting τ is a random value. Second, if the adversary adapts the other components of the ciphertext, the resulting τ' is a random value. The probability that both values collide for one query is bounded in the size of the tag space $\mathcal{T}$. Summing over all q_d decapsulation queries, we bound the gap between the games by

$$|\Pr[G_6 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1]| \leq \frac{q_d}{|\mathcal{T}|}.$$

Finally, in G_7 the adversary cannot win, since in game G_3 we abort if the adversary creates asymmetric forgery, and in the game sequence from G_4 to G_7 we abort if the adversary creates symmetric forgery. As a result, in G_7 we excluded all possibilities for the adversary to win and the adversary's success probability is zero, i.e., we have

$$\Pr[G_7 \Rightarrow 1] = 0.$$

Collecting the bounds from all game hops, we obtain that the AUTH advantage of $\mathcal{A}$ is bounded by the sum of the derived hop advantages, which is exactly the claimed bound of the theorem. This concludes the proof. $\square$

E Security of the AKEM with Full Redundancy

E.1 Proof of Theorem 7

Theorem 7. *The AKEM construction $\text{AK}_{\text{sa-sa}}$ from Figure 9 offers key indistinguishability. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{sa-sa}}$ in game IND from Figure 3, there exists a reduction $\mathcal{R}_{\text{KEM}}$ in game OW-KCA from Figure 14 against the KEM scheme KEM in the ROM, such that*

$$\text{Adv}_{\text{AK}_{\text{sa-sa}}}^{\text{IND}_{\text{sa-sa}}}(\mathcal{A}) \leq \frac{2(q_{e+d})^2}{|\mathcal{N}|} + 2 \left(\text{Adv}_{\text{KEM}}^{\text{OW-KCA}}(\mathcal{R}_{\text{KEM}}) + \frac{q_{\text{RO}} q_{\text{psk}}}{|\mathcal{PSK}|} + \frac{2q_{\text{RO}} q_{e+d}}{|\mathcal{K}|} \right),$$

where q_{e+d} are the number of queries to the encapsulation and decapsulation oracle, $\mathcal{N}$ is the nonce space, q_{RO} the number of adversarial RO queries, $\mathcal{PSK}$ the pre-shared key space, $\mathcal{K}$ the space of intermediate keys output by the RO, where the running time of $\mathcal{R}_{\text{KEM}}$ is about that of $\mathcal{A}$.

Proof. We prove the key indistinguishability (see Figure 3) of our fully redundant AKEM construction (Figure 9) via a sequence of games, as shown in Figure 21. The IND definition requires that for a valid challenge encapsulation, at least one of the underlying primitives (the symmetric pre-shared key or the asymmetric KEM receiver key) remains uncorrupted. Since we allow for adaptive corruptions, both primitives of a challenge query might initially be uncorrupted, and the reduction cannot predict which of the two (symmetric or asymmetric side) might be corrupted later in the game. This creates a problem: At challenge time, we do not know which primitive's output should be replaced with a random value. To avoid this problem, we keep both potential corruptions open by using a random oracle with lazy sampling.

Games for $\text{IND}_{\text{AK}}^{b, \text{CF}, \text{AU}}(\mathcal{A})$ 00 $(n, m) \leftarrow 0^2; sk, pk, psk \leftarrow \perp^3; CH, XS, XA \leftarrow \emptyset^3$ 01 $H \leftarrow \perp; AK \leftarrow \emptyset; M \leftarrow \emptyset; N \leftarrow \emptyset$ 02 $b' \leftarrow_{\$} \mathcal{A}$ 03 For all $(s, i, r, ad, c) \in CH$: 04 Require $s \in \text{CF} \wedge i \notin XS \vee a \in \text{CF} \wedge \perp \neq r \notin XA$ 05 Stop with b'	
Oracle $\text{Enc}(s, i, pk, ad, ch)$ 06 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$ 07 Require $a \in \text{AU} \iff s \in [n]$ 08 Require $s \in \text{CF} \cup \text{AU} \iff i \in [m]$ 09 Require $a \in \text{CF} \iff pk \in \mathcal{PK}$ 10 If $\exists r' \in [n] : pk_{r'} = pk : r \leftarrow r'$; Else: $r \leftarrow \perp$ 11 $(sk_{\text{KEM}}, sk_{\text{SIG}}, h) \leftarrow sk_s; (pk_{\text{KEM}}, \_) \leftarrow pk$ 12 $n \leftarrow_{\$} \mathcal{N}$ 13 If $n \in N$: Abort 14 $N \leftarrow_{\$} \{n\}$ 15 $k_{\text{tag}} \leftarrow \text{RO}(psk_i, \text{tag})$ 16 $(k_{\text{KEM}}, c_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$ 17 $(\_, c_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$ 18 $AK \leftarrow_{\$} \{(c_{\text{KEM}}, r)\}$ 19 $c_{\text{T}} \leftarrow_{\$} \text{T.tag}(k_{\text{tag}})$ 20 $k_{\text{otp1}} \leftarrow \text{RO}(k_{\text{KEM}}, \text{otp1}, c_{\text{KEM}}, r, n)$ 21 $k_{\text{otp1}} \leftarrow \text{RO}(\text{otp1}, c_{\text{KEM}}, r, n)$ 22 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_{\text{SIG}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, ad))$ 23 $k_1 \leftarrow \text{RO}(k_{\text{KEM}}, \text{mix}, c_{\text{KEM}}, r, n)$ 24 $k_2 \leftarrow \text{RO}(\text{mix}, c_{\text{KEM}}, r, n)$ 25 $k_{\text{mix}} \leftarrow \text{RO}(psk_i, \text{mix})$ 26 $k_{\text{mix}} \leftarrow \text{RO}(\text{dual}, k_1, k_2)$ 27 If ch : 28 If $\exists a : H[k_{\text{mix}}, a] \neq \perp$: Abort 29 $M \leftarrow_{\$} \{k_{\text{mix}}\}$ 30 $\tau \leftarrow \text{RO}(k_{\text{mix}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, \sigma, ad))$ 31 $k_{\text{otp2}} \leftarrow \text{RO}(k_{\text{mix}}, \text{otp2})$ 32 $c \leftarrow (c_{\text{KEM}}, n, c_{\text{T}} \oplus k_{\text{otp1}}, (h, \sigma, \tau) \oplus k_{\text{otp2}})$ 33 $k_0 \leftarrow \text{RO}(k_{\text{mix}}, (\text{key}, c, ad)); k_1 \leftarrow_{\$} \mathcal{K}$ 34 If ch : $CH \leftarrow_{\$} \{(s, i, r, ad, c)\}$; Return (k_b, c) 35 Return (k_0, c)	Oracle $\text{Dec}(r, PK, I, ad, c)$ 57 Require $(r, PK, I) \in ([n] \times \mathcal{P}(\mathcal{PK}) \times \mathcal{P}([m])) \cup \{\perp^3\}$ 58 Require $a \in \text{AU} \iff PK \subseteq \mathcal{PK}$ 59 Require $s \in \text{CF} \cup \text{AU} \iff I \subseteq [m]$ 60 Require $a \in \text{CF} \iff r \in [n]$ 61 If $\exists (s', i', r, ad, c) \in CH : i' \in I \cup \{\perp\}$ 62 $\wedge (s' = \perp \vee pk_{s'} \in PK)$: Return $\perp$ 63 $K \leftarrow \emptyset; P[\cdot] \leftarrow \perp$ 64 For all $j \in I$: 65 $k_{\text{tag}, j} \leftarrow \text{RO}(psk_j, \text{tag})$ 66 $K \leftarrow_{\$} \{k_{\text{tag}, j}\}; P[k_{\text{tag}, j}] \leftarrow psk_j$ 67 $st' \leftarrow_{\$} \text{T.prec}(K)$ 68 $(sk'_{\text{KEM}}, \_) \leftarrow sk_r; (\_, pk'_{\text{SIG}}) \leftarrow pk$ 69 $(ck_{\text{KEM}}, n, c', c'') \leftarrow c$ 70 $N \leftarrow_{\$} \{n\}$ 71 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, ck_{\text{KEM}})$ 72 $k_{\text{otp1}} \leftarrow \text{RO}(k_{\text{KEM}}, \text{otp1}, c_{\text{KEM}}, r, n)$ 73 $k_{\text{otp1}} \leftarrow \text{RO}(\text{otp1}, c_{\text{KEM}}, r, n)$ 74 $c_{\text{T}} \leftarrow c' \oplus k_{\text{otp1}}$ 75 $k_{\text{T}} \leftarrow \text{T.det}(st', c_{\text{T}})$ 76 $psk \leftarrow P[k_{\text{T}}]$ 77 If $psk = \perp$: Return $\perp^3$ 78 $k_1 \leftarrow \text{RO}(k_{\text{KEM}}, \text{mix}, c_{\text{KEM}}, r, n)$ 79 $k_1 \leftarrow \text{RO}(\text{mix}, c_{\text{KEM}}, r, n)$ 80 $k_2 \leftarrow \text{RO}(psk, \text{mix})$ 81 $k_{\text{mix}} \leftarrow \text{RO}(\text{dual}, k_1, k_2)$ 82 $k_{\text{otp2}} \leftarrow \text{RO}(k_{\text{mix}}, \text{otp2})$ 83 $(h, \sigma, \tau) \leftarrow c'' \oplus k_{\text{otp2}}$ 84 $pk \leftarrow \{pk' \in PK \mid H(pk') = h\}$ 85 $b \leftarrow \text{SIG.vfy}(pk_{\text{SIG}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, ad), \sigma)$ 86 $\tau' \leftarrow \text{RO}(k_{\text{mix}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, \sigma, ad))$ 87 If $b = \text{fail} \vee \tau \neq \tau' \vee pk = \emptyset$: Return $\perp^3$ 88 $k \leftarrow \text{RO}(k_{\text{mix}}, (\text{key}, c, ad))$ 89 If $\exists i' \in [m] : psk_{i'} = psk : i \leftarrow i'$; Else: $i \leftarrow psk$ 90 Return (pk, i, k)
Oracle $\text{RO}(k, l, a)$ 36 $H[k, l, a] \leftarrow_{\$} H[k, l, a] ? \mathcal{K}$ 37 Return $H[k, l, a]$	Oracle $\text{Psk}(psk)$ 91 Require $\nexists i \in XS : psk_i = psk$ 92 $m \leftarrow m + 1$ 93 If $psk \in \mathcal{PSK}$: 94 $psk_m \leftarrow psk; XS \leftarrow_{\$} \{m\}$ 95 Else: $psk_m \leftarrow_{\$} \mathcal{PSK}$ 96 Return
Oracle $\text{RO}^A(k, l, a)$ 38 If $l \in \{\text{otp1}, \text{mix}\} \wedge \exists c_{\text{KEM}}, r : (c_{\text{KEM}}, r) = a$: 39 If $k = \perp$: Return $\perp$ 40 Else if $r \in [n] \wedge k = \text{KEM.dec}(sk_r, sk_{\text{KEM}}, c)$: 41 If $r \in [n] \setminus XA$: Abort 42 $H[l, c_{\text{KEM}}, r] \leftarrow_{\$} H[l, c_{\text{KEM}}, r] ? \mathcal{K}$ 43 Return $H[l, c_{\text{KEM}}, r]$ 44 If $k \in (psk_j)_{j \in [m]} \setminus (psk_i)_{i \in XS}$: Abort 45 If $l = \text{dual}$: 46 $k_1, k_2 \leftarrow a$ 47 If $\exists r, k' : (k', r) \in AK \wedge r \in [n] \setminus XA$ 48 $\wedge H[k', \text{mix}, \perp] = k_1$: Abort 49 If $\exists j \in [m] \setminus XS : H[psk_j, \text{mix}, \perp] = k_2$: 50 Abort 51 If $k \in M$: Abort 52 $H[k, l, a] \leftarrow_{\$} H[k, l, a] ? \mathcal{K}$ 53 Return $H[k, l, a]$	Oracle $\text{Gen}(sk)$ 97 Require $\nexists u \in [n] : pk_u = \mu(sk)$ 98 $n \leftarrow n + 1$ 99 If $sk \in \mathcal{SK}$: 100 $(sk_n, pk_n) \leftarrow (sk, \mu(sk)); XA \leftarrow_{\$} \{n\}$ 101 Else: 102 $(sk_{\text{KEM}}, pk_{\text{KEM}}) \leftarrow_{\$} \text{KEM.gen}$ 103 $(sk_{\text{SIG}}, pk_{\text{SIG}}) \leftarrow_{\$} \text{SIG.gen}$ 104 $sk_n \leftarrow (sk_{\text{KEM}}, sk_{\text{SIG}}, H(pk_{\text{SIG}}))$ 105 $pk_n \leftarrow (pk_{\text{KEM}}, pk_{\text{SIG}})$ 106 Return pk_n
Oracle $\text{CorruptS}(i)$ 54 Require $i \in [m]$ 55 $XS \leftarrow_{\$} \{i\}$ 56 Return psk_i	Oracle $\text{CorruptA}(u)$ 107 Require $u \in [n]$ 108 $XA \leftarrow_{\$} \{u\}$ 109 Return sk_u

Fig. 21. Games for the proof of Theorem 7. We denote the games as G_i for $i \in [5]$. The grey parts emphasize parts that are the same in the original $\text{IND}_{\text{AK}}^{b, \text{CF}, \text{AU}}(\mathcal{A})$ game.

Again, we use a sequence of games, but instead of splitting the proof into a case distinction for the symmetric and asymmetric parts, we employ a sequential series of game hops. This sequential approach is necessary for the multi-challenge setting; an adversary might issue multiple challenge queries relying on different uncorrupted primitives, which would complicate branching and recom-

binning the game execution. To prove indistinguishability, our game sequence progressively replaces the real derived keys with uniformly random values. We rely on the definition that a valid challenge guarantees at least one primitive remains uncorrupted, allowing us to leverage RO checks sequentially to ensure the adversary never learns the underlying secrets. First, we guarantee that the adversary cannot deduce an uncorrupted KEM secret key without breaking the OW-KCA security of the underlying KEM scheme. Second, we show that the adversary has at most a guessing advantage for finding an uncorrupted pre-shared key. Because both the asymmetrically and symmetrically derived keys (k_1 and k_2) are used as inputs to the KDF (modeled as an RO), the final challenge key is uniformly random as long as at least one of these inputs remains perfectly hidden. For ease of bookkeeping, we introduce the keyword `dual` as context for querying k_{mix} . We denote the games as G_i for $i \in [6]$. For $b \in \{0, 1\}$, let $\Pr[G_i^b \Rightarrow 1]$ denote the probability that $\mathcal{A}$ outputs 1 in game G_i with the challenge bit fixed to b .

Game G_1 is the original $\text{IND}_{\text{AK}}^{b, \text{CF}, \text{AU}}(\mathcal{A})$ experiment (see Figure 3) instantiated with the construction from Figure 9.

$$\Pr[G_1^b \Rightarrow 1] = \Pr[\text{IND}_{\text{AK}}^{b, \text{CF}, \text{AU}}(\mathcal{A}) \Rightarrow 1]$$

and

$$\text{Adv}_{\text{AK}}^{\text{IND}, \text{CF}, \text{AU}}(\mathcal{A}) = |\Pr[G_1^0 \Rightarrow 1] - \Pr[G_1^1 \Rightarrow 1]|.$$

Game G_2 is identical to G_1 , except that it aborts if a nonce is repeated across encapsulation and decapsulation queries.

G_1 to G_2 : A nonce is sampled uniformly at random in each encapsulation query. Moreover, the adversary can guess with each decapsulation query a new nonce. The probability of nonce collisions is thus bounded by the birthday bound over q_e encapsulation queries, q_d decapsulation queries, and the nonce space $\mathcal{N}$. Since G_1 and G_2 are identical until nonce collision event E_n occurs, $\Pr[E_n]$ bounds the gap between the games by

$$|\Pr[G_1^b \Rightarrow 1] - \Pr[G_2^b \Rightarrow 1]| \leq \frac{(q_e + q_d)^2}{|\mathcal{N}|}.$$

Game G_3 is identical to G_2 except we abort if the adversary queries the random oracle RO for a pre-shared key which they did not corrupt before. Because the uncorrupted pre-shared key is never queried to the random oracle, its output k_2 from $\text{RO}(psk_i, \text{mix})$ is perfectly hidden. In the random oracle model, the output of an unqueried point is distributed identically to a uniformly random value, allowing us to replace k_2 for the challenge with a uniformly random sampled value.

G_2 to G_3 : The probability of the adversary querying an uncorrupted key is bounded by the pre-shared key space $\mathcal{PSK}$, the number of pre-shared key generation queries q_{psk} , and the number of adversarial queries to the RO q_{RO} .

$$|\Pr[G_2^b \Rightarrow 1] - \Pr[G_3^b \Rightarrow 1]| \leq \frac{q_{psk} q_{\text{RO}}}{|\mathcal{PSK}|}.$$

Game G_4 is identical to G_3 except we abort if the adversary queries the random oracle RO with the real symmetric key k_{KEM} encapsulated within the challenge KEM ciphertext c_{KEM} . With this hop, we make the game independent of the actual value of the key k_{KEM} : The game internally only queries the RO with the identifying context (ciphertext c_{KEM} and receiver index r); the adversary, on the other hand, must always also provide a key k_{KEM} to obtain the corresponding RO entry. With the reduction, we show that the adversary can only learn the output of the RO which is derived from k_{KEM} if it breaks the OW-KCA of the underlying KEM. Then, because the uncorrupted key k_{KEM} is never queried to the random oracle, its output k_1 from $\text{RO}(k_{\text{KEM}}, \text{mix})$ is perfectly hidden. In the random oracle model, the output of an unqueried point is distributed identically to a uniformly random value, allowing us to replace k_1 for the challenge with a uniformly random sampled value.

G_3 to G_4 : We construct a reduction $\mathcal{R}_{\text{KEM}}^b$ against $\text{OW-KCA}_{\text{KEM}}$ (cf. Figure 14). Reduction $\mathcal{R}_{\text{KEM}}^b$ runs adversary $\mathcal{A}$ as a subroutine and simulates G_3^b while using its $\text{OW-KCA}_{\text{KEM}}$ -oracles to detect whether $\mathcal{A}$ ever “hits” a real KEM key belonging to a challenge encapsulation. The simulation is as follows:

- On query to Gen where $\mathcal{A}$ requests an honest AKEM key pair, $\mathcal{R}_{\text{KEM}}^b$ calls $\text{OW-KCA}_{\text{KEM}}.\text{Gen}$ to obtain pk_{KEM} , samples $(sk_{\text{SIG}}, pk_{\text{SIG}}) \leftarrow_{\$} \text{SIG.gen}$, sets $h \leftarrow H(pk_{\text{SIG}})$ and returns $pk = (pk_{\text{KEM}}, pk_{\text{SIG}})$. It keeps the corresponding (sk_{SIG}, h) internally.
- On query to Enc, $\mathcal{R}_{\text{KEM}}^b$ simulates G_3 except that if queries the OW-KCA challenger for encapsulations with $c_{\text{KEM}} \leftarrow \text{OW-KCA}_{\text{KEM}}.\text{Enc}(r)$.
- On query to CorruptA, $\mathcal{R}_{\text{KEM}}^b$ forwards to $\text{OW-KCA}_{\text{KEM}}.\text{Expose}(u)$ to obtain $sk_{\text{KEM},u}$, returns $sk_u = (sk_{\text{KEM},u}, sk_{\text{SIG},u}, h_u)$.
- On query $\text{Dec}(r, PK, I, ad, c)$, $\mathcal{R}_{\text{KEM}}^b$ makes all queries to the RO which include k_{KEM} without this key itself but only through the context, as in the encapsulation. Note that the adversary always needs to make full queries, including the shared key, to the RO to obtain the RO output as described in the next item.
- On query $\text{RO}(k, l, c_{\text{KEM}}, r)$ where $l \in \{\text{mix}, \text{otp1}\}$, forward to the OW-KCA challenger’s check interface $\text{OW-KCA}_{\text{KEM}}.\text{Check}(r, c_{\text{KEM}}, k)$. If the secret KEM key was not corrupted and the adversary produced a valid decapsulation key, they win the OW-KCA game. This is also exactly where G_4 aborts.

Let E_{KM} be the event that $\mathcal{A}$ makes some query such that k equals the real KEM key of some $(r, c_{\text{KEM}}) \in XC$ while $r \notin XA$. By construction, $\mathcal{R}_{\text{KEM}}^b$ wins the $\text{OW-KCA}_{\text{KEM}}$ game whenever E_{KM} occurs, hence $\Pr[E_{\text{KM}}] \leq \text{Adv}_{\text{KEM}}^{\text{OW-KCA}}(\mathcal{R}_{\text{KEM}}^b)$. Moreover, G_3 and G_4 are identical until E_{KM} occurs. Therefore,

$$|\Pr[G_3^b \Rightarrow 1] - \Pr[G_4^b \Rightarrow 1]| \leq \Pr[E_{\text{KM}}] \leq \text{Adv}_{\text{KEM}}^{\text{OW-KCA}}(\mathcal{R}_{\text{KEM}}^b).$$

Game G_5 is identical to G_4 except that we abort if the adversary queries the random oracle RO with the label **dua1** and keys (k_1, k_2) , where at least one key (k_1 or k_2) is derived from an uncorrupted primitive of a challenge encapsulation.

G_4 to G_5 : By the definition of the IND game, for any challenge encapsulation, at least one primitive must remain uncorrupted. An uncorrupted primitive means that either k_2 (via G_3) or k_1 (via G_4) was replaced by a uniformly random value and is perfectly hidden from the adversary. The adversary can only detect this change if they guess the respective hidden key correctly. The probability of the adversary guessing the correct key is bounded by the size of the key space $\mathcal{K}$, the number of challenge encapsulations and decapsulations q_{e+d} , and the number of RO queries q_{RO} .

$$|\Pr[G_4^b \Rightarrow 1] - \Pr[G_5^b \Rightarrow 1]| \leq \frac{q_{e+d}q_{\text{RO}}}{|\mathcal{K}|}.$$

Game G_6 is identical to G_5 except we abort if the adversary queries the random oracle RO with a key k_{mix} which is used in a challenge query.

G_5 to G_6 : Per definition, a challenge query means that at least one of the primitives is uncorrupted. In other words, the adversary may only legitimately query this key if they corrupted both primitives, which can never happen for a challenge query. We ensured through the previous games that the input leading to the derivation of k_{mix} (either k_1 or k_2) is uncorrupted and thus perfectly hidden. Therefore, k_{mix} is a uniformly random value from the adversary’s perspective, and their best strategy is to guess. The advantage is bounded by the size of the key space $\mathcal{K}$, the number of challenge encapsulations and decapsulations q_{e+d} , and the number of RO queries from the adversary q_{RO} .

$$|\Pr[G_5^b \Rightarrow 1] - \Pr[G_6^b \Rightarrow 1]| \leq \frac{q_{e+d}q_{\text{RO}}}{|\mathcal{K}|}.$$

Finally, observe that in the decapsulation, the output key is generated by inputting the whole ciphertext and context into the RO. Thereby, the adversary can only get the decapsulation of an encapsulated key if they query a challenge ciphertext – which is prohibited per definition. Then, through this sequence of games, we ensured that the output key k_0 is a random value, such as k_1 , and therefore

$$\Pr[G_6^0 \Rightarrow 1] = \Pr[G_6^1 \Rightarrow 1].$$

Absorbing the b -dependent reductions into a single reduction yields the claimed bound. $\square$

E.2 Proof of Theorem 8

Games for $\text{ANON}_{\text{AK}}^{\text{b}, \text{CF}, \text{AU}}(\mathcal{A})$	
00 $(n, m) \leftarrow 0^2; sk, pk, psk \leftarrow \perp^3$	
01 $CH, XS, XA \leftarrow \emptyset^3$	
02 $H \leftarrow \perp$	
03 $q \leftarrow 0; Q \leftarrow \emptyset$	
04 $KQ[\cdot] \leftarrow \perp; IQ[\cdot] \leftarrow \perp; RQ[\cdot] \leftarrow \perp$	
05 $ZQ[\cdot] \leftarrow \perp; CQ[\cdot] \leftarrow \perp; \text{Case}[\cdot] \leftarrow \perp$	
06 $AK \leftarrow \emptyset$	
07 $b' \leftarrow_{\$} \mathcal{A}$	
08 For all $(s, i, r, ad, c) \in CH$:	
09 Require $s \in \text{CF} \wedge i \notin XS \vee a \in \text{CF} \wedge r \notin XA \vee (r \neq \perp \vee i \notin XS)$	
10 Stop with b'	
Oracle $\text{Enc}(s, i, pk, ad, ch)$	
11 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$	
12 Require $a \in \text{AU} \iff s \in [n]$	
13 Require $s \in \text{CF} \cup \text{AU} \iff i \in [m]$	
14 Require $a \in \text{CF} \iff pk \in \mathcal{PK}$	
15 If $\exists r' \in [n] : pk_{r'} = pk : r \leftarrow r'$; Else: $r \leftarrow \perp$	
16 $(\_, sk_{\text{SIG}}, h) \leftarrow sk_s$	
17 $(pk_{\text{KEM}}, \_) \leftarrow pk$	
18 $(k_{\text{KEM}}, c_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$	
19 $n \leftarrow_{\$} \mathcal{N}$	
20 If $n \in N$: Abort	
21 $N \leftarrow \{n\}$	
22 $k_{\text{tag}} \leftarrow \text{RO}(psk_i, \text{tag})$	
23 $(k_{\text{KEM}}, c_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$	
24 $(\_, c_{\text{KEM}}) \leftarrow_{\$} \text{KEM.enc}(pk_{\text{KEM}})$	
25 $c_{\text{T}} \leftarrow_{\$} \text{T.tag}(k_{\text{tag}})$	
26 $k_{\text{otp1}} \leftarrow \text{RO}(k_{\text{KEM}}, \text{otp1}, c_{\text{KEM}}, r, n)$	
27 $k_{\text{otp1}} \leftarrow \text{RO}(\text{otp1}, c_{\text{KEM}}, r, n)$	
28 $\sigma \leftarrow_{\$} \text{SIG.sig}(sk_{\text{SIG}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, ad))$	
29 $k_1 \leftarrow \text{RO}(k_{\text{KEM}}, \text{mix}, c_{\text{KEM}}, r, n)$	
30 $k_1 \leftarrow \text{RO}(\text{mix}, c_{\text{KEM}}, r, n)$	
31 $k_2 \leftarrow \text{RO}(psk_i, \text{mix})$	
32 $k_{\text{mix}} \leftarrow \text{RO}(\text{dual}, k_1, k_2)$	
33 $\tau \leftarrow \text{RO}(k_{\text{mix}}, (c_{\text{KEM}}, n, c_{\text{T}}, h, \sigma, ad))$	
34 $k_{\text{otp2}} \leftarrow \text{RO}(k_{\text{mix}}, \text{otp2})$	
35 $c_0 \leftarrow (c_{\text{KEM}}, n, c_{\text{T}} \oplus k_{\text{otp1}}, (h, \sigma, \tau) \oplus k_{\text{otp2}})$	
36 If ch :	
37 $c_4 \leftarrow_{\$} \mathcal{R}$	
38 $C_{r,4}[c_0] \leftarrow (h, \sigma, \tau) \oplus k_{\text{otp2}}$	
39 $c_3 \leftarrow_{\$} \mathcal{R}$	
40 $C_{r,3}[c_0] \leftarrow c_{\text{T}} \oplus k_{\text{otp1}}$	
41 $c_1 \leftarrow_{\$} \mathcal{R}$	
42 $C_{r,1}[c_0] \leftarrow c_{\text{T}} \oplus k_{\text{otp1}}$	
43 $c_0 \leftarrow (c_1?c_{\text{KEM}}, n, c_3?c_{\text{T}} \oplus k_{\text{otp1}}, c_4?(h, \sigma, \tau) \oplus k_{\text{otp2}})$	
44 $k \leftarrow \text{RO}(k_{\text{mix}}, (\text{key}, c_0, ad))$	
45 $c_1 \leftarrow_{\$} \mathcal{C}$	
46 If ch :	
47 $CH \leftarrow \{(s, i, r, ad, c_b)\}$	
48 Return (k, c_b)	
49 Return (k, c_0)	
Oracle $\text{RO}(k, l, a)$	
50 $H[k, l, a] \leftarrow_{(\$)} H[k, l, a] ? \mathcal{K}$	
51 Return $H[k, l, a]$	
Oracle $\text{RO}^A(k, l, a)$	
52 If $l \in \{\text{otp1}, \text{mix}\} \wedge \exists c_{\text{KEM}}, r : (c_{\text{KEM}}, r) = a$:	G ₄
53 If $k = \perp$: Return $\perp$	G ₃
54 Else if $r \in [n] \wedge k = \text{KEM.dec}(sk_r, sk_{\text{KEM}}, c)$:	G ₃
55 If $r \in [n] \setminus XA$: Abort	G ₃
56 $H[l, c_{\text{KEM}}, r] \leftarrow_{(\$)} H[l, c_{\text{KEM}}, r] ? \mathcal{K}$	G ₃
57 Return $H[l, c_{\text{KEM}}, r]$	G ₃
58 If $k \in (psk_j)_{j \in [m]} \setminus (psk_{\ell})_{\ell \in XS}$: Abort	G ₂
59 $H[k, l, a] \leftarrow_{(\$)} H[k, l, a] ? \mathcal{K}$	
60 Return $H[k, l, a]$	
Oracle $\text{Dec}(r, PK, I, ad, c)$	
61 Require $(r, PK, I) \in ([n] \times \mathcal{P}(\mathcal{PK}) \times \mathcal{P}([m])) \cup \{\perp^3\}$	
62 Require $a \in \text{AU} \iff PK \subseteq \mathcal{PK}$	
63 Require $s \in \text{CF} \cup \text{AU} \iff I \subseteq [m]$	
64 Require $a \in \text{CF} \iff r \in [n]$	
65 If $\exists (s', i', r, ad, c) \in CH : i' \in I \cup \{\perp\}$ $\wedge (s' = \perp \vee pk_{s'} \in PK)$: Return $\perp$	
66 $K \leftarrow \emptyset; P[\cdot] \leftarrow \perp$	
67 For all $j \in I$:	
68 $k_{\text{tag},j} \leftarrow \text{RO}(psk_j, \text{tag})$	
69 $K \leftarrow \{k_{\text{tag},j}\}; P[k_{\text{tag},j}] \leftarrow psk_j$	
70 $st' \leftarrow_{\$} \text{T.prec}(K)$	
71 $(sk_{\text{KEM}}, \_, \_) \leftarrow sk_r$	
72 $(c_{\text{KEM}}, n, c_3, c_4) \leftarrow c$	
73 If $C_{r,4}[c] \neq \perp$:	G ₄
74 If $c_4 \leftarrow C_{r,4}[c]$	G ₄
75 If $C_{r,3}[c] \neq \perp$:	G ₅
76 If $c_3 \leftarrow C_{r,3}[c]$	G ₅
77 If $C_{r,1}[c] \neq \perp$:	G ₆
78 If $c_1 \leftarrow C_{r,1}[c]$	G ₆
79	
80 $N \leftarrow \{n\}$	G ₁
81 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, c_{\text{KEM}})$	G ₁ -G ₂
82 $k_{\text{otp1}} \leftarrow \text{RO}(k_{\text{KEM}}, \text{otp1}, c_{\text{KEM}}, r, n)$	G ₁ -G ₂
83 $k_{\text{otp1}} \leftarrow \text{RO}(\text{otp1}, c_{\text{KEM}}, r, n)$	G ₃
84 $c_{\text{T}} \leftarrow c' \oplus k_{\text{otp1}}$	G ₁ -G ₂
85 $k_{\text{T}} \leftarrow \text{T.det}(st', c_{\text{T}})$	G ₃
86 $psk \leftarrow P[k_{\text{T}}]$	
87 If $psk = \perp$: Return $\perp^3$	
88 $k_1 \leftarrow \text{RO}(k_{\text{KEM}}, \text{mix}, c_{\text{KEM}}, r, n)$	G ₁ -G ₂
89 $k_1 \leftarrow \text{RO}(\text{mix}, c_{\text{KEM}}, r, n)$	G ₃
90 $k_2 \leftarrow \text{RO}(psk, \text{mix2})$	
91 $k_{\text{mix}} \leftarrow \text{RO}(\text{dual}, (k_1, k_2))$	
92 $k_{\text{otp2}} \leftarrow \text{RO}(k_{\text{mix}}, \text{otp2})$	
93 $(h, \sigma, \tau) \leftarrow c_4 \oplus k_{\text{otp2}}$	
94 $pk \leftarrow \{(\_, pk_{\text{SIG}}) \in PK \mid H(pk_{\text{SIG}}) = h\}$	
95 If $pk = \emptyset$: Return $\perp^3$	
96 Choose $(\_, pk_{\text{SIG}}) \in pk$	
97 $b_{\text{SIG}} \leftarrow \text{SIG.vfy}(pk_{\text{SIG}}, (c_{\text{KEM}}, h, ad), \sigma)$	G ₄
98 $\tau' \leftarrow \text{RO}(k_{\text{mix}}, \text{tg}, (c_{\text{KEM}}, n, h, \sigma, ad))$	G ₄
99 If $b_{\text{SIG}} = \text{fal} \vee \tau \neq \tau'$: Return $\perp^3$	G ₄
100 $k \leftarrow \text{RO}(k_{\text{mix}}, \text{key}, (c, ad))$	G ₅
101 If $\exists i' \in [m] : psk_{i'} = psk : i \leftarrow i'$; Else: $i \leftarrow psk$	G ₅
102 Return (pk, i, k)	G ₆
Oracle $\text{CorruptS}(i)$	
103 Require $i \in [m]$	G ₄
104 $XS \leftarrow \{i\}$	
105 Return psk_i	
Oracle $\text{CorruptA}(u)$	
106 Require $u \in [n]$	
107 $XA \leftarrow \{u\}$	
108 Return sk_u	
Oracle $\text{Gen}(sk)$	
109 See Figure 21.	

Fig. 22. Overview for the proof of Theorem 8. The grey parts emphasize parts that are the same in the original $\text{ANON}_{\text{AK}}^{\text{b}, \text{CF}, \text{AU}}(\mathcal{A})$ game.

Theorem 8. *The AKEM construction $\text{AK}_{\text{sa-sa}}$ from Figure 9 offers anonymity. More precisely, for every adversary $\mathcal{A}$ against AK in game ANON from Figure 4, there exist a reduction $\mathcal{R}_{\text{T}, \text{IND}}$ against the IND security of the tagging scheme T, a reduction $\mathcal{R}_{\text{KEM}, \text{EXP-ANON}}$ against the EXP-ANON security of the KEM scheme KEM, and a reduction $\mathcal{R}_{\text{KEM}, \text{OW-KCA}}$ against the OW-KCA security*

of the KEM scheme KEM in the ROM, such that

$$\begin{aligned} \text{Adv}_{\text{AK}_{\text{sa-sa}}}^{\text{ANON}}(\mathcal{A}) &\leq \frac{(q_{e+d})^2}{|\mathcal{N}|} + 2\left(\frac{q_{\text{RO}}q_{\text{psk}}}{|\mathcal{PSK}|} + \text{Adv}_{\text{KEM}}^{\text{OW-KCA}}(\mathcal{R}_{\text{KEM,OW-KCA}})\right) \\ &\quad + 2\text{Adv}_{\text{KEM}}^{\text{EXP-ANON}}(\mathcal{R}_{\text{KEM,EXP-ANON}}) + \text{Adv}_{\text{T}}^{\text{IND}}(\mathcal{R}_{\text{T,IND}}), \end{aligned}$$

with variables as in Theorem 7 and the running time of the reductions is about that of $\mathcal{A}$.

Proof. We prove anonymity, i.e., the ciphertext indistinguishability (see Figure 4), of our construction from Figure 9 with a sequence of games, as shown in Figure 22. The anonymity definition essentially states that an encapsulation ciphertext should be indistinguishable from random if at least one of the asymmetric or symmetric primitive remains uncorrupted. Now, consider the four parts of the returned ciphertext $(c_{\text{KEM}}, n, c_{\text{T}} \oplus k_{\text{otp1}}, (h, \sigma, \tau) \oplus k_{\text{otp2}})$: We reduce the anonymity of c_{KEM} to the EXP-ANON security of the KEM scheme; the nonce n is random per definition; $c_{\text{T}} \oplus k_{\text{otp1}}$ looks random because c_{T} is random (IND security of tagging scheme) or k_{otp1} derived from KEM key is random (indistinguishability of PRF modeled as RO and OW-KCA security of KEM scheme); and $(h, \sigma, \tau) \oplus k_{\text{otp2}}$ looks random because k_{otp2} is derived from the pre-shared symmetric key and the KEM key (indistinguishability of PRF modeled as RO and at least one of: secrecy of pre-shared key, OW-KCA security of KEM scheme). Through a sequence of games, we now show that the returned ciphertext is anonymous.

Game G_1 is the original $\text{ANON}_{\text{AK}_{\text{s-sa}}}^{b,\text{s},\{\text{s},\text{a}\}}(\mathcal{A})$ game instantiated with our construction except that we prohibit nonce collisions as in Appendix D.1. Let q_{e+d} be number of encapsulation and decapsulation queries and $\mathcal{N}$ the nonce space. Then,

$$\Pr[G_1^b \Rightarrow 1] = \Pr[\text{ANON}_{\text{AK}_{\text{s-sa}}}^{b,\text{s},\{\text{s},\text{a}\}}(\mathcal{A}) \Rightarrow 1] + \frac{(q_{e+d})^2}{|\mathcal{N}|}$$

Game G_2 is identical to G_1 , except that we abort if the adversary queries the RO with an input that contains an uncorrupted pre-shared key psk (i.e., a query of form $\text{RO}(\text{psk}, \_)$). Consequently, the respective outputs of the game's internal queries of form $\text{RO}(\text{psk}, \_)$ yield a random output (per definition of the random oracle).

G_1 to G_2 : Since pre-shared keys are randomly sampled, the probability that the adversary correctly guesses a pre-shared key is bounded by the number of generated pre-shared keys q_{psk} and the number of adversarial RO queries q_{RO} in the pre-shared key space $\mathcal{PK}$.

$$|\Pr[G_2^b \Rightarrow 1] - \Pr[G_1^b \Rightarrow 1]| \leq \frac{q_{\text{RO}}q_{\text{psk}}}{|\mathcal{PSK}|}.$$

Game G_3 is identical to G_2 , except that we abort if the adversary queries the RO with an input that contains the symmetric KEM key k_{KEM} for an encapsulation whose secret asymmetric key is uncorrupted (i.e., a query of form $\text{RO}(k_{\text{KEM}}, \_)$). Consequently, if the adversary cannot compute k_{KEM} , it cannot query the RO at the positions to learn the game's internal keys k_{otp1}, k_1 ; these values thus are random to the adversary (per definition of the random oracle).

G_2 to G_3 : We can reduce this change to breaking OW-KCA security of the underlying KEM scheme KEM. The reduction is the same as the reduction for hop G_3 to G_4 : from Appendix E.1. The game aborts if the adversary breaks the OW-KCA security of the KEM scheme.

$$|\Pr[G_3^b \Rightarrow 1] - \Pr[G_2^b \Rightarrow 1]| \leq \text{Adv}_{\text{KEM}}^{\text{OW-KCA}}(\mathcal{R}_{\text{KEM,OW-KCA}}).$$

Game G_4 is identical to G_3 , except that we replace the fourth part of challenge ciphertexts with a random value. Since we require that for challenge encapsulation at least one of the primitives remains uncorrupted, it holds that one of the keys k_1 and k_2 is unknown to the adversary (through the above game hops). This also means that k_{mix} , derived from k_1 and k_2 , is random and unknown to the adversary and thereby also k_{otp2} , derived from k_{mix} . Consequently, the last part of a challenge ciphertext c is random per XOR with the one-time-pad k_{otp2} . Since k_{otp2} is already a random value, as argued above, this does not change the output for the adversary.

G_3 to G_4 :

$$\Pr[G_4^b \Rightarrow 1] = \Pr[G_3^b \Rightarrow 1]$$

Remark. Now, let us consider the third part, $c_3 = c_T \oplus k_{\text{otp1}}$. Through the following game hop, we show that c_3 in challenges looks random to the adversary. For challenge encapsulations, we have the following potential scenarios at the time of the query:

1. The pre-shared key is corrupted and the asymmetric keys are uncorrupted. In this case, the adversary can know the value of k_{otp2} and all anonymity comes from the indistinguishability of c_T .
2. The asymmetric keys are uncorrupted and the pre-shared key is corrupted. In this case, we do not require any anonymity from the tag c_T since the corresponding secret key is known to the adversary. Instead, all anonymity comes from k_{otp2} which is unknown to the adversary and serves as a one-time-pad.
3. Both keys are uncorrupted (but can potentially be corrupted later). In this case, anonymity comes (at least) from k_{otp2} , which is a random value, as shown in G_3 — until it potentially becomes corrupted. For the case of corruption, we show that anonymity is then given through anonymity of c_T .

In cases (1) and (2), we can immediately reduce to the security of the respective other primitive, showing that c_T or k_{otp1} is random, because we know, per definition of ANON, that the corresponding secret key cannot be exposed later. In case (3), we need to prepare for the possibility of one of the primitives being corrupted later. If the asymmetric key is uncorrupted, we always have anonymity for the resulting ciphertext (this especially holds if both primitives for a challenge are never corrupted). In the following, we reduce to the indistinguishability of the underlying tagging scheme as follows.

Game G_5 is identical to G_4 , except that we replace the third part of challenge encapsulations, $c_T \oplus k_{\text{otp1}}$, with a random value c_3 . First, observe that, as long as both symmetric and asymmetric keys remain uncorrupted, this change is undetectable to the adversary. That is because k_{otp1} is a random value from the RO which the adversary never queried (game 3). Specifically, this means that the game does not need to compute a value for c_T as long as both primitives are uncorrupted. Moreover, since (h, σ, τ) are hidden through k_{otp2} anyways (cf. previous game hop), the value c_T needed for their computations is irrelevant. In the following, we use this fact to retroactively program the RO if the adversary queried a challenge with both primitives uncorrupted and later exposes one of them. *Note that if at the time of the challenge, one of the primitives was corrupted (cf. cases (1) and (2) in the above case distinction), we can immediately proceed as in the following case distinction.* To prove that an adversary has negligible advantage in detecting this game hop, we reduce to the IND security of the tagging scheme.

G_4 to G_5 : — On query Gen (with $psk = \perp$), forward to $\text{IND}_T.\text{Gen}$.

— On query Enc and Dec, forward to $\text{IND}_T.\text{Tag}$ and $\text{IND}_T.\text{Det}$. In case (1) or (2), where one key is corrupted, directly implement the below case distinction into the queries.

G_4 to G_5 : **Case 1**, pre-shared key corrupted. If the pre-shared key psk is corrupted, we compute c_T by querying the corresponding tagging challenger.

- On query CorruptS, forward to $\text{IND}_T.\text{Corrupt}$ and embed the returned k in the RO entry $\text{RO}(psk, \text{tag})$. (Note that we prevented the adversary from querying this value before, but now they can do it since they know the pre-shared key.)
- Find all challenge ciphertexts (c, ad) and the used KEM key k_{KEM} which were produced with this pre-shared key psk . Find the randomness c_3 used to replace this challenge ciphertext part. Then, query the corresponding tagging challenger for a non-challenge tag t for psk . Program the RO such that $\text{RO}(k_{\text{KEM}}, \text{otp1}) = c_3 \oplus t$.
- On query Enc for an encapsulation challenge with the corrupted pre-shared key, query the tagging challenger for a non-challenge tag.

Anonymity is given through KEM security as per game 3.

$$\Pr[G_5^b \Rightarrow 1] = \Pr[G_4^b \Rightarrow 1]$$

G_4 to G_5 : **Case 2**, KEM (asymmetric) key corrupted. If the KEM key k_{KEM} is corrupted, we compute c_T by querying the corresponding tagging challenger for a challenge query.

- On query CorruptA, return the internal KEM key sk . (Now, the adversary can use sk and c_{KEM} to compute k_{KEM} . Note that we prevented the adversary from querying the RO with k_{KEM} before, but now they can do it since they know the KEM key.)

- Find all challenge ciphertexts (c, ad) and the used pre-shared key psk which were produced with the KEM key k_{KEM} . Find the randomness c_3 used to replace this challenge ciphertext part. Then, query the corresponding tagging challenger for a challenge tag t for psk . Program the RO such that $\text{RO}(k_{\text{KEM}}, \text{otp1}) = r_j \oplus t$.
- On query Enc for an encapsulation challenge with the corrupted pre-shared key, query the tagging challenger for a challenge tag.

Observe that the pre-shared key is only used to ask the tagging challenger for a challenge ciphertext once the KEM key is corrupted. Per ANON definition, this means that the pre-shared key, and thereby the tagging challenger instance, cannot be corrupted anymore thereby complying with the restriction that we are not allowed to corrupt a tagging challenger once it has produced a challenge ciphertext. If the adversary can distinguish the game hops, then we can build a reduction $\mathcal{R}_{\text{T,IND}}$ against the IND security of the underlying tagging scheme with the following advantage.

$$|\Pr[G_5^b \Rightarrow 1] - \Pr[G_4^b \Rightarrow 1]| \leq \text{Adv}_{\text{T}}^{\text{IND}}(\mathcal{R}_{\text{T,IND}}).$$

Game G_6 Through the above game hops, we showed that the last three parts of challenge encapsulations are indistinguishable from random for an adversary. Finally, we show that the first part is, as well, by replace the KEM ciphertext c_{KEM} with a random value. We reduce this to the EXP-ANON security of the KEM scheme through the following reduction $\mathcal{R}_{\text{KEM,EXP-ANON}}$. item[G_5 to G_6 :]

- On query Gen (with $sk = \perp$), forward to EXP-ANON.Gen.
- On query Enc, forward the KEM encapsulation to the KEM challenger as indicated by the public key to EXP-ANON.Enc and embed the returned ciphertext c_{KEM} in the encapsulation output. Sample a random value as replacement for k_{KEM} in internal queries to the RO. Then, store the resulting ciphertext, index of public key, and this random value. (Per game 3, the adversary never queries the RO for this value.)
- On query Corrupt, forward to EXP-ANON.Corrupt and embed the returned k_{KEM} in all RO entries that correspond to encapsulation queries for the respective challenger and in the returned state.
- On query Dec, proceed as described in the game which is independent of the decapsulated key k_{KEM} .
- Forward the output of the adversary b' to the KEM challenger.

If the adversary wins the IND game, it breaks the EXP-ANON security of the KEM scheme with reduction $\mathcal{R}_{\text{KEM,EXP-ANON}}$.

$$|\Pr[G_6^b \Rightarrow 1] - \Pr[G_5^b \Rightarrow 1]| \leq \text{Adv}_{\text{KEM}}^{\text{EXP-ANON}}(\mathcal{R}_{\text{KEM,EXP-ANON}}).$$

In the end, all parts of the challenge ciphertext are random and thereby both worlds are indistinguishable for the adversary.

$$\Pr[G_6^0 \Rightarrow 1] = \Pr[G_6^1 \Rightarrow 1]$$

Absorbing the two b -dependent reductions into single reductions $\mathcal{R}_{\text{KEM}}$ and $\mathcal{R}_{\text{F}}$ gives the final bound of the theorem. $\square$

E.3 Proof of Theorem 9

Theorem 9. *The AKEM construction $\text{AK}_{\text{sa-sa}}$ from Figure 9 offers authenticity. More precisely, for every adversary $\mathcal{A}$ against $\text{AK}_{\text{sa-sa}}$ in game AUTH from Figure 5, there exist two reductions $\mathcal{R}_{\text{F},1}$ as well as $\mathcal{R}_{\text{F},3}$ against the IND security of the PRF F , a reduction $\mathcal{R}_{\text{F},2}$ against the IND security of the dual PRG scheme modeled as a PRF, and a reduction $\mathcal{R}_{\text{SIG}}$ against the SUF security of the underlying signature scheme SIG, such that*

$$\text{Adv}_{\text{AK}_{\text{sa-sa}}}^{\text{AUTH}}(\mathcal{A}) \leq \frac{q_{e+d}^2}{|\mathcal{N}|} + \text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{R}_{\text{SIG}}) + q_{psk} \left(\sum_{i=1}^3 \text{Adv}_{\text{F}}^{\text{IND}}(\mathcal{R}_{\text{F},i}) + \frac{q_d}{|\mathcal{T}|} \right),$$

with variables as in Theorem 7, tag space $\mathcal{T}$, and the running time of the reductions is about that of $\mathcal{A}$.

Game $G_1^{\{s,a\}}; G_2^{\{s,a\}}; G_3^s - G_7^s; G_8^s$ 00 $(n, m) \leftarrow 0^2; sk, pk, psk \leftarrow \perp^3$ 01 $CH, XS, XA, N \leftarrow \emptyset^4$ 02 $H \leftarrow \perp$ 03 Invoke $\mathcal{A}$ 04 Stop with 0 Oracle Enc(s, i, pk, ad) 05 Require $(s, i, pk) \in ([n] \times [m] \times \mathcal{PK}) \cup \{\perp^3\}$ 06 Require $a \in \text{AU} \iff s \in [n]$ 07 Require $s \in \text{CF} \cup \text{AU} \iff i \in [m]$ 08 Require $a \in \text{CF} \iff pk \in \mathcal{PK}$ 09 $(\_, sk_{\text{SIG}}, h) \leftarrow sk_s; (pk_{\text{KEM}}, \_) \leftarrow pk$ 10 $(k_{\text{KEM}}, ck_{\text{KEM}}) \leftarrow \text{KEM.enc}(pk_{\text{KEM}})$ 11 $n \leftarrow \text{\$ } N$ 12 If $n \in N$: Abort 13 $N \leftarrow \{n\}$ 14 $k_{\text{tag}} \leftarrow F(psk_i, \text{tag})$ 15 $c_T \leftarrow \text{\$ } T.\text{tag}(k_{\text{tag}})$ 16 $k_{\text{otp1}} \leftarrow F(k_{\text{KEM}}, \text{otp1})$ 17 $\sigma \leftarrow \text{SIG.sig}(sk_{\text{SIG}}, (ck_{\text{KEM}}, n, c_T, h, ad))$ 18 $k_1 \leftarrow F(k_{\text{KEM}}, \text{mix}, ck_{\text{KEM}}, r, n)$ 19 $k_2 \leftarrow F(psk_i, \text{mix})$ 20 If $i = q_k^*$: 21 $k_2 \leftarrow \text{\$ } H[psk_i, \text{mix}] ? \mathcal{K}$ 22 $H[psk_i, \text{mix}] \leftarrow k_2$ 23 $k_{\text{mix}} \leftarrow G_d(k_1, k_2)$ 24 If $i = q_k^*$: 25 $k_{\text{mix}} \leftarrow \text{\$ } H[k_2, k_1] ? \mathcal{K}$ 26 $H[k_2, k_1] \leftarrow k_{\text{mix}}$ 27 $\tau \leftarrow F(k_{\text{mix}}, (ck_{\text{KEM}}, n, c_T, h, \sigma, ad))$ 28 $k_{\text{otp2}} \leftarrow F(k_{\text{mix}}, \text{otp2})$ 29 $c \leftarrow (ck_{\text{KEM}}, n, c_T \oplus k_{\text{otp1}}, (h, \sigma, \tau) \oplus k_{\text{otp2}})$ 30 $k \leftarrow F(k_{\text{mix}}, (\text{key}, c, ad))$ 31 If $i \in [m] \vee \exists r \in [n] : pk_r = pk$: 32 $C \leftarrow \{(s, r, i, ad, c)\}$ 33 Return (k, c) Oracle Gen(sk) 34 Require $\nexists u \in [n] : pk_u = \mu(sk)$ 35 $n \leftarrow n + 1$ 36 If $sk \in \text{SK}$: 37 $(sk_n, pk_n) \leftarrow (sk, \mu(sk)); XA \leftarrow \{n\}$ 38 Else: 39 $(sk_{\text{KEM}}, pk_{\text{KEM}}) \leftarrow \text{\$ } \text{KEM.gen}$ 40 $(sk_{\text{SIG}}, pk_{\text{SIG}}) \leftarrow \text{\$ } \text{SIG.gen}$ 41 $pk_n \leftarrow (pk_{\text{KEM}}, pk_{\text{SIG}})$ 42 $sk_n \leftarrow (sk_{\text{KEM}}, sk_{\text{SIG}}, H(pk_n))$ 43 Return pk_n Oracle CorruptA(u) 44 Require $u \in [n]$ 45 $XA \leftarrow \{u\}$ 46 Return sk_u	Oracle Dec(r, PK, I, ad, c) 47 Require $(r, PK, I) \in ([n] \times \mathcal{P}(\mathcal{PK}) \times \mathcal{P}([m])) \cup \{\perp^3\}$ 48 Require $a \in \text{AU} \iff PK \subseteq \mathcal{PK}$ 49 Require $s \in \text{CF} \cup \text{AU} \iff I \subseteq [m]$ 50 $K' \leftarrow \emptyset; P'[\cdot] \leftarrow \perp$ 51 For all $psk \in (psk_i)_{i \in I}$: 52 $k_{\text{tag}} \leftarrow F(psk_i, \text{tag}); K' \leftarrow \{k_{\text{tag}}\}; P'[k_{\text{tag}}] \leftarrow psk$ 53 $st' \leftarrow \text{\$ } T.\text{prec}(K')$ 54 $(sk_{\text{KEM}}, \_, \_) \leftarrow sk_r; (\_, pk_{\text{SIG}}) \leftarrow pk_r$ 55 $(ck_{\text{KEM}}, n, c', c'') \leftarrow c$ 56 $N \leftarrow \{n\}$ 57 $k_{\text{KEM}} \leftarrow \text{KEM.dec}(sk_{\text{KEM}}, ck_{\text{KEM}})$ 58 $k_{\text{otp1}} \leftarrow F(k_{\text{KEM}}, \text{otp1}, ck_{\text{KEM}}, r, n)$ 59 $c_T \leftarrow c' \oplus k_{\text{otp1}}$ 60 $k_T \leftarrow T.\text{det}(st', c_T)$ 61 $psk \leftarrow P'[k_T]$ 62 Require $psk \neq \perp$ 63 If $\exists i' \in I : psk_{i'} = psk : i \leftarrow i'$ 64 Else: $i \leftarrow \perp$ 65 $k_1 \leftarrow F(k_{\text{KEM}}, \text{mix}, ck_{\text{KEM}}, r, n)$ 66 $k_2 \leftarrow F(psk, \text{mix})$ 67 If $i = q_k^*$: 68 $k_2 \leftarrow \text{\$ } H[psk, \text{mix}] ? \mathcal{K}$ 69 $H[psk, \text{mix}] \leftarrow k_2$ 70 $k_{\text{mix}} \leftarrow G_d(k_1, k_2)$ 71 If $i = q_k^*$: 72 $k_{\text{mix}} \leftarrow \text{\$ } H[k_2, k_1] ? \mathcal{K}$ 73 $H[k_2, k_1] \leftarrow k_{\text{mix}}$ 74 $k_{\text{otp2}} \leftarrow F(k_{\text{mix}}, \text{otp2})$ 75 $(h, \sigma, \tau) \leftarrow c'' \oplus k_{\text{otp2}}$ 76 $pk \leftarrow \{pk' \in PK \mid H(pk') = h\}$ 77 If $\exists s' \in [n] : pk_{s'} = pk : s \leftarrow s'$ 78 Else: $s \leftarrow \perp$ 79 $b \leftarrow \text{SIG.vfy}(pk_{\text{SIG}}, (ck_{\text{KEM}}, n, c_T, h, ad), \sigma)$ 80 If $i = q_k^*$: 81 $\tau' \leftarrow \text{\$ } H[\text{mac2}, n, c_T, h, \sigma, ad] ? \mathcal{K}$ 82 $H[\text{mac2}, n, c_T, h, \sigma, ad] \leftarrow \tau$ 83 $\tau' \leftarrow F(k_{\text{mix}}, (ck_{\text{KEM}}, n, c_T, h, \sigma, ad))$ 84 If $b = \text{fail} \vee \tau \neq \tau' \vee pk = \emptyset$: Return $\perp^3$ 85 If $b \wedge (s, r, i, ad, c) \notin C \wedge s \notin XA$: 86 $\vee \tau' = \tau \wedge (s, r, i, ad, c) \notin C \wedge i \notin XS$: 87 If $b \wedge (s, r, i, ad, c) \notin C \wedge s \notin XA$: Abort 88 If $i \neq q_k^*$: Abort 89 If $\tau' = \tau \wedge (s, r, i, ad, c) \notin C \wedge i \notin XS$: Abort 90 $k \leftarrow F(k_{\text{mix}}, (\text{key}, c, ad))$ 91 If $k \neq \perp \wedge (s, r, i, ad, c) \notin C \wedge (\perp \neq s \notin XA \vee i \notin XS)$: 92 Stop with 1 93 Return (pk, i, k) Oracle CorruptS(i) 93 Require $i \in [m]$ 94 $XS \leftarrow \{i\}$ 95 Return psk_i Oracle Psk(psk) 96 Require $\nexists i \in XS : psk_i = psk$ 97 $m \leftarrow m + 1$ 98 If $psk \in \text{PSK}$: 99 $psk_m \leftarrow psk$ 100 $XS \leftarrow \{m\}$ 101 Else: $psk_m \leftarrow \text{\$ } \text{PSK}$ 102 Return
---	---

Fig. 23. Games for the proof of Theorem 9 for AUTH security of AKEM construction from Figure 9 with $\text{CF} = \{s, a\}$; $\text{AU} = \{s, a\}$. We highlight the symmetric case in teal and the asymmetric case in violet.

Proof. We prove that the fully redundant AKEM construction (Figure 9) satisfies AUTH security from Figure 5 through a sequence of game hops. Figure 23 provides an overview of these games. According to the AUTH definition, at least one of the two requirements must be satisfied: For each forgery attempt, the pre-shared key is uncorrupted, or the signing key is uncorrupted. That means, for each forgery attempt, it holds that the adversary has to break IND security of the PRF or SUF security of the signature scheme. We denote the games as $G_1^{\{s,a\}}$ and $G_2^{\{s,a\}}$ for both games initial

to the case distinction, as G_3^a for the game in the asymmetric case, and as $G_3^s - G_7^s$ for the games in the symmetric case.

Game G_1 is the original $\text{AUTH}_{\text{AK}_{\text{sa-sa}}}^{\{s,a\},\{s,a\}}(\mathcal{A})$ game instantiated with our construction from Figure 9. In particular,

$$\Pr[G_1^{\{s,a\}} \Rightarrow 1] = \Pr[\text{AUTH}_{\text{AK}_{\text{sa-sa}}}^{\{s,a\},\{s,a\}}(\mathcal{A}) \Rightarrow 1]$$

and

$$\text{Adv}_{\text{AK}_{\text{sa-sa}}}^{\text{AUTH}_{\text{sa-sa}}}(\mathcal{A}) = \Pr[G_1^{\{s,a\}} \Rightarrow 1].$$

Game G_2 is identical to G_1 , except that it aborts if a nonce is repeated across encapsulation and decapsulation queries.

G_1 to G_2 : A nonce is sampled uniformly at random in each encapsulation query. Moreover, the adversary can guess with each decapsulation query a new nonce. The probability of nonce collisions is thus bounded by the birthday bound over q_{e+d} encapsulation and decapsulation queries, and the nonce space $\mathcal{N}$. Since G_1 and G_2 are identical until nonce collision event E_n occurs, $\Pr[E_n]$ bounds the gap between the games by

$$\left| \Pr[G_1^{\{s,a\}} \Rightarrow 1] - \Pr[G_2^{\{s,a\}} \Rightarrow 1] \right| \leq \frac{(q_{e+d})^2}{|\mathcal{N}|}.$$

Game G_3 considers the asymmetric case. G_3 aborts if the adversary successfully forges a valid signature for an uncorrupted signing key sk_s (with $s \notin XA$). We call this event E_f .

G_2 to G_3 : We construct a reduction $\mathcal{R}_{\text{SIG}}$ against the multi-instance SUF_{SIG} security of the signature scheme (cf. Figure 12), which internally runs $\mathcal{A}$ and simulates $\text{AUTH}_{\text{AK}_{\text{s-sa}}}^{\{s,a\},\{s,a\}}$. We show that if $\mathcal{A}$ triggers E_f , reduction $\mathcal{R}_{\text{SIG}}$ can output a valid forgery. The simulation is as in Figure 20, handling corrupted queries locally and with the following specifications:

- On query from $\mathcal{A}$ to Gen, if $sk = \perp$, reduction $\mathcal{R}_{\text{SIG}}$ queries $pk_{\text{SIG}} \leftarrow \text{SIG.Gen}$. Similar to earlier proofs, we assume the reduction handles index mapping internally, and we omit the specification from Figure 20.
- On query from $\mathcal{A}$ to Enc, reduction $\mathcal{R}_{\text{SIG}}$ simulates the construction locally but obtains $\sigma \leftarrow \text{SIG.Sig}(s, (n, c_T, h, ad))$ for honest senders $s \notin XA$.
- On query from $\mathcal{A}$ to Dec, reduction $\mathcal{R}_{\text{SIG}}$ simulates according to the specification of the game. If E_f is triggered for some honest query (i.e., $s \notin XA$), the reduction $\mathcal{R}_{\text{SIG}}$ aborts the simulation.
- On query from $\mathcal{A}$ to CorruptA(u), reduction $\mathcal{R}_{\text{SIG}}$ forwards the query to $\text{SIG.Corrupt}(u)$ and returns the full secret key.

Eventually, $\mathcal{A}$ triggers E_f , and the reduction aborts the simulation. If so, we have $(s, r, i, ad, c) \notin C$ with $s \notin XA$ and a signature σ , which verifies for some $M = (n, c_T, h, ad)$. Since (s, r, i, ad, c) is fresh, (M, σ) must be fresh. As a consequence, the reduction can win its game by querying its SIG.Vfy oracle on $(s, (n, c_T, h, ad))$. This leads to two cases for the success analysis of $\mathcal{R}_{\text{SIG}}$:

- If M was never signed for s , this is a standard forgery.
- If M was signed, then for fixed n and psk_i , k_{otp} is fixed. Since c' in the forgery must differ from the encapsulated c'_{enc} , the unmasked values $(h, \sigma, \tau) = c' \oplus k_{\text{otp}}$ must differ. If σ or h changed, it is a strong forgery.

Consequently, this bound the gap between the games, and we have

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \text{Adv}_{\text{SIG}}^{\text{SUF}}(\mathcal{R}_{\text{SIG}}).$$

Game G_4 is the first game of the symmetric case, in which we assume the first forgery in the game is against an uncorrupted pre-shared key $psk_{q_k^*}$. In this game, we guess the index q_k^* to isolate the specific pre-shared key instance used for the eventual forgery attempt, and abort if our guess is wrong. This guessing step is necessary to reduce the security to a single-instance PRF challenger while allowing the adversary to adaptively corrupt any other pre-shared key instances.

G_3 to G_4 : We guess the index of $q_k^* \in [q_{psk}]$ where q_{psk} is the number of pre-shared key generation queries. We abort if the forgery attempt does not correspond to $psk_{q_k^*}$; note that because we prevented this forgery attempt to break the signature scheme, if we arrive here, the forgery must be against an uncorrupted pre-shared key. Since the choice of q_k^* is independent of the adversary's view, we lose a factor of q_{psk} in the advantage, which bounds the gap between the games as

$$\Pr[G_3 \Rightarrow 1] \leq q_{psk} \cdot \Pr[G_4 \Rightarrow 1].$$

Game G_5 replaces the output of k_2 from $F(psk_{q_k^*}, \text{mac})$ with uniformly random sampled values for the uncorrupted pre-shared key $psk_{q_k^*}$.

G_4 to G_5 : We construct a reduction $\mathcal{R}_{F,1}$ against the multi-instance IND security of the PRF scheme (cf. Figure 11). The reduction simulates the game for $\mathcal{A}$ by querying its PRF challenge oracle for the query by $\mathcal{A}$ on the guessed pre-shared key index q_k^* . Formally, the simulation is as in Figure 20 with the following specifications:

- In the q_k^* -th pre-shared key generation query, instantiate a new PRF challenger through $F.\text{Gen}$.
- On query to Enc and Dec , reduction $\mathcal{R}_{F,1}$ simulates as in the original construction from Figure 4 except that all PRF invocations with $psk_{q_k^*}$ are forwarded to the PRF challenger. The encapsulation and decapsulation computations for k_2 are forwarded to $F.\text{Chall}$ (observe that we enforced that the corresponding pre-shared key is never corrupted), and all other computations (i.e., the ones for k_{tag} and k) are forwarded to $F.\text{Eval}$.
- Per assumption, $psk_{q_k^*}$ is uncorrupted and thus there is no query $\text{CorruptS}(q_k^*)$.

If the PRF challenger computes the real PRF, the simulation is as in G_4 ; otherwise, as in G_5 . Finally, $\mathcal{R}_{F,1}$ outputs 1 if $\mathcal{A}$ triggers a successful forgery (the winning condition of the AUTH game) and 0 otherwise. The resulting advantage is then

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_F).$$

Game G_6 replaces the output of k_{mix} from $G_d(k_1, k_2)$ with uniformly random sampled values for the uncorrupted pre-shared key $psk_{q_k^*}$. At this point, we know that k_2 is sampled uniformly at random and we can outsource this computation to a PRF challenger. We thus model G_d as a F with IND security.

G_5 to G_6 : We construct a reduction $\mathcal{R}_{F,2}$ against the multi-instance IND security of the PRF scheme (cf. Figure 11). The reduction simulates the game for $\mathcal{A}$ by querying its PRF challenge oracle for the query by $\mathcal{A}$ on the guessed pre-shared key index q_k^* . Formally, the simulation is as in Figure 20 with the following specifications:

- In the q_k^* -th pre-shared key generation query, instantiate a new PRF challenger through $F.\text{Gen}$.
- On query to Enc and Dec , reduction $\mathcal{R}_{F,2}$ simulates as in the original construction from Figure 4 except that all PRF invocations with $psk_{q_k^*}$ are forwarded to the PRF challenger. The decapsulation computations for k_{mix} are forwarded to $F.\text{Chall}$, and all other computations (i.e., the ones for k_{tag} and k) are forwarded to $F.\text{Eval}$.
- Per assumption, $psk_{q_k^*}$ is uncorrupted and thus there is no query $\text{CorruptS}(p)$.

If the PRF challenger computes the real PRF, the simulation is as in G_5 ; otherwise, as in G_6 . Finally, $\mathcal{R}_{F,2}$ outputs 1 if $\mathcal{A}$ triggers a successful forgery (the winning condition of the AUTH game) and 0 otherwise. The resulting advantage is then

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1]| \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_F).$$

Game G_7 replaces the output τ from $F(k_{\text{mix}}, (n, c_T, h, \sigma, ad))$ with a uniform random sampled value.

G_6 to G_7 : This game hop is similar to the previous hop from G_4 to G_5 . We construct a reduction $\mathcal{R}_{F,3}$ against the multi-instance IND security of the PRF scheme (cf. Figure 11). The reduction simulates the game for $\mathcal{A}$ by using its PRF challenger to evaluate the PRF keyed by the random value k_{mix} belonging to the guessed instance q_k^* . Formally, the simulation is as in Figure 20 with the following specifications:

- In the q_k^* -th pre-shared key generation query, instantiate a new PRF challenger through $F.\text{Gen}$.

- On query to Enc and Dec, $\mathcal{R}_{F,2}$ simulates as in the original construction from Figure 4 except that the computation for the tag τ is obtained by forwarding (n, c_T, h, σ, ad) to the challenger’s F.Chall oracle. All other values (including k_{otp} and k_2) are handled locally as in G_6^s .
- Since k_{mix} is an internal value derived from the uncorrupted $psk_{q_k^*}$ and was replaced by a uniform random value in G_6 , it remains unknown to the adversary, satisfying the security requirements of the PRF challenger. Consequently, there is no query from $\mathcal{A}$ to $\text{CorruptS}(q_k^*)$.

If the PRF challenger computes the real PRF, the simulation is as in G_6 ; otherwise, as in G_7 . Finally, $\mathcal{R}_{F,3}$ outputs 1 if $\mathcal{A}$ triggers a successful forgery (the winning condition of the AUTH game) and 0 otherwise. The resulting advantage is then

$$|\Pr[G_6 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1]| \leq \text{Adv}_F^{\text{IND}}(\mathcal{R}_{F,2}).$$

Game G_8 is the final game, which aborts if the adversary creates a valid forgery where $\tau = \tau'$ under an uncorrupted pre-shared key.

G_7 to G_8 : To bound the advantage of detecting this change, consider the following. The forgery attempt is a new ciphertext, which means that the tuple of ciphertext components must be different from those of all existing encapsulations. Observe that partial ciphertext replays would count as a forgery attempt. The argument now is that, in order to produce a forgery, the adversary needs to be able to achieve $\tau = \tau'$ in the decapsulation, which happens with negligible probability.

First, k_{otp} is a random value (via the previous games), and since we excluded nonce collisions, the adversary never sees two ciphertexts that use the same one-time pad. That means, if the adversary adapts the ciphertext c' , the resulting τ is a random value. Second, if the adversary adapts the other components of the ciphertext, the resulting τ' is a random value. The probability that both values collide for one query is bounded in the size of the tag space $\mathcal{T}$. Summing over all q_d decapsulation queries, we bound the gap between the games by

$$|\Pr[G_6 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1]| \leq \frac{q_d}{|\mathcal{T}|}.$$

Finally, in G_8 the adversary cannot win, since we abort if the adversary creates asymmetric forgery and if the adversary creates a symmetric forgery. As a result, in G_8 we excluded all possibilities for the adversary to win and the adversary’s success probability is zero, i.e., we have

$$\Pr[G_8 \Rightarrow 1] = 0.$$

Since for each forgery attempt, at least one of the asymmetric and symmetric keys needs to be uncorrupted, we arrive at the bound for the theorem. Specifically, we bound the advantage of $\mathcal{A}$ by the sum of the gaps between the games. This concludes the proof. $\square$

F Tagging Equivalences

Some of our AKEM constructions of Section 5 make use of pre-shared keys to achieve confidentiality and/or authenticity. To improve the practical efficiency of determining the used pre-shared key from a set of candidates we rely on tagging mechanisms. In this section we argue the latter task must in fact inherently rely on some form of tagging. More precisely, we consider variants of MACs and symmetric encryption, the basic primitives for confidentiality and authenticity, generalized to a multi-key verification/decryption setting, where the recipient of a MAC/ciphertext holds several candidate keys and besides the main task of the primitive aims to efficiently determine the appropriate key.

We then show that, unsurprisingly, the respective ‘baseline’ primitive combined with a tagging mechanism yields a multi-key variant, that requires a single tag detection and verification/decryption operation and preserves anonymity. Interestingly, in the opposite direction, each multi-key primitive can be used to construct a tagging mechanism such that tag detection requires a single verification/decryption operation. This, in our eyes illustrates that tagging is indeed the fundamental problem at core of efficient key-detection for anonymous communication and further justifies its study as an object of independent cryptographic interest.

F.1 MAC with Multi-Key Verification

We begin with defining and building Multi-Key MACs.

Syntax. A Message Authentication Code with Multi-Key Verification (MK-MAC) is a tuple of algorithms $M = (M.\text{tag}, M.\text{agr}, M.\text{vfy})$ with key space $\mathcal{K}$, message space $\mathcal{M}$, tag space $\mathcal{T}$, and aggregated key space $\mathcal{AK}$ such that

- $M.\text{tag} : \mathcal{K} \times \mathcal{M} \rightarrow_{\S} \mathcal{T}$ computes a MAC tag for message $m \in \mathcal{M}$.
- $M.\text{agr} : \mathcal{P}(\mathcal{K}) \rightarrow \mathcal{AK}$ aggregates a set of keys.
- $M.\text{vfy} : \mathcal{AK} \times \mathcal{M} \times \mathcal{T} \rightarrow (\mathcal{K} \cup \{\perp\}) \times \{0, 1\}$ checks whether the input message-tag pair verifies for one of the aggregated keys and, if so, outputs this key.

For standard MACs, we simply assume that algorithm $M.\text{vfy}$ outputs the input key upon successful verification. Alternatively, we could define a weaker syntax for MK-MACs by not requiring that a successful verification outputs the specific verification key. In fact, also for this weaker syntax, we can show that MK-MACs imply Tagging Mechanisms with a logarithmic computation overhead (see Appendix F.1). Yet, we do not consider this weakened notion of MK-MACs useful (enough) in practice.

Correctness. For correctness, we require that the verification of an MK-MAC accepts the message-tag pair if it would accept it for one of the keys from the aggregated key set; in case of acceptance, also the suitable verification key is output. Formally, we capture this in game CORR in Figure 24 and we call an MK-MAC M correct if $\Pr[\text{CORR}_M(\mathcal{A}) = 1] = 0$ for every adversary $\mathcal{A}$.

Game $\text{CORR}_M(\mathcal{A})$	Oracle $\text{Vfy}(I, m, \tau)$
00 $n \leftarrow 0$	11 Require $I \subseteq [n]$
01 Invoke $\mathcal{A}$	12 $K \leftarrow \emptyset$
02 Stop with 0	13 For all $i \in I$: $K \stackrel{\cup}{\leftarrow} \{k_i\}$
	14 $K^* \leftarrow_{\S} M.\text{agr}(K)$
Oracle Gen	15 $(k, b) \leftarrow M.\text{vfy}(K^*, m, \tau)$
03 $n \leftarrow n + 1$	16 If $\exists i \in I : (m, \tau) \in MT_i$
04 $MT_n \leftarrow \emptyset$	$\wedge (k_i \neq k \vee b = \text{fal})$:
05 $k_n \leftarrow_{\S} \mathcal{K}$	17 Stop with 1
06 Return	18 If $\exists i' \in I : k = k_{i'} : i \leftarrow i'$
	19 Else: $i \leftarrow \perp$
Oracle Tag (i, m)	20 Return (i, b)
07 Require $i \in [n]$	
08 $\tau \leftarrow_{\S} M.\text{tag}(k_i, m)$	
09 $MT_i \stackrel{\cup}{\leftarrow} \{(m, \tau)\}$	
10 Return t	

Fig. 24. Correctness of Multi-Key MAC M .

Security: Unforgeability For security, we extend the standard strong unforgeability notion by allowing verification with aggregated key sets. This is formally captured in game SUF in Figure 25 and we define the advantage in winning this game against an MK-MAC M for adversary $\mathcal{A}$ as

$$\text{Adv}_M^{\text{SUF}}(\mathcal{A}) := \Pr[\text{SUF}_M(\mathcal{A}) \Rightarrow 1].$$

Security: Indistinguishability. We furthermore require that the MAC tags look pseudorandom. Building an efficient MK-MAC without this property can be done trivially: The MAC tag could just contain a fixed plain label that refers to the used MAC key. (E.g., let k be the MAC key, then (k', l) are derived from k via a pseudorandom generator; k' is actually used for computing the MAC tag for message m and label l is attached to the final MAC tag. At aggregation, all keys k' are stored with their indexing label l such that, at verification, finding the right key for a tag-label pair can be done in constant time.)

For this reason, we specify game IND in Figure 26 and we define the advantage in winning this game against an MK-MAC M for adversary $\mathcal{A}$ as

$$\text{Adv}_M^{\text{IND}}(\mathcal{A}) := |\Pr[\text{IND}_M^0(\mathcal{A}) \Rightarrow 1] - \Pr[\text{IND}_M^1(\mathcal{A}) \Rightarrow 1]|.$$

Game $\text{SUF}_M(\mathcal{A})$	Oracle $\text{Tag}(i, m)$
00 $n \leftarrow 0; X \leftarrow \emptyset$	10 Require $i \in [n]$
01 Invoke $\mathcal{A}$	11 $\tau \leftarrow_{\$} \text{M.tag}(k_i, m)$
02 Stop with 0	12 $MT_i \leftarrow MT_i \cup \{(m, \tau)\}$
	13 Return τ
Oracle Gen	
03 $n \leftarrow n + 1$	Oracle $\text{Vfy}(I, K, m, \tau)$
04 $MT_n \leftarrow \emptyset$	14 Require $I \subseteq [n] \wedge K \subseteq \mathcal{K}$
05 $k_n \leftarrow_{\$} \mathcal{K}$	15 For all $i \in I: K' \stackrel{\cup}{\leftarrow} \{k_i\}$
06 Return	16 $K^* \leftarrow \text{M.agr}(K \cup K')$
Oracle $\text{Corrupt}(i)$	17 $(k, b) \leftarrow \text{M.vfy}(K^*, m, \tau)$
07 Require $i \in [n]$	18 If $b \wedge K = I \cap X = \emptyset$
08 $X \stackrel{\cup}{\leftarrow} \{i\}$	$\wedge \nexists i \in I: k_i = k \wedge (m, \tau) \in MT_i:$
09 Return k_i	19 Stop with 1
	20 If $\exists i' \in I: k = k_{i'}: i \leftarrow i'$
	21 Else: $i \leftarrow \perp$
	22 Return (i, b)

Fig. 25. Game SUF for Multi-Key MAC M.

Game $\text{IND}_M^b(\mathcal{A})$	Oracle $\text{Chall}(i, m)$
00 $n \leftarrow 0; X \leftarrow \emptyset$	13 Require $i \in [n] \setminus X$
01 $b' \leftarrow_{\$} \mathcal{A}$	14 $\tau_0 \leftarrow_{\$} \text{M.tag}(k_i, m)$
02 Stop with b'	15 $\tau_1 \leftarrow_{\$} \mathcal{T}$
	16 $T_i \stackrel{\cup}{\leftarrow} \{(\tau_0, \tau_b)\}$
Oracle Gen	17 Return τ_b
03 $n \leftarrow n + 1$	Oracle $\text{Vfy}(I, K, m, \tau)$
04 $T_n \leftarrow \emptyset$	18 Require $I \subseteq [n] \wedge K \subseteq \mathcal{K}$
05 $k_n \leftarrow_{\$} \mathcal{K}$	19 For all $i \in I: K \stackrel{\cup}{\leftarrow} \{k_i\}$
06 Return	20 $K^* \leftarrow_{\$} \text{M.agr}(K)$
Oracle $\text{Corrupt}(i)$	21 If $\exists i \in I, \tau^*: (\tau^*, \tau) \in T_i:$
07 Require $i \in [n] \wedge T_i = \emptyset$	22 $\tau \leftarrow \tau^*$
08 $X \stackrel{\cup}{\leftarrow} \{i\}$	23 $(k, \beta) \leftarrow \text{M.vfy}(K^*, m, \tau)$
09 Return k_i	24 If $\exists i' \in I: k = k_{i'}: i \leftarrow i'$
Oracle $\text{Tag}(i, m)$	25 Else: $i \leftarrow \perp$
10 Require $i \in [n]$	26 Return (i, β)
11 $\tau \leftarrow_{\$} \text{M.tag}(k_i, m)$	
12 Return τ	

Fig. 26. Game IND for Multi-Key MAC M.

Construction of MAC. For proving equivalence, we start with building an MK-MAC from an ordinary MAC scheme with random tags and a Tagging Mechanism in Figure 27. The idea is that the final MAC tag $\tau = (t, \tau')$ consists of one tag of each underlying primitive: The Tagging Mechanism tags the key in t and the MAC tags the input message m together with the Tagging Mechanism's tag t in τ' . Thereby, the ordinary MAC scheme provides unforgeability for both components; both the Tagging Mechanism and the ordinary MAC scheme offer indistinguishability of their respective component; and the Tagging Mechanism detects the right key at multi-key verification such that a single actual MAC verification suffices.

Note that each algorithm of the MK-MAC only executes one algorithm per underlying primitive; in particular, the key aggregation of the MK-MAC only executes the Tagging Mechanism's pre-computation (plus some further efficient operations), and the tag verification only executes the detection and tag verification once. Thus, if the two building blocks are efficient, the MK-MAC construction is efficient, too.

Lemma 4. *The MK-MAC construction from Figure 27 offers unforgeability and indistinguishability, and its computational costs are bounded by the sum of the computational costs of the underlying Tagging Mechanism and MAC. More precisely, for every adversary $\mathcal{A}$ against MK-MAC M from Figure 27 in games SUF and IND from figures 25 and 26, there exist adversaries $\mathcal{B}_{M'}^{\text{SUF}}, \mathcal{B}_{M'}^{\text{IND}}$, and $\mathcal{B}_T$ in games SUF, IND, and IND from figures 25, 26, and 6 against the underlying MAC M' and the Tagging Mechanism T, respectively, such that*

$$\begin{aligned} \text{Adv}_M^{\text{SUF}}(\mathcal{A}) &\leq \text{Adv}_{M'}^{\text{SUF}}(\mathcal{B}_{M'}^{\text{SUF}}) \\ \text{Adv}_M^{\text{IND}}(\mathcal{A}) &\leq \text{Adv}_{M'}^{\text{SUF}}(\mathcal{B}_{M'}^{\text{SUF}}) + \text{Adv}_{M'}^{\text{IND}}(\mathcal{B}_{M'}^{\text{IND}}) + \text{Adv}_T^{\text{IND}}(\mathcal{B}_T) \end{aligned}$$

Proc M.tag(k, m)	Proc M.agr(K)
00 $(k', k_T) \leftarrow k$	10 $K[\cdot] \leftarrow \perp; K' \leftarrow \emptyset$
01 $t \leftarrow_{\$} \text{T.tag}(k_T)$	11 For all $k \in K$:
02 $\tau \leftarrow_{\$} \text{M.tag}'(k', (m, t))$	12 $(k', k_T) \leftarrow k$
03 Return (t, τ)	13 $K[k_T] \leftarrow k'$
Proc M.vfy($K^*, m, (t, \tau)$)	14 $K' \leftarrow \bigcup \{k_T\}$
04 $(st, K) \leftarrow K^*$	15 $st \leftarrow \text{T.prec}(K')$
05 $k_T \leftarrow \text{T.det}(st, t)$	16 $K^* \leftarrow (st, K)$
06 $k' \leftarrow K[k_T]$	17 Return K^*
07 $b \leftarrow \text{M.vfy}'(k', (m, t), \tau)$	
08 If b : Return (k', b)	
09 Return $(\perp, b)$	

Fig. 27. Multi-Key MAC M based on Tagging Mechanism T and ordinary MAC $M' = (\text{M.tag}', \text{M.vfy}')$.

where the running time of $\mathcal{B}_{M'}^{\text{SUF}}$, $\mathcal{B}_{M'}^{\text{IND}}$, and $\mathcal{B}_T$ is about that of $\mathcal{A}$.

Proof (Strong Unforgeability). The reduction from strong unforgeability of MK-MAC M to the ordinary MAC's strong unforgeability is straight forward: All algorithm executions of ordinary MAC M' in game SUF_M for construction M are simulated via queries to the corresponding oracles of underlying game $\text{SUF}_{M'}$; the Tagging Mechanism's algorithms are honestly and directly executed by the reduction itself. If the adversary in game SUF_M successfully forges a MAC tag τ for some tuple (m, t) and a set of key indices I , then the detection algorithm yields some k_T ; via key array K , created during key aggregation, k_T refers to a MAC instance that is simulated by the underlying game $\text{SUF}_{M'}$. Thus, if (m, t) is a forgery for index set I in oracle Vfy of game SUF_M , then it is also a forgery for the index referred from k_T in oracle Vfy of game $\text{SUF}_{M'}$; otherwise, (m, t) is rejected and, thereby, not a relevant forgery that needs to be reduces. Thus, an adversary winning game SUF_M yields a direct solution for winning game $\text{SUF}_{M'}$, which proves that

$$\text{Adv}_M^{\text{SUF}}(\mathcal{A}) \leq \text{Adv}_{M'}^{\text{SUF}}(\mathcal{B}_{M'}^{\text{SUF}}).$$

Proof (Indistinguishability). For proving indistinguishability of the MK-MAC's tags output from oracle Chall in game IND_M , we proceed in a sequence of games in which we start with rejecting tag forgeries that contain components of MK-MAC challenges, then we replace the Tagging Mechanism's tags with random elements, and then we do the same for the tags of ordinary MAC M' .

Game G_1 differs from the original game IND_M by rejecting every MK-MAC tag τ^* input to oracle Vfy that differs from but contains a component of an output from a query to oracle Chall. More concretely, let $\tau^* = (t, \tau)$ be an output of oracle Chall, then oracle Vfy rejects all inputs that either have the form $\tau' = (t, \tau''), \tau'' \neq \tau$ or $\tau' = (t'', \tau), t'' \neq t$, unless τ' was previously output by oracles Tag or Chall; note that τ^* is either a real or a random MK-MAC tag.

IND_M to G_1 : Reduction $\mathcal{R}$ simulates all algorithm executions of ordinary MAC M' via queries to the corresponding oracles of underlying game $\text{SUF}_{M'}$; the Tagging Mechanism's algorithms are honestly and directly executed by the reduction itself. Consider the case that the adversary in game IND_M , resp. G_1 , successfully produces an MK-MAC tag τ' of either of the two forms specified above, and this tag would be accepted in game IND_M , but it would be rejected in G_1 . Then $\tau' = (t'', \tau'')$ contains an ordinary MAC tag τ'' that has not been produced for Tagging Mechanism tag t'' in oracle Tag or Chall. Therefore, if τ' was accepted in IND_M , then τ'' is a valid MAC tag, which yields a forgery in game $\text{SUF}_{M'}$. If such a forgery is never provided, games IND_M and G_1 are identical; hence $\text{Adv}_M^{\text{ind}}(\mathcal{A}) \leq \text{Adv}_{G_1}(\mathcal{A}) + \text{Adv}_{M'}^{\text{SUF}}(\mathcal{B})$.

Game G_2 differs from G_1 by replacing all tags t computed by algorithm T.tag in algorithm M.tag in queries to oracle Chall by randomly sampled ones: $t \leftarrow_{\$} \mathcal{T}$.

G_1 to G_2 : Reduction $\mathcal{R}$ simulates all algorithm executions of Tagging Mechanism T in oracle queries to game G_1 , resp. G_2 , via oracle queries to underlying game IND_T . That is, for algorithm T.tag in algorithm M.tag in queries to oracles Tag and Chall, oracles Tag and Chall of game IND_T are queried, respectively; for algorithm T.prec in algorithm M.agr and algorithm T.det in algorithm M.vfy, both executed together in queries to oracle Vfy, oracle Det of game IND_T is queried. A distinguisher who identifies whether G_1 or G_2 was played yields bit b' for distinguishing whether oracle Chall in game IND_T outputs real or random tags, which successfully wins this game; hence $\text{Adv}_{G_1}(\mathcal{A}) \leq \text{Adv}_{G_2}(\mathcal{A}) + \text{Adv}_T^{\text{ind}}(\mathcal{B})$.

Game G_3 differs from G_2 by replacing all ordinary MAC tags τ computed by algorithm $M.\text{tag}'$ in algorithm $M.\text{tag}$ in queries to oracle Chall by randomly sampled ones: $\tau \leftarrow_{\$} \mathcal{T}$.

G_2 to G_3 : Reduction $\mathcal{R}$ simulates all algorithm executions of Tagging Mechanism T in oracle queries to game G_2 , resp. G_3 , via oracle queries to underlying game $\text{IND}_{M'}$. That is, for algorithm $M.\text{tag}'$ in algorithm $M.\text{tag}$ in queries to oracles Tag and Chall , oracles Tag and Chall of game $\text{IND}_{M'}$ are queried, respectively; for algorithm $M.\text{vfy}'$ in algorithm $M.\text{vfy}$, executed in queries to oracle Vfy , oracle Vfy of game $\text{IND}_{M'}$ is queried. A distinguisher who identifies whether G_2 or G_3 was played yields bit b' for distinguishing whether oracle Chall in game $\text{IND}_{M'}$ outputs real or random tags, which successfully wins this game; hence $\text{Adv}^{G_2}(\mathcal{A}) \leq \text{Adv}^{G_3}(\mathcal{A}) + \text{Adv}_{M'}^{\text{ind}}(\mathcal{B})$.

In G_3 , all outputs of oracle Chall are random elements of the respective tag spaces such that the adversary's perspective is independent of challenge bit b . Therefore, it holds that $\text{Adv}^{G_3}(\mathcal{A}) = 0$ and, furthermore:

$$\text{Adv}_M^{\text{IND}}(\mathcal{A}) \leq \text{Adv}_{M'}^{\text{SUF}}(\mathcal{B}_{M'}^{\text{SUF}}) + \text{Adv}_{M'}^{\text{IND}}(\mathcal{B}_{M'}^{\text{IND}}) + \text{Adv}_T^{\text{IND}}(\mathcal{B}_T).$$

Construction of Tagging Mechanism. The second step of proving equivalence is to build a Tagging Mechanism from an MK-MAC in Figure 28. Here, the idea is to tag the key by always computing a MAC tag for message $m = 1$. The Tagging Mechanism's pre-computation simply uses the MK-MAC's key aggregation, and the tag detection derives the output key from the MK-MAC tag verification.

Consequently, an efficient MK-MAC construction yields an equally efficient Tagging Mechanism.

Proc $T.\text{tag}(k)$	Proc $T.\text{prec}(K)$
00 $\tau \leftarrow_{\$} M.\text{tag}(k, 1)$	04 $st \leftarrow M.\text{agr}(K)$
01 Return τ	05 Return st
Proc $T.\text{det}(st, t)$	
02 $(k, b) \leftarrow M.\text{vfy}(st, 1, t)$	
03 Return k	

Fig. 28. Tagging Mechanism T based on Multi-Key MAC M .

Lemma 5. *The Tagging Mechanism from Figure 28 offers indistinguishability and its computational costs are bounded by the computational costs of the underlying MK-MAC. More precisely, for every adversary $\mathcal{A}$ against Tagging Mechanism T from Figure 28 in game IND from Figure 6, there exists an adversary $\mathcal{B}_M$ in game IND from Figure 26 against the underlying MK-MAC M such that*

$$\text{Adv}_T^{\text{IND}}(\mathcal{A}) \leq \text{Adv}_M^{\text{IND}}(\mathcal{B}_M)$$

where the running time of $\mathcal{B}_M$ is about that of $\mathcal{A}$.

Proof (Indistinguishability). The reduction from indistinguishability of Tagging Mechanism T to the indistinguishability of MK-MAC M is straight forward: All algorithm executions of MK-MAC M in game IND_T are simulated via queries to the corresponding oracles of underlying game IND_M .

That is, for algorithm $M.\text{tag}$ in algorithm $T.\text{tag}$ in queries to oracles Tag and Chall , oracles Tag and Chall of game IND_M are queried, respectively; for algorithm $M.\text{agr}$ in algorithm $T.\text{prec}$ and algorithm $M.\text{vfy}$ in algorithm $T.\text{det}$, both executed together in queries to oracle Det , oracle Vfy of game IND_M is queried. A distinguisher who guesses whether real or random Tagging Mechanism tags were output by oracle Chall of game IND_T yields bit b' for distinguishing whether oracle Chall in game IND_M outputs real or random tags, which successfully wins this game; hence

$$\text{Adv}_T^{\text{IND}}(\mathcal{A}) \leq \text{Adv}_M^{\text{IND}}(\mathcal{B}).$$

Construction of Without Key Output. If MK-MAC wasn't defined to also output the used key at verification, building a Tagging Mechanism from an MK-MAC is less trivial. Our Tagging Mechanism construction for this more advanced case is in Figure 28. Again, the idea is to tag the key by always computing a MAC tag for message $m = 1$. The Tagging Mechanism's pre-computation builds a binary tree in which all input keys are stored at leafs; each remaining node in the tree up to the root aggregates the two (aggregated) sets of keys of its two children. Upon detection, the MK-MAC verification is executed on a path from the root to a leaf node; at every step, the MAC tag is verified with both children's (aggregated) keys for message $m = 1$; the next step continues with the child for which this verification was successful.

As a result, the computation overheads of the Tagging Mechanism's pre-computation and tag detection are logarithmic in the MK-MAC's key aggregation and tag verification, respectively. Consequently, an efficient MK-MAC construction yields an almost equally efficient Tagging Mechanism.

Proc T.tag(k)	Proc T.prec(K)
00 $\tau \leftarrow_{\S} \text{M.tag}(k, 1)$	13 $st[\cdot] \leftarrow \perp$
01 Return τ	14 $i \leftarrow 0$
Proc T.det($(st, L), t$)	15 For all $k \in K$:
02 $i \leftarrow 0$	16 $st[0, i] \leftarrow (k, \diamond)$
03 For l from L to 1:	17 $i \leftarrow i + 1$
04 $(k_0, m_0) \leftarrow st[l, i]$	18 For l from 1 to $\log(K) - 1$:
05 $(k_1, m_1) \leftarrow st[l, i + 1]$	19 For i from 0 to $ K /2^l - 1$:
06 $d_0 \leftarrow \text{M.vfy}(k_0, 1, t)$	20 $(k_l, \cdot) \leftarrow st[l - 1, 2i]$
07 $d_1 \leftarrow \text{M.vfy}(k_1, 1, t)$	21 $(k_r, \cdot) \leftarrow st[l - 1, 2i + 1]$
08 If $\exists b \in \{0, 1\} : d_b = 1$:	22 $k_i \leftarrow \text{M.agr}(\{k_l, k_r\})$
09 If $m_b = \diamond$: Return k_b	23 $st[l, i] \leftarrow (k_i, \perp)$
10 $i \leftarrow 2(i + b)$	24 Return $(st, \log(K) - 1)$
11 Else: Return $\perp$	
12 Return $\perp$	

Fig. 29. Tagging Mechanism T based on Multi-Key MAC M.

Corollary 1. *The Tagging Mechanism from Figure 29 offers indistinguishability and its computational costs are bounded by an additional logarithmic factor of the computational costs of the underlying MK-MAC. More precisely, for every adversary $\mathcal{A}$ against Tagging Mechanism T from Figure 29 in game IND from Figure 6, there exists an adversary $\mathcal{B}_M$ in game IND from Figure 26 against the underlying MK-MAC M such that*

$$\text{Adv}_T^{\text{IND}}(\mathcal{A}) \leq \text{Adv}_M^{\text{IND}}(\mathcal{B}_M)$$

where the running time of $\mathcal{B}_M$ is about that of $\mathcal{A}$.

Adapting the proof of Lemma 5 only requires simulating multiple MAC instances per Tagging Mechanism.

F.2 Symmetric Encryption with Multi-Key Decryption

We prove a similar implication with different techniques from Symmetric Encryption that allows for multi-key decryption to Tagging Mechanisms.

Syntax. A Symmetric Encryption with Multi-Key Decryption (MK-SE) is a tuple of algorithms $E = (E.\text{enc}, E.\text{agr}, E.\text{dec})$ with key space $\mathcal{K}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and aggregated key space $\mathcal{AK}$ such that

- $E.\text{enc} : \mathcal{K} \times \mathcal{M} \rightarrow_{\S} \mathcal{C}$ encrypts message $m \in \mathcal{M}$.
- $E.\text{agr} : \mathcal{P}(\mathcal{K}) \rightarrow \mathcal{AK}$ aggregates a set of keys.
- $E.\text{dec} : \mathcal{AK} \times \mathcal{C} \rightarrow \mathcal{M}$ decrypts a ciphertext with one of the aggregated keys.

Game $\text{CORR}_E(\mathcal{A})$	Oracle $\text{Dec}(I, c)$
00 $n \leftarrow 0$	11 Require $I \subseteq [n]$
01 Invoke $\mathcal{A}$	12 $K \leftarrow \emptyset$
02 Stop with 0	13 For all $i \in I$: $K \leftarrow \{k_i\}$
Oracle Gen	14 $K^* \leftarrow_{\mathcal{S}} \text{E.agr}(K)$
03 $n \leftarrow n + 1$	15 $m \leftarrow \text{E.dec}(K^*, c)$
04 $M_n[\cdot] \leftarrow \perp$	16 If $\exists i \in I : M_i[c] \neq \perp \wedge M_i[c] = m$:
05 $k_n \leftarrow_{\mathcal{S}} \mathcal{K}$	17 Stop with 1
06 Return	18 Return m
Oracle $\text{Enc}(i, m)$	
07 Require $i \in [n]$	
08 $c \leftarrow_{\mathcal{S}} \text{E.enc}(k_i, m)$	
09 $M_i[c] \leftarrow m$	
10 Return t	

Fig. 30. Correctness of Multi-Key SE E.

Correctness. For correctness, we require that the message, decrypted from a ciphertext, equals the message encrypted to it, if the aggregated key set contains the key that was used at encryption. Formally, we capture this in game CORR in Figure 30 and we call an MK-SE E correct if $\Pr[\text{CORR}_E(\mathcal{A}) = 1] = 0$ for every adversary $\mathcal{A}$.

Security. The only security requirement for MK-SE is that ciphertexts look pseudorandom. As for MK-MACs, building an efficient MK-SE without this property can be done trivially: The ciphertext could just contain a fixed plain label that refers to the used encryption key. (E.g., let k be the SE key, then (k', l) are derived from k via a pseudorandom generator; k' is actually used for encryption and label l is attached to the resulting ciphertext. At aggregation, all keys k' are stored with their indexing label l such that, at decryption, finding the right key for a label can be done in constant time.)

For this reason, we specify game IND\$ in Figure 31 and we define the advantage in winning this game against an MK-SE E for adversary $\mathcal{A}$ as

$$\text{Adv}_E^{\text{IND\$}}(\mathcal{A}) := |\Pr[\text{IND\$}_E^0(\mathcal{A}) \Rightarrow 1] - \Pr[\text{IND\$}_E^1(\mathcal{A}) \Rightarrow 1]|.$$

Game $\text{IND\$}_E^b(\mathcal{A})$	Oracle $\text{Chall}(i, m)$
00 $n \leftarrow 0$; $X \leftarrow \emptyset$	13 Require $i \in [n] \setminus X$
01 $b' \leftarrow_{\mathcal{S}} \mathcal{A}$	14 $c_0 \leftarrow_{\mathcal{S}} \text{E.enc}(k_i, m)$
02 Stop with b'	15 $c_1 \leftarrow_{\mathcal{S}} \mathcal{C}$
Oracle Gen	16 $C_i \leftarrow \{(c_0, c_b)\}$
03 $n \leftarrow n + 1$	17 Return c_b
04 $C_n \leftarrow \emptyset$	Oracle $\text{Dec}(I, K, c)$
05 $k_n \leftarrow_{\mathcal{S}} \mathcal{K}$	18 Require $I \subseteq [n] \wedge K \subseteq \mathcal{K}$
06 Return	19 For all $i \in I$: $K \leftarrow \{k_i\}$
Oracle $\text{Corrupt}(i)$	20 $K^* \leftarrow_{\mathcal{S}} \text{E.agr}(K)$
07 Require $i \in [n] \wedge C_i = \emptyset$	21 If $\exists i \in I, c^* : (c^*, c) \in C_i$:
08 $X \leftarrow \{i\}$	22 $c \leftarrow c^*$
09 Return k_i	23 $m \leftarrow \text{E.dec}(K^*, c)$
Oracle $\text{Enc}(i, m)$	24 Return m
10 Require $i \in [n]$	
11 $c \leftarrow_{\mathcal{S}} \text{E.enc}(k_i, m)$	
12 Return c	

Fig. 31. Game IND\$ for Multi-Key SE E.

Construction of Tagging Mechanism. To build a Tagging Mechanism from an MK-SE, we simply encrypt a label that is fixed for the used key (i.e., derived with the key). At pre-computation, all key-label pairs are stored together such that, at detection, the label is decrypted, which then efficiently points to the right key.

Also for this construction, each algorithm of the Tagging Mechanism only executes one algorithm of the MK-SE; in particular, the pre-computation of the Tagging Mechanism only executes the MK-SE's key aggregation (plus some further efficient operations), and the tag detection only executes the decryption once. Thus, if the used MK-SE is efficient, the Tagging Mechanism is efficient, too.

Proc T.tag(k)	Proc T.prec(K)
00 $(k_E, n) \leftarrow k$	07 $KN[\cdot] \leftarrow \perp$
01 $t \leftarrow_{\$} \text{E.enc}(k_E, n)$	08 For all $k \in K$:
02 Return t	09 $(k_E, n) \leftarrow k$
	10 $KN[n] \leftarrow k$
Proc T.det(st, t)	11 $K \leftarrow \{k_E\}$
03 $(K^*, KN) \leftarrow st$	12 $K^* \leftarrow \text{E.agr}(K)$
04 $n \leftarrow \text{E.dec}(K^*, t)$	13 $st \leftarrow (K^*, KN)$
05 $k \leftarrow KN[n]$	14 Return st
06 Return k	

Fig. 32. Tagging Mechanism T based on Multi-Key Symmetric Encryption E.

Lemma 6. *The Tagging Mechanism from Figure 32 offers indistinguishability and its computational costs are bounded by the computational costs of the underlying MK-SE. More precisely, for every adversary $\mathcal{A}$ against Tagging Mechanism T from Figure 32 in game IND from Figure 6, there exists an adversary $\mathcal{B}_E$ in game IND from Figure 31 against the underlying MK-SE E such that*

$$\text{Adv}_T^{\text{IND}}(\mathcal{A}) \leq \text{Adv}_E^{\text{IND\$}}(\mathcal{B}_E)$$

where the running time of $\mathcal{B}_E$ is about that of $\mathcal{A}$.

Proof (Indistinguishability). The reduction from indistinguishability of Tagging Mechanism T to the indistinguishability of MK-SE E is straight forward: All algorithm executions of MK-SE E in game IND_T are simulated via queries to the corresponding oracles of underlying game $\text{IND\$}_E$.

That is, for algorithm E.enc in algorithm T.tag in queries to oracles Tag and Chall, oracles Enc and Chall of game $\text{IND\$}_E$ are queried, respectively; for algorithm E.agr in algorithm T.prec and algorithm E.dec in algorithm T.det, both executed together in queries to oracle Det, oracle Dec of game $\text{IND\$}_E$ is queried. A distinguisher who guesses whether real or random Tagging Mechanism tags were output by oracle Chall of game IND_T yields bit b' for distinguishing whether oracle Chall in game $\text{IND\$}_E$ outputs real or random ciphertexts, which successfully wins this game; hence

$$\text{Adv}_T^{\text{IND}}(\mathcal{A}) \leq \text{Adv}_E^{\text{IND\$}}(\mathcal{B}).$$

Construction of SE. The idea of building MK-SE construction from an ordinary SE and a Tagging Mechanism has almost been described in an earlier paragraph: the ciphertext of the underlying ordinary SE is attached with a label that helps the receiver to detect the right key for decryption; the only important difference to the prior sketched scheme is that, by using the Tagging Mechanism, the attached label is pseudorandom.

Since we capture Chosen Ciphertext Attacks in our model of $\text{IND\$}$ security, we would have to bind the Tagging Mechanism tag to the SE ciphertext; for this, we would need to replace the underlying SE scheme with Authenticated Encryption. We refrain from executing this step formally as it demonstrates only the obvious: Tagging Mechanisms can be used to implement multi-key decryption for Authenticated Encryption efficiently, where the latter is essentially just a simple variant of our $\text{AK}_{\text{s-sa}}$ construction from Section 5.2.