IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2020, No. 2, pp. 40–67. DOI:10.13154/tosc.v2020.i2.40-67

Key Assignment Schemes with Authenticated Encryption, revisited*

Jeroen Pijnenburg[1] and Bertram Poettering[2]

[1] Royal Holloway, University of London, Egham, United Kingdom
jeroen.pijnenburg.2017@rhul.ac.uk

[2] IBM Research – Zurich, Rüschlikon, Switzerland
poe@zurich.ibm.com

Abstract. A popular cryptographic option to implement Hierarchical Access Control in organizations is to combine a key assignment scheme with a symmetric encryption scheme. In brief, key assignment associates with each object in the hierarchy a unique symmetric key, and provides all higher-ranked "authorized" subjects with a method to recover it. This setup allows for encrypting the payloads associated with the objects so that they can be accessed by the authorized and remain inaccessible for the unauthorized. Both key assignment and symmetric encryption have been researched for roughly four decades now, and a plethora of efficient constructions have been the result. Surprisingly, a treatment of the joint primitive (key assignment combined with encryption, as used in practice) in the framework of provable security was conducted only very recently, leading to a publication in ToSC 2018(4). We first carefully revisit this publication. We then argue that there are actually two standard use cases for the combined primitive, which also require individual treatment. We correspondingly propose a fresh set of security models and provably secure constructions for each of them. Perhaps surprisingly, the two constructions call for different symmetric encryption primitives: While standard AEAD is the right tool for the one, we identify a less common tool called Encryptment as best fitting the other.

Keywords: Cryptographic Access Control · AEAD · Encryptment · Provable Security

1 Introduction

Access control is the protection of resources (objects) against access by unauthorized entities (users) [Shi07]. The set of access control rules is defined by an information flow policy (IFP). An IFP assigns each object a security label and each user a clearance level. The classical example is government documents that can be labelled from 'Top Secret' to 'Unclassified' and a civil servant would need a high enough security clearance to access these documents. As another example, consider a university building where everyone has access to the diligent student's office, professors are additionally granted access to their own offices, and security staff can access everyone's office. We note that the access rights of two professors are in general incomparable as neither can access all the offices the other has access to. The (partially) ordered set defined by the IFP can be represented as a hierarchy, hence we will refer to Hierarchical Access Control (HAC) in this paper.

A key assignment scheme (KAS) is a mechanism to cryptographically enforce an information flow policy, first proposed by Akl and Taylor [AT83]. With such a mechanism each clearance level is associated with some unique private information. A user can use

*The full version of this article is available at [PP20]

Received: 2019-11-23, Revised: 2020-03-02, Accepted: 2020-05-01, Published: 2020-07-24

their private information to derive symmetric encryption keys assigned to all objects lower in the hierarchy. These symmetric keys can subsequently be used to decrypt objects, ensuring that only authorized users, i.e. those who can derive the correct key, will be able to decrypt and access the object. In this article we will focus on the combined primitive of key assignment schemes and encryption, as it would be used in practice.

1.1 Prior Work

Akl and Taylor's work [AT83] on Key Assignment Schemes (KAS) for arbitrary partially ordered sets laid the foundations to implement cryptographic Hierarchical Access Control (HAC) in organizations. Earlier work by Gudes [Gud80] introduced a KAS for totally ordered sets and only offered a trivial solution for partially ordered sets: storing each (independent) key in the user's state. Since these works, many key assignment schemes have been proposed in the literature offering different time versus storage trade-offs [MTMA85, CC02, HL90, WC01, ADFM06, ADFM12, Tze06, AFB05, ABFF09, CT17, CFG+17]. A survey by Crampton *et al.* [CMW06] provides a categorization of KAS in five generic schemes. Analyzing many proposals, they note that most have been reinvented, differing only in the choice of cryptographic primitives to implement one of the five generic schemes. Furthermore, they conclude that many have made unsubstantiated claims and lack formal security analysis. Unsurprisingly, they remark many proposals have later been found to be flawed. For example they were vulnerable to attacks where two users collude to gain access to a security level neither of them has access to.

The lack of formal analysis was first addressed by Atallah *et al.* [AFB05] proposing two different security notions: Key Recovery (KR) security and Key Indistinguishability (KI) security. Informally, KR-security states that an adversary should not be able to recover a full key to which it should not have access. KI-security states that an adversary should not even be able to distinguish between the real key and a random key sampled from the same key space. After Atallah *et al.*'s work, several constructions have been proposed satisfying these security notions [DDFM09, DDFM10, DFM07, DFM11, FP11]. Freire *et al.* [FPP13] introduced the notion of Strong Key Indistinguishability (S-KI). This notion differs from the one provided in [AFB05] in the sense that the adversary is now also allowed to access keys used by users higher in the hierarchy (but not their secret state). The authors argue that a key may leak through its use, but this should not allow an adversary to derive information about other, unrelated keys. S-KI-security is also crucial to securely compose KAS with other cryptographic primitives, e.g. an encryption scheme. Castiglione *et al.* [CDM+16] proved KI-security and S-KI-security to be technically equivalent, albeit S-KI is more versatile.

A common construction technique for KAS, first considered in [CDM10], is by partitioning the IFP poset into a collection of totally ordered sets ('chains'), and solving the much easier problem of constructing KAS for chains. More recently, Crampton *et al.* have generalized this technique and realized KAS via tree partitions instead of chain partitions [CFG+15, CFG+17].

To the best of our knowledge there has not been attempted a formal treatment of the joint primitive of key assignment combined with encryption, in particular not in the domain of provable security, except for the recent work by Kandele and Paul [KP18a].

1.2 Motivation of this Work

Kandele and Paul (KP) assume that HAC should be implemented from KAS by KAS-deriving a key and using the latter with authenticated encryption (AE) [KP18a, p. 151]. Whilst this appears the natural way to realise the composition, the question arises which kind of authenticity is expected for the combined construction. Authentication issues might for instance emerge in the face of insider attacks, i.e. if users at higher hierarchy levels

manipulate ciphertexts of objects accessible by users at lower hierarchy levels. This topic was first formally approached by KP [KP18a] who study the HAC-promising joint primitive and whether it can be securely built from KAS+AE as described. They demonstrate, by presenting an attack, that the naive combination of KAS and AE is insecure [KP18a, p. 151], but claim security for a similar construction [KP18a, p. 151]. Unfortunately, as we point out, not only their attack cannot be formalized in their security notions, also the construction they propose as a fix turns out to fall prey to the same attack. We refer the reader to Sec. 2 for a further discussion. In this work, to remedy the situation, we fully re-think the security models of KAS+AE and develop provably secure constructions.

1.3 Contributions

We study the joint primitive of key assignment combined with encryption as it would be used in practice. KP [KP18a, KP18b] examined this primitive first and we reconsider their work, focusing particularly on their generic constructions. Our first contribution is that we identify two use cases of the joint primitive and observe that they require different security profiles. By consequence, we separate the notions into two independent primitives, the one allowing read-only access to authorized users and the other allowing read and write access. The read-only primitive guards against insider attacks as any modification by an authorized user would count as a forgery. On the other hand, the read-and-write primitive, which may be useful to organizations who wish to allow authorized employees to create and edit files, does, for obvious reasons, not protect against such attacks. KP only consider the read-only primitive.

Before we develop our notions for read-only and read-and-write access control, we first refine the definition of KAS in Sec. 4.2. In particular, we provide new security models allowing more interaction compared to the static models in previous work [FPP13]. Moreover, we introduce associated data to the KAS domain. The option to perform operations in the context of an explicitly specified associated-data string was proposed about two decades ago and since then has become standard in cryptography.[1] By offering a method to cleanly domain-separate inputs, e.g. for different applications, the availability of an associated-data input can considerably improve the versatility of a primitive. Finally, our KAS definition drops the necessity of *authentic* 'public information'. The conceptual separation of state and public information is typically made to improve storage efficiency, but it should not be assumed the public information is authentic if it is not stored in the user's state. Thus, we allow the adversary to provide forged public information.

As it turns out, surprisingly, the combined read-only and read-and-write primitives are considerably different. We thus develop two independent sets of strong and versatile security notions to analyze them. In Sec. 5 we define security for the read-only enforcement scheme and provide a provably secure construction. Next, in Sec. 6 we do the same for read-and-write enforcement. In both cases we focus on generic constructions (from KAS and some encryption primitive), allowing for modularity and ease of implementation. In particular, we do not commit to any specific KAS type as categorized by [CMW06], and allow for instance also the recent efficient construction from [CFG+17].

2 A Critique of [KP18a]

We carefully studied the models and schemes considered in [KP18a], and our verdict is that some of the arguments made in that work are questionable. In the following we walk

[1] Different names for the same concept are used in different domains. For instance 'associated data' for symmetric encryption [Rog02], 'tweak' for block ciphers [LRW02], and 'label' for public-key encryption [Sho04].

the reader through a line of issues that we found particularly worrisome from the security perspective. References in brackets refer to items in [KP18a].

The overall concern of [KP18a] is to marry key assignment with authenticated encryption (AE), in a sound way. The corresponding definition of AE is specified in [Sec. 2.2.6/ pp. 157–158], with formalizations of confidentiality and authenticity in two separate games, IND-PRV and INT, both made explicit in [Fig. 2/p. 158]. We note that the details of how the notions are formalized are non-standard, with severe implications on security that might not have been foreseen by the authors. Concretely, the IND-PRV notion models a kind of indistinguishability against passive adversaries (a.k.a. privacy), but with an encryption oracle missing. The latter means that, generically speaking, an instantiation that is IND-PRV secure according to the definition may become insecure the moment the adversary sees sample ciphertexts emerging from an application. Independently of this, the INT game requires that for adversaries with access to an encryption oracle it should be hard to find two different valid ciphertexts that share the same authentication tag. This crucially deviates from the standard understanding of integrity that rather considers the unforgeability of (whole) ciphertexts. Towards a separating example we found that an encryption scheme that uses a collision-resistant hash function to compute the tag from the ciphertext body meets INT-security according to [Fig. 2], yet is trivially forgeable in the classic (intuitive) sense. The two just described issues let us conclude that the AE related definitions in [KP18a] are not suitable for most AE applications.[2] To support this point of view also formally, in Appendix A.1 we specify an encryption scheme that is secure with respect to the IND-PRV and INT notions from [KP18a], yet allows for arbitrary ciphertext decryption and universal forgery attacks when operated (as an AE scheme) in the real world.

One might argue that demanding unorthodox security notions of primitives does not necessarily have to lead to issues—possibly the targeted application just doesn't require any stronger type of security. Without doubt, however, care has to be taken with instantiating the primitive, simply as off-the-shelf constructions might not have been tested with respect to the special goals. Indeed, in [Sec. 2.3.1/p. 162], KP explicitly propose a total of seven AE instantiations, some of them rather exotic, by referencing academic articles that specify such schemes. As these proposals are made without proofs of sufficiency, we checked all these references, just to confirm that not a single one of the articles tested for the non-standard INT definition of [KP18a]. We thus (have to) expect that all seven proposals are in fact insufficient to meet the IND-PRV and INT notions.[3]

We continue with discussing the confidentiality and authenticity notions of the combination of key assignment and AE. The definitions are in [Sec. 3/pp. 164–167]. Also here the authors define IND-PRV and INT notions, based on the games in [Fig. 6/p. 166]. Our observation on the IND-PRV game is that it does not consider insider attacks: If the adversary learns the state of any user, say at the bottom of an IFP hierarchy, the model would not require that the information of other users, including those higher in the hierarchy, remain confidential. This contradicts the core idea of access control. The INT game, surprisingly, represents the other extreme: Here the adversary *assigns* all secrets and public information (rather than just learning them), which reaches well beyond insider security. The restrictions that the game imposes on the adversary are actually so liberal that the correctness definition [p. 165] does not apply. One consequence of this is that different (authorized) users could decrypt the information associated with the same object differently, showing that a simplifying assumption on which the game crucially depends, namely that information is accessed exclusively by the 'owning' user, in general does not

[2] Another interpretation would be that the primitive considered by KP should not be referred to as AE. Indeed, the Encryptment primitive considered in [DGRW18] (see also Sec. 3.3) seems to be much closer in spirit to what KP describe.

[3] That is, in continuation of Footnote 2, even if KP actually meant to refer to encryptment schemes, they propose to instantiate them with (weaker) AE constructions.

hold. Thus, whatever behavioural regime the INT game is meant to enforce, it can be evaded by switching to an equivalent (authorized) user.

Our final set of remarks is on generic constructions that combine key assignment with authenticated encryption. In [Sec. 4.1/p. 167] and [Sec. 4.2/pp. 167–168] two such constructions are exposed. The first construction is canonic: To read the information stored for an object, the user first derives the corresponding symmetric key via the key assignment scheme, then uses the key with the AE scheme and a stored ciphertext. In [Sec. 4.1/p. 167] it is argued that this construction is insecure, and a corresponding attack is described. Unfortunately, while it is communicated that the attack is against the INT notion, the attack is not expressed in formal terms, and the specific adversarial actions seem to map neither to the INT nor the IND-PRV game. (For instance, the attacker shall "replace a ciphertext by a different ciphertext", but the games do not provide such an option.) The second construction is like the first one, but users store in their local information also the tags of all acceptable ciphertexts. The intuition seems to be that the INT notion of AE from [KP18a] (see above) prevents the adversary from finding a valid ciphertext that can replace an original one, explicitly ruling out the attack suggested for the first construction. A theorem statement in [p. 168] claims that if the AE scheme is INT secure, then the same holds for the combination of key assignment and AE. The proof sketch given is not very precise, and indeed we believe the statement is actually wrong. The crucial observation is that the INT notion for AE considers adversaries that 'only' have access to an encryption oracle, rather than to the encryption keys, while in the INT game for the key assignment plus AE combination the adversary controls, and thus knows, all keys. It is thus unclear how the one security notion can be leveraged to prove the other. To illustrate this further, in Appendix A.2 we specify an AE instance that provides IND-PRV and INT as per [Fig. 2/p. 158], yet allows trivial attacks against INT if the keys are known. We note that this AE scheme not only exemplifies that the theorem statement from [p. 168] is wrong, it also shows that the construction from [Sec. 4.2/pp. 167–168] falls prey to the same attack as suggested in [KP18a] against the construction from [Sec. 4.1/p. 167].

3 Preliminaries

3.1 Notation

For the Boolean constants True and False we either write `T` and `F`, respectively, or 1 and 0, respectively, depending on the context. For sets A, B we write B^A for the universe of functions $A \to B$. If the cardinality $|A|$ of A is small enough, computer implementations can represent such functions via tabulation. In this article, whenever an algorithm receives a function on input, or generates one as output, this should be understood using tabulation.

Unless explicitly communicated otherwise, all algorithms considered in this article may be randomized, i.e., are assumed to have access to a source of private random coins. We specify scheme algorithms and security games in pseudocode. In such code we write '$var \leftarrow exp$' for evaluating expression exp and assigning the result to variable var. Here, expression exp may comprise the invocation of algorithms.[4] If var is a set variable and exp evaluates to a set, we write $var \xleftarrow{\cup} exp$ shorthand for $var \leftarrow var \cup exp$. If S is a finite set, expression $\$(S)$ stands for picking an element of S uniformly at random; in particular, instruction $b \leftarrow \$(\{0,1\})$ flips a fair bit-valued coin and assigns the outcome to variable b. Associative arrays implement the 'dictionary' data structure: Once the instruction $A[\cdot] \leftarrow exp$ initialized all items of array A to the default value exp, with $A[idx] \leftarrow exp$ and $var \leftarrow A[idx]$ individual items indexed by expression idx can be updated or extracted.

[4] Non-deterministic algorithms are always executed with fresh uniform coins.

Security games are parameterized by an adversary, and consist of a main game body plus zero or more oracle specifications. The execution of a game starts with the main game body and terminates when a 'Stop with *exp*' instruction is reached, where the value of expression *exp* is taken as the outcome of the game. If the outcome of a game G is Boolean, we write $\Pr[\mathrm{G}(\mathcal{A})]$ for the probability that an execution of G with adversary $\mathcal{A}$ results in True (where the probability is taken over the random coins of G and $\mathcal{A}$). We define macros for specific combinations of game-ending instructions: We write 'Win' for 'Stop with `T`' and 'Lose' for 'Stop with `F`', and further 'Reward *cond*' for 'If *cond*: Win', 'Promise *cond*' for 'If ¬*cond*: Win', and 'Require *cond*' for 'If ¬*cond*: Lose'. (For an overview consider also Table 1 in Appendix B.) We use these macros to emphasize the specific semantics of game termination conditions. For instance, we terminate games with 'Reward *cond*' in cases where the adversary arranged for a situation —indicated by *cond* resolving to True— that should be awarded a win (e.g., the successful crafting of a forgery in an authenticity game).

We finally draw attention to a possibly unusual yet important detail of our algorithm and game notation that is connected with how algorithms handle failures. Here, by failure we understand the case where an algorithm does not generate output according to its syntax specification, but instead outputs some kind of error indicator.[5] In this article, for generality we assume that *any* scheme algorithm may fail. However, instead of encoding this explicitly in syntactical constraints which would heavily clutter the notation, we assume that if an algorithm invokes another algorithm as a subroutine, and the latter fails, then also the former immediately fails. We assume the same for game oracles: If an invoked scheme algorithm fails, then the oracle immediately aborts as well. The adversary is informed about the failure by receiving a special symbol, specifically $\perp$, as the result of the query.

We note that our approach to handle algorithm failures borrows from how modern programming languages handle 'exceptions', where any algorithm can raise (or 'throw') an exception, and if the caller does not explicitly 'catch' it, the caller is terminated as well and the exception is passed on to the next level.[6] We believe that our way to handle errors implicitly rather than explicitly contributes to obtaining definitions with clean and clear semantics.

3.2 AEAD

A scheme providing *authenticated encryption with associated data* (AEAD) for associated-data space $\mathcal{AD}$ and message space $\mathcal{M}$ consists of algorithms enc, dec, a key space $\mathcal{K}$, and a ciphertext space $\mathcal{C}$. The encryption algorithm enc takes a key $k \in \mathcal{K}$, an associated-data string $ad \in \mathcal{AD}$, and a message $m \in \mathcal{M}$, and returns a ciphertext $c \in \mathcal{C}$. The decryption algorithm dec takes a key $k \in \mathcal{K}$, an associated-data string $ad \in \mathcal{AD}$, and a ciphertext $c \in \mathcal{C}$, and returns a message $m \in \mathcal{M}$. A shortcut notation for this syntax is

$$\mathcal{K} \times \mathcal{AD} \times \mathcal{M} \to \mathrm{enc} \to \mathcal{C} \qquad \mathcal{K} \times \mathcal{AD} \times \mathcal{C} \to \mathrm{dec} \to \mathcal{M} \ .$$

Correctness and Security. We require of an AEAD scheme that if a message m is encrypted to a ciphertext c and then ciphertext c is (successfully) decrypted to a message m', and the involved associated-data strings ad are identical, then also the messages m, m'

[5] An example for this is an AEAD decryption algorithm that rejects a ciphertext that is too short to be valid, or one that is deemed unauthentic. (Other works in the field represent 'failing' with 'outputting $\perp$'.)

[6] See Wikipedia: `Exception_handling_syntax` for a first idea of exception handling syntaxes for many different programming languages. More concrete examples are documented in e.g. docs.python.org/3/tutorial/errors and docs.oracle.com/javase/tutorial/essential/exceptions.

Game SAFE($\mathcal{A}$)	**Oracle** Enc(ad, m)	**Oracle** Dec(ad, c)
00 $k \leftarrow \$(\mathcal{K})$	05 $c \leftarrow \text{enc}(k, ad, m)$	10 $m \leftarrow \text{dec}(k, ad, c)$
01 $\text{C}[\cdot] \leftarrow \emptyset$	06 Promise $c \notin \text{C}[ad]$	11 If $c \in \text{C}[ad]$:
02 $\text{M}[\cdot] \leftarrow \cdot$	07 $\text{C}[ad] \xleftarrow{\cup} \{c\}$	12 Promise $m = \text{M}[ad, c]$
03 Invoke $\mathcal{A}$	08 $\text{M}[ad, c] \leftarrow m$	13 $m \leftarrow \diamond$
04 Lose	09 Return c	14 Return m
Game INT($\mathcal{A}$)	**Oracle** Enc(ad, m)	**Oracle** Dec(ad, c)
15 $k \leftarrow \$(\mathcal{K})$	19 $c \leftarrow \text{enc}(k, ad, m)$	22 $m \leftarrow \text{dec}(k, ad, c)$
16 $\text{C}[\cdot] \leftarrow \emptyset$	20 $\text{C}[ad] \xleftarrow{\cup} \{c\}$	23 Reward $c \notin \text{C}[ad]$
17 Invoke $\mathcal{A}$	21 Return c	24 $m \leftarrow \diamond$
18 Lose		25 Return m
Game IND^b($\mathcal{A}$)	**Oracle** Enc(ad, m^0, m^1)	**Oracle** Dec(ad, c)
26 $k \leftarrow \$(\mathcal{K})$	30 Require $m^0 \equiv m^1$	34 $m \leftarrow \text{dec}(k, ad, c)$
27 $\text{C}[\cdot] \leftarrow \emptyset$	31 $c \leftarrow \text{enc}(k, ad, m^b)$	35 If $c \in \text{C}[ad]$:
28 $b' \leftarrow \mathcal{A}$	32 $\text{C}[ad] \xleftarrow{\cup} \{c\}$	36 $m \leftarrow \diamond$
29 Stop with b'	33 Return c	37 Return m

Figure 1: Games for AEAD. For the values ad, m, m^0, m^1, c provided by the adversary we require that $ad \in \mathcal{AD}$, $m, m^0, m^1 \in \mathcal{M}$, $c \in \mathcal{C}$. Read C like in ciphertext and M like in message. Assuming $\diamond \notin \mathcal{M}$, we encode suppressed messages with $\diamond$. We refer the reader to Appendix F.1 for a further discussion.

shall be identical. This is formalized via the SAFE game in Fig. 1.[7] Intuitively, the scheme is *safe* if the maximum advantage $\mathbf{Adv}^{\text{safe}}(\mathcal{A}) := \Pr[\text{SAFE}(\mathcal{A})]$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible. The scheme is perfectly safe if $\mathbf{Adv}^{\text{safe}}(\mathcal{A}) = 0$ for all $\mathcal{A}$.

Our security notions demand that the integrity of ciphertexts be protected (INT-CTXT), and that encryptions be indistinguishable in the presence of chosen-ciphertext attacks (IND-CCA). The notions are formalized via the INT and $\text{IND}^0, \text{IND}^1$ games in Fig. 1, the latter two with respect to some equivalence relation $\equiv \subseteq \mathcal{M} \times \mathcal{M}$ on the message space.[8] We say that a scheme provides *integrity* if the maximum advantage $\mathbf{Adv}^{\text{int}}(\mathcal{A}) := \Pr[\text{INT}(\mathcal{A})]$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible, and that it provides *indistinguishability* if the same holds for the advantage $\mathbf{Adv}^{\text{ind}}(\mathcal{A}) := |\Pr[\text{IND}^1(\mathcal{A})] - \Pr[\text{IND}^0(\mathcal{A})]|$.

3.3 Encryptment

The encryptment primitive, proposed by Dodis *et al.* in [DGRW18] in the context of secure messaging, provides one-time secure encryption with authenticity guarantees that hold beyond key compromise. In more detail, processing a message with an encryptment scheme yields a pair of ciphertext and *binding tag*, where the ciphertext hides the message

[7] We borrow the SAFETY notion, which should not be confused with a notion of security, from the Distributed Computing community. Informally, safety properties require that "bad things" will not happen. (In the case of encryption, it would be a bad thing if the decryption of an encryption would yield the wrong message.) Its counterpart LIVENESS is not relevant for modelling cryptographic properties of AEAD: The absence of liveness damages neither the integrity nor the confidentiality of a scheme. For an initial overview we refer to Wikipedia: `Safety_property` and Wikipedia: `Liveness`, and for the details to [AS87].

[8] We use relation $\equiv$ (in line 30 of IND^b) to deal with certain restrictions that practical AEAD schemes may feature. Concretely, most constructions we are aware of do not take effort to hide the length of encrypted messages, implying that indistinguishability is necessarily limited to same-length messages. In our formalization such a technical restriction can be expressed by defining $\equiv$ such that $m^0 \equiv m^1 :\Leftrightarrow |m^0| = |m^1|$.

contents as in regular encryption and the binding tag prevents forgery attacks even against insiders: A receiver equipped with an authentic copy of the binding tag will not accept any unauthentic ciphertext, even if all secrets of the sender and receiver become public. In Appendix C we reproduce details of a generic construction of this primitive from a passively secure secret key encryption scheme and a collision resistant hash function. More efficient though less general constructions are considered in [DGRW18]. Our formalization of encryptment follows the one of [DGRW18], but simplifies it by removing the option to process associated data, and by merging the decryption and verification algorithms into one.[9]

Definition 1. An *encryptment* scheme for message space $\mathcal{M}$ consists of algorithms enc, dec, a key space $\mathcal{K}$, a binding-tag space $\mathcal{B}t$, and a ciphertext space $\mathcal{C}$. The encryptment algorithm enc takes a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and returns a binding tag $bt \in \mathcal{B}t$ and a ciphertext $c \in \mathcal{C}$. The decryptment algorithm dec takes a key $k \in \mathcal{K}$, a binding tag $bt \in \mathcal{B}t$, and a ciphertext $c \in \mathcal{C}$, and returns a message $m \in \mathcal{M}$. A shortcut notation for this syntax is

$$\mathcal{K} \times \mathcal{M} \to \text{enc} \to \mathcal{B}t \times \mathcal{C} \qquad \mathcal{K} \times \mathcal{B}t \times \mathcal{C} \to \text{dec} \to \mathcal{M} \ .$$

Correctness and Security. We require of an encryptment scheme that if a message m is processed to a tag-ciphertext pair, and then a message m' is recovered from this pair, then the messages m, m' shall be identical. This is formalized via the SAFE game in Fig. 2. Intuitively, the scheme is *safe* if the maximum advantage $\mathbf{Adv}^{\text{safe}}(\mathcal{A}) := \max_{k\in\mathcal{K},m\in\mathcal{M}} \Pr[\text{SAFE}(k, m, \mathcal{A})]$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible. The scheme is perfectly safe if $\mathbf{Adv}^{\text{safe}}(\mathcal{A}) = 0$ for all $\mathcal{A}$.

Game SAFE$(k, m, \mathcal{A})$
00 $(bt, c) \leftarrow \text{enc}(k, m)$
01 $\mathcal{A}(k, m, bt, c)$
02 Lose

Oracle Dec$(\bar{c})$
03 $\bar{m} \leftarrow \text{dec}(k, bt, \bar{c})$
04 If $\bar{c} = c$:
05 Promise $\bar{m} = m$
06 $\bar{m} \leftarrow \diamond$
07 Return $\bar{m}$

Game INT$(k, m, \mathcal{A})$
08 $(bt, c) \leftarrow \text{enc}(k, m)$
09 $\mathcal{A}(k, m, bt, c)$
10 Lose

Oracle Dec$(\bar{c})$
11 $\bar{m} \leftarrow \text{dec}(k, bt, \bar{c})$
12 Reward $\bar{c} \neq c$
13 $\bar{m} \leftarrow \diamond$
14 Return $\bar{m}$

Game IND$^b(m^0, m^1, \mathcal{A})$
15 Require $m^0 \equiv m^1$
16 $k \leftarrow \$(\mathcal{K})$
17 $(bt, c) \leftarrow \text{enc}(k, m^b)$
18 $b' \leftarrow \mathcal{A}(m^0, m^1, bt, c)$
19 Stop with b'

Oracle Dec$(\bar{c})$
20 $\bar{m} \leftarrow \text{dec}(k, bt, \bar{c})$
21 If $\bar{c} = c$:
22 $\bar{m} \leftarrow \diamond$
23 Return $\bar{m}$

Figure 2: Games for encryptment. For the values $\bar{c}$ provided by the adversary we require that $\bar{c} \in \mathcal{C}$. Assuming $\diamond \notin \mathcal{M}$, we encode suppressed messages with $\diamond$. We refer the reader to Appendix F.2 for a further discussion.

Our security notions demand that the integrity of ciphertexts be protected (INT-CTXT), and that encryptions be indistinguishable in the presence of chosen-ciphertext attacks (IND-CCA). The notions are formalized via the INT and IND^0, IND^1 games in Fig. 2, where like in Sec. 3.2 the latter two depend on some equivalence relation $\equiv \subseteq \mathcal{M} \times \mathcal{M}$ on the message space. We say that a scheme provides *integrity* if the maximum advantage $\mathbf{Adv}^{\text{int}}(\mathcal{A}) := \max_{k\in\mathcal{K},m\in\mathcal{M}} \Pr[\text{INT}(k, m, \mathcal{A})]$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible, and that it provides *indistinguishability* if the same holds for the advantage $\mathbf{Adv}^{\text{ind}}(\mathcal{A}) := \max_{m^0,m^1\in\mathcal{M}} |\Pr[\text{IND}^1(m^0, m^1, \mathcal{A})] - \Pr[\text{IND}^0(m^0, m^1, \mathcal{A})]|$.

[9] While a considerable number of different security notions for encryptment is considered in [DGRW18], here we only reproduce those relevant for our work. They may appear in [DGRW18] under a different name.

4 Information Flow Policies and Key Assignment

We recall standard definitions from the domain of cryptographically enforced access control. While an information flow policy is an abstract structure that defines access rules, a key assignment scheme is a cryptographic primitive that helps implementing such a policy.

4.1 Information Flow Policies

An information flow policy for a hierarchical organization is a specification that describes which user can access which object. A key property is monotonicity in the sense that if an object is accessible by a specific user, then the object is also accessible by all higher-ranked users, where the ranking is defined via a *partially ordered set* (or *poset*), i.e., a set X equipped with a reflexive, anti-symmetric, transitive relation $\leqslant \subseteq X \times X$.[10] In this article we denote posets as a pair $(X, \leqslant)$, but we may also just write X if the relation is clear from the context. Our IFP definition follows [CMW06].

Definition 2. Let U and O be disjoint sets of *users* and *objects*, respectively. An *information flow policy* (**IFP**) for U, O is a tuple $(L, \leqslant, \nu, \omega)$ where

- $(L, \leqslant)$ is a (finite) partially ordered set of *security labels*;
- $\nu\colon U \to L$ and $\omega\colon O \to L$ are *security functions* that associate users and objects, respectively, with security labels.

We say that a user $u \in U$ is *authorized* to access an object $o \in O$ (e.g. for reading or writing) if $\omega(o) \leqslant \nu(u)$; otherwise, if $\omega(o) \not\leqslant \nu(u)$, the user is unauthorized to access the object.

For $u, v \in U$ and $o \in O$, as shortcut notations we also write $o \leqslant u$ if $\omega(o) \leqslant \nu(u)$, and $u \leqslant v$ if $\nu(u) \leqslant \nu(v)$. Note the transitivity $o \leqslant u \wedge u \leqslant v \Rightarrow o \leqslant v$. This further suggests to denote the sets of authorized and unauthorized users for an object $o \in O$ with $\{u : o \leqslant u\}$ and $\{u : o \not\leqslant u\}$, respectively, and to denote with $\{o : o \leqslant u\}$ the set of objects a user $u \in U$ is authorized for.[11] To avoid trivial side cases it is often useful to demand that for each object there is at least one authorized user, i.e., that $\{u : o \leqslant u\} \neq \emptyset$ for all $o \in O$. We refer with $\mathcal{IFP}_U^O$ to the space of all IFPs for U, O with this property.

4.2 Enforcement via Key Assignment

A classic option to efficiently implement an information flow policy is via cryptographic enforcement [AT83]. The idea is that all users in an organization are assigned individual secrets (also referred to as their secret states) that allow them to derive keys associated with the objects they are authorized to access. These keys protect the object payloads by means of some cryptographic primitive, e.g. symmetric encryption. This section focuses on the key assigning component, referred to as KAS. We specify its syntax and an appropriate security notion in the upcoming paragraphs, where the syntactical framework enriches the one from [CMW06] by the option to derive keys depending on an associated-data

[10] Recall that while the symbols $<$ and $\not\geqslant$ are equivalent in totally ordered sets, this may not be assumed in posets. More precisely, specific elements $x, x' \in X$ in a poset may be incomparable, meaning that the relations $x \neq x'$ and $x \not\leqslant x'$ and $x \not\geqslant x'$ hold simultaneously.

[11] In continuation of Footnote 10, the set of users *not* authorized for object o is in general *not* equal to $\{u : u < o\}$. This needs emphasis as [KP18a] seem to be using the terms interchangeably, which leads to artificially weak security definitions. For instance, we believe that in all games of [Fig. 3/p. 159] the set P_u should be defined as $\{S_v : u \not\leqslant v\}$ rather than $\{S_v : v < u\}$. Similar comments apply to the games in [Fig. 6/p. 166], and the running text on [Sec. 2.2.7/pp. 158–160] and [p. 166].

input, and the security notion strengthens the strongest definition of [FPP13] by tolerating potentially unauthentic public inputs.[12]

A large number of KAS constructions is proposed in prior work [CMW06, CFG+17][13]. Considering that these not necessarily support associated-data inputs, we show in Appendix D how a classic KAS, i.e., one that lacks support for associated data, can be transformed into a KAS of our type, with only a minimal overhead incurring due to an auxiliary PRF invocation.[14]

Definition 3. A *key assignment scheme* (**KAS**) for sets U, O, associated-data space $\mathcal{AD}$, and key space $\mathcal{K}$, consists of the two algorithms setup and derive, a secret-state space Σ, and a public-state space Π.[15] The initialization algorithm setup takes an information flow policy $I = (L, \leqslant, \nu, \omega) \in \mathcal{IFP}_U^O$ and outputs a mapping $\vec{\sigma}\colon U \to \Sigma$ that assigns to each user $u \in U$ a corresponding secret state $\vec{\sigma}(u)$, and a public state $\pi \in \Pi$ (shared by all users). We let $\sigma_u := \vec{\sigma}(u)$ for all $u \in U$. The key derivation algorithm derive takes on input a secret state $\sigma \in \Sigma$, a public state $\pi \in \Pi$, an object $o \in O$, and an associated-data string $ad \in \mathcal{AD}$, and outputs a key $k \in \mathcal{K}$. A shortcut notation for the algorithms' syntax is

$$\mathcal{IFP}_U^O \to \text{setup} \to \Sigma^U \times \Pi \qquad\qquad \Sigma \times \Pi \times O \times \mathcal{AD} \to \text{derive} \to \mathcal{K} \ .$$

Correctness and Security. We require of a KAS that if any two (authorized) users independently of each other derive the key associated with an object, and the involved associated-data strings are identical, then also the derived keys shall be identical. This is formalized via the SAFE game in Fig. 3. Intuitively, the scheme is *safe* if for all IFPs I the maximum advantage $\mathbf{Adv}^{\text{safe}}(I, \mathcal{A}) := \Pr[\text{SAFE}(I, \mathcal{A})]$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible. The scheme is perfectly safe if $\mathbf{Adv}^{\text{safe}}(I, \mathcal{A}) = 0$ for all $\mathcal{A}$.

Our security notion demands that the keys associated with objects be secret and uniformly distributed. The notion is formalized in a model supporting user corruptions via the real-or-random style $\text{KIND}_t^0, \text{KIND}_t^1$ games in Fig. 3, where $t \in \mathbb{N}$ is a parameter that specifies the maximum number of challengeable keys. We say that the scheme provides t-challenge *indistinguishable keys* if for all IFPs I the maximum advantage $\mathbf{Adv}^{t\text{-kind}}(I, \mathcal{A}) := |\Pr[\text{KIND}_t^1(I, \mathcal{A})] - \Pr[\text{KIND}_t^0(I, \mathcal{A})]|$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible.

We note that key indistinguishability definitions proposed in prior works, e.g. in [FPP13], assume that scheme algorithms always have authentic access to the public state. Our model is stronger by not making this assumption and letting the adversary provide forged public information. Fortunately, as we detail in Appendix D, a classic KAS (satisfying the notions of [FPP13]) can readily be transformed into a KAS that satisfies our notions.

The following result formally connects the t-challenge and single-challenge cases of key indistinguishability. The proof is based on a simple hybrid argument and provided in Appendix G.1.

Lemma 1. *Let I be an IFP and $\mathcal{A}$ an adversary. Then for any $t \in \mathbb{N}$ there exists an adversary $\mathcal{A}'$ such that $\mathbf{Adv}^{t\text{-kind}}(I, \mathcal{A}) \leq t \cdot \mathbf{Adv}^{1\text{-kind}}(I, \mathcal{A}')$.*

[12] Concretely, the notion defined by our KIND_1^b games implies the notion defined by the S-KI-ST game of [FPP13]. While S-KI-ST is less interactive than KIND_1^b, it is not hard to see that there are simple reductions between the two (for the case of one fixed associated-data string and authentic access to the public state). See Lemma 1 for the more general case.

[13] Many more works propose KAS constructions. Here we reference [CFG+17] for a recent example of an efficient construction and [CMW06] as it surveys general construction techniques. See Sec. 1.1 for further pointers.

[14] The reverse direction is, of course, trivial: To obtain a classic KAS from a KAS according to our definitions it suffices to restrict the associated-data space to a single element.

[15] The 'state' term should not suggest that states are dynamic objects. In the KAS context, states are assigned once and then remain invariant.

Game SAFE$(I, \mathcal{A})$
00 $\mathrm{K}[\cdot] \leftarrow \emptyset$
01 $(\vec{\sigma}, \pi) \leftarrow \mathrm{setup}(I)$
02 $\mathcal{A}(I, \pi)$
03 Lose

Oracle Derive$(u, \bar{\pi}, o, ad)$
04 $k \leftarrow \mathrm{derive}(\sigma_u, \bar{\pi}, o, ad)$
05 Promise $o \leqslant u$
06 If $\bar{\pi} = \pi$:
07 $\quad \mathrm{K}[o, ad] \xleftarrow{\cup} \{k\}$
08 $\quad$ Promise $|\mathrm{K}[o, ad]| \leq 1$
09 Return k

Oracle Corrupt(u)
10 Return σ_u

Game KIND$^b_t(I, \mathcal{A})$
11 $\mathrm{K}[\cdot] \leftarrow \times$
12 $\mathrm{CO} \leftarrow \emptyset$
13 $\mathrm{CH} \leftarrow \emptyset$
14 $(\vec{\sigma}, \pi) \leftarrow \mathrm{setup}(I)$
15 $b' \leftarrow \mathcal{A}(I, \pi)$
16 Stop with b'

Oracle Reveal(o, ad)
17 Require $\mathrm{K}[o, ad] \in \mathcal{K}$
18 $k \leftarrow \mathrm{K}[o, ad]$
19 $\mathrm{K}[o, ad] \leftarrow \diamond$
20 Return k

Oracle Corrupt(u)
21 $O_u \leftarrow \{o : o \leqslant u\}$
22 Require $\mathrm{CH} \cap O_u = \emptyset$
23 $\mathrm{CO} \xleftarrow{\cup} O_u$
24 Return σ_u

Oracle Derive$(u, \bar{\pi}, o, ad)$
25 $k \leftarrow \mathrm{derive}(\sigma_u, \bar{\pi}, o, ad)$
26 If $\bar{\pi} = \pi$:
27 $\quad$ If $\mathrm{K}[o, ad] = \times$:
28 $\quad\quad \mathrm{K}[o, ad] \leftarrow k$
29 $\quad k \leftarrow \star$
30 Return k

Oracle Challenge(o, ad)
31 Require $\mathrm{K}[o, ad] \in \mathcal{K}$
32 Require $o \notin \mathrm{CO}$
33 $k^0 \leftarrow \mathrm{K}[o, ad]$
34 $k^1 \leftarrow \$(\mathcal{K})$
35 $\mathrm{K}[o, ad] \leftarrow \diamond$
36 $\mathrm{CH} \xleftarrow{\cup} \{o\}$
37 Require $|\mathrm{CH}| \leq t$
38 Return k^b

Figure 3: Games for KAS. For all values $u, \bar{\pi}, o, ad$ provided by the adversary we require that $u \in U$, $\bar{\pi} \in \Pi$, $o \in O$, $ad \in \mathcal{AD}$. Read K like in key, CO like in corrupted object, and CH like in challenge. Assuming $\times, \star, \diamond \notin \mathcal{K}$, we encode uninitialized keys with $\times$, challengeable keys with $\star$, and revealed/challenged keys with $\diamond$. We refer the reader to Appendix F.3 for a further discussion.

5 Read-Only Enforcement

We first develop the syntax and security notions for read-only cryptographically enforced access control, and then provide a provably secure solution. In read-only enforcement, files or messages are specified at setup, and no modifications are allowed, not even by users that are authorized to (read-only) access them.

5.1 Syntax and Security

As in Sec. 4.1, let U be a set of users and O be a set of objects.

Definition 4. A *read-only enforcement scheme* (**ROES**) for sets U, O and message space $\mathcal{M}$ consists of the two algorithms setup, read, a secret-state space Σ, a public-state space Π, and a ciphertext space $\mathcal{C}$. The initialization algorithm setup takes an information flow policy $I = (L, \leqslant, \nu, \omega) \in \mathcal{IFP}^O_U$ and a mapping $\mathrm{M}\colon O \to \mathcal{M}$ (one message per object), and outputs a mapping $\vec{\sigma}\colon U \to \Sigma$ (one secret state per user), a public state $\pi \in \Pi$ (shared by all users), and a mapping $\mathrm{C}\colon O \to \mathcal{C}$ (one ciphertext per object). We let $\sigma_u := \vec{\sigma}(u)$ for all $u \in U$. The retrieve algorithm read takes on input a secret state $\sigma \in \Sigma$, a public state $\pi \in \Pi$, an object $o \in O$, and a ciphertext $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$. A shortcut notation for the algorithms' syntax is

$$\mathcal{IFP}^O_U \times \mathcal{M}^O \to \mathrm{setup} \to \Sigma^U \times \Pi \times \mathcal{C}^O \qquad \Sigma \times \Pi \times O \times \mathcal{C} \to \mathrm{read} \to \mathcal{M} \ .$$

Correctness. We require of a ROES that if a message m is specified at setup for an object and then a message m' is (successfully) retrieved for that object, then the retrieving user must be authorized and the messages m, m' identical. This is formalized via the SAFE game in Fig. 4. Intuitively, the scheme is *safe* if for all IFPs I the advantage

$\mathbf{Adv}^{\text{safe}}(I,\mathcal{A}) := \max_{\mathrm{M}} \Pr[\text{SAFE}(I,\mathrm{M},\mathcal{A})]$, where the maximum is over all object-to-message mappings $\mathrm{M} \in \mathcal{M}^O$, is negligible for all realistic adversaries $\mathcal{A}$. The scheme is perfectly safe if $\mathbf{Adv}^{\text{safe}}(I,\mathcal{A}) = 0$ for all $\mathcal{A}$.

Game SAFE$(I,\mathrm{M},\mathcal{A})$ 00 $(\vec{\sigma},\pi,\mathrm{C}) \leftarrow \text{setup}(I,\mathrm{M})$ 01 Invoke $\mathcal{A}(I,\mathrm{M},\pi,\mathrm{C})$ 02 Lose	**Oracle** Read$(u,\bar{\pi},o,c)$ 03 $m \leftarrow \text{read}(\sigma_u,\bar{\pi},o,c)$ 04 Promise $o \leqslant u$ 05 If $\bar{\pi} = \pi \wedge c = \mathrm{C}(o)$: 06 Promise $m = \mathrm{M}(o)$ 07 $m \leftarrow \diamond$ 08 Return m	**Oracle** Corrupt(u) 09 Return σ_u
Game INT$(I,\mathrm{M},\mathcal{A})$ 10 $(\vec{\sigma},\pi,\mathrm{C}) \leftarrow \text{setup}(I,\mathrm{M})$ 11 Invoke $\mathcal{A}(I,\mathrm{M},\pi,\mathrm{C})$ 12 Lose	**Oracle** Read$(u,\bar{\pi},o,c)$ 13 $m \leftarrow \text{read}(\sigma_u,\bar{\pi},o,c)$ 14 Reward $\bar{\pi} \neq \pi$ 15 Reward $c \neq \mathrm{C}(o)$ 16 $m \leftarrow \diamond$ 17 Return m	**Oracle** Corrupt(u) 18 Return σ_u
Game IND$^b(I,\mathrm{M}^0,\mathrm{M}^1,\mathcal{A})$ 19 For all $o \in O$: 20 Require $\mathrm{M}^0(o) \equiv \mathrm{M}^1(o)$ 21 $\mathrm{CH} \leftarrow \{o : \mathrm{M}^0(o) \neq \mathrm{M}^1(o)\}$ 22 $(\vec{\sigma},\pi,\mathrm{C}) \leftarrow \text{setup}(I,\mathrm{M}^b)$ 23 $b' \leftarrow \mathcal{A}(I,\mathrm{M}^0,\mathrm{M}^1,\pi,\mathrm{C})$ 24 Stop with b'	**Oracle** Read$(u,\bar{\pi},o,c)$ 25 $m \leftarrow \text{read}(\sigma_u,\bar{\pi},o,c)$ 26 If $\bar{\pi} = \pi \wedge c = \mathrm{C}(o)$: 27 $m \leftarrow \diamond$ 28 Return m	**Oracle** Corrupt(u) 29 $O_u \leftarrow \{o : o \leqslant u\}$ 30 Require $\mathrm{CH} \cap O_u = \emptyset$ 31 Return σ_u

Figure 4: Games for ROES. For all values $u,\bar{\pi},o,c$ provided by the adversary we require that $u \in U$, $\bar{\pi} \in \Pi$, $o \in O$, $c \in \mathcal{C}$. Read CH like in challenge. Assuming $\diamond \notin \mathcal{M}$, we encode suppressed messages with $\diamond$. We refer the reader to Appendix F.4 for a further discussion.

Security. Our security notions demand that the messages associated with objects remain authentic and confidential. The notions are formalized in models supporting user corruptions. Authenticity is defined via the INT game and confidentiality via the left-or-right style $\text{IND}^0, \text{IND}^1$ games in Fig. 4. The latter depend on some equivalence relation $\equiv \subseteq \mathcal{M} \times \mathcal{M}$ on the message space, like the IND^b games in Sections 3.2 and 3.3. We say that the scheme provides *integrity* if for all IFPs I the advantage $\mathbf{Adv}^{\text{int}}(I,\mathcal{A}) := \max_{\mathrm{M}} |\Pr[\text{INT}(I,\mathrm{M},\mathcal{A})]|$, where the maximum is over all object-to-message mappings $\mathrm{M} \in \mathcal{M}^O$, is negligible for all realistic adversaries $\mathcal{A}$. We say that the scheme provides *indistinguishability* if for all IFPs I the advantage $\mathbf{Adv}^{\text{ind}}(I,\mathcal{A}) := \max_{\mathrm{M}^0,\mathrm{M}^1} |\Pr[\text{IND}^1(I,\mathrm{M}^0,\mathrm{M}^1,\mathcal{A})] - \Pr[\text{IND}^0(I,\mathrm{M}^0,\mathrm{M}^1,\mathcal{A})]|$, where the maximum is over all object-to-message mappings $\mathrm{M}^0,\mathrm{M}^1 \in \mathcal{M}^O$, is negligible for all realistic adversaries $\mathcal{A}$.

5.2 Construction

In Fig. 5 we specify a construction of ROES that is secure according to our definitions. As generic building blocks we employ a KAS, an encryptment scheme, and a collision-resistant hash function.[16]

[16] We recall from [KP18a] that simply composing a KAS with a regular AEAD scheme does *not* yield a secure ROES. Indeed, in Fig. 8 we formally consider this construction (though in a different context; assume the associated-data input ad is fixed to some constant #) and the following attack shows that it falls short of providing authenticity. The attack succeeds by corrupting any user, recovering the AEAD key from their state, and forging by simply creating a fresh ciphertext using this key. In detail, consider the adversary $\mathcal{A}$ against the INT game that receives $(I,\mathrm{M},\pi,\mathrm{C})$ in line 11 (of Fig. 4), picks any user

We provide details of the construction. Numbers in brackets refer to line numbers in the figure. The setup procedure [00–14] initializes in [00,01] the two arrays B[·] and C[·] that will store for each object a binding tag and a ciphertext, respectively. It then runs the KAS initialization algorithm setup′ [02] to generate a secret-state vector, which assigns a secret state to each user, and a public state. Subsequently, it computes the hash value of the public state [03]. In a loop [04–10], for each object [04] the procedure picks an(y) authorized user [05,06] to derive a key for the object [07], uses this key and the specified message [08] with the encryptment algorithm enc′ [09], and stores the resulting binding tag and ciphertext in arrays B and C, respectively [10]. In a second loop [11–13], for each user [11] the procedure considers all objects the user is authorized for [12], and encodes the hash value of the public state, the KAS secret state and the binding tags of these objects in the user's ROES secret state [13]. The setup procedure returns such a secret state for each user, the KAS public state, and for each object the encryptment ciphertext [14]. Given this description, the details of the read procedure should be clear. Note that the procedure fails if any of the steps in [16,17,19] fail.

Proc setup(I, M)
00 $\mathrm{B}[\cdot] \leftarrow \times$
01 $\mathrm{C}[\cdot] \leftarrow \times$
02 $(\vec{\sigma}', \pi) \leftarrow \mathrm{setup}'(I)$
03 $h' \leftarrow H(\pi)$
04 For all $o \in O$:
05 $\quad U' \leftarrow \{u' : o \leqslant u'\}$
06 $\quad$ Pick any $u' \in U'$
07 $\quad k \leftarrow \mathrm{derive}'(\sigma'_{u'}, \pi, o, \#)$
08 $\quad m \leftarrow \mathrm{M}(o)$
09 $\quad (bt, c) \leftarrow \mathrm{enc}'(k, m)$
10 $\quad (\mathrm{B}[o], \mathrm{C}[o]) \leftarrow (bt, c)$
11 For all $u \in U$:
12 $\quad O' \leftarrow \{o' : o' \leqslant u\}$
13 $\quad \sigma_u \leftarrow (h', \sigma'_u, \mathrm{B}[O'])$
14 Return $(\vec{\sigma}, \pi, \mathrm{C})$

Proc read(σ_u, π, o, c)
15 $(h', \sigma'_u, \mathrm{B}[\cdot]) \leftarrow \sigma_u$
16 Require $H(\pi) = h'$
17 $k \leftarrow \mathrm{derive}'(\sigma'_u, \pi, o, \#)$
18 $bt \leftarrow \mathrm{B}[o]$
19 $m \leftarrow \mathrm{dec}'(k, bt, c)$
20 Return m

Figure 5: Our ROES construction roes with procedures setup, read using procedures setup′, derive′ of a generic KAS kas and procedures enc′, dec′ of a generic encryptment scheme enc. In [00,01] we encode uninitialized values with $\times$. In [07,17] we write # for any fixed associated-data string.

We conduct the security analysis of our scheme in Sec. 7.1. Here we just provide shortened versions of the formal statements.

Theorem 1 (informal version). Fix any IFP I. If the key assignment scheme kas and the encryptment scheme enc provide indistinguishability, then so does our ROES construction roes. More precisely, for any adversary $\mathcal{A}$ there exist adversaries $\mathcal{A}', \mathcal{A}''$ of comparable efficiency such that

$$\mathbf{Adv}^{\mathrm{ind}}_{\mathrm{roes}}(I, \mathcal{A}) \leq t \cdot \big(\mathbf{Adv}^{\text{1-kind}}_{\mathrm{kas}}(I, \mathcal{A}') + \mathbf{Adv}^{\mathrm{ind}}_{\mathrm{enc}}(\mathcal{A}'')\big),$$

where $t = |O|$ is the number of objects defined by the IFP, and the advantages are defined using the indistinguishability games corresponding to the primitive.

$u \in U$ and object $o \in \{o : o \leqslant u\}$ and message $m' \neq \mathrm{M}[o]$, queries Corrupt(u) to receive state σ_u, recovers key $k_u \leftarrow \mathrm{derive}'(\sigma_u, \pi, o, \#)$ as in line 07 of Fig. 8, computes $c' \leftarrow \mathrm{enc}'(k_u, ad, m')$ as in line 08 (of Fig. 8), and queries Read(u, π, o, c') to score a win by line 15 (of Fig. 4). For this adversary we have $\mathbf{Adv}^{\mathrm{int}}(I, \mathcal{A}) = 1$, for any IFP I.

Theorem 2 (informal version). Fix any IFP I. If the encryptment scheme enc provides integrity, then so does our ROES construction roes. More precisely, for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{A}'$ of comparable efficiency such that

$$\mathbf{Adv}^{\text{int}}_{\text{roes}}(I, \mathcal{A}) \leq t \cdot \mathbf{Adv}^{\text{int}}_{\text{enc}}(\mathcal{A}'),$$

where $t = |O|$ is the number of objects defined by the IFP, and the advantages are defined using the integrity game corresponding to the primitive.

6 Read-Write Enforcement

We first develop the syntax and security notions for read-write cryptographically enforced access control, and then provide a provably secure solution. In read-write enforcement, only the IFP has to be specified at setup, while the encryptions for each object happen dynamically.

6.1 Syntax and Security

As in Sec. 4.1, let U be a set of users and O be a set of objects.

Definition 5. A *read and write enforcement scheme* (**RWES**) for sets U, O, associated-data space $\mathcal{AD}$, and message space $\mathcal{M}$, consists of the three algorithms setup, enc, dec, a secret-state space Σ, a public-state space Π, and a ciphertext space $\mathcal{C}$. The initialization algorithm setup takes an information flow policy $I = (L, \leqslant, \nu, \omega) \in \mathcal{IFP}^O_U$ and outputs a mapping $\vec{\sigma}\colon U \to \Sigma$ (one secret state per user) and a public state $\pi \in \Pi$ (shared by all users). We let $\sigma_u := \vec{\sigma}(u)$ for all $u \in U$. The encryption algorithm enc takes on input a secret state $\sigma \in \Sigma$, a public state $\pi \in \Pi$, an object $o \in O$, an associated-data string $ad \in \mathcal{AD}$, and a message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$. The decryption algorithm dec takes on input a secret state $\sigma \in \Sigma$, a public state $\pi \in \Pi$, an object $o \in O$, an associated-data string $ad \in \mathcal{AD}$, and a ciphertext $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$. A shortcut notation for the algorithms' syntax is

$$\mathcal{IFP}^O_U \to \text{setup} \to \Sigma^U \times \Pi$$

$$\Sigma \times \Pi \times O \times \mathcal{AD} \times \mathcal{M} \to \text{enc} \to \mathcal{C} \qquad \Sigma \times \Pi \times O \times \mathcal{AD} \times \mathcal{C} \to \text{dec} \to \mathcal{M} \ .$$

Correctness. We require of a RWES that if a message m is (successfully) encrypted with respect to an object to a ciphertext c and then ciphertext c is (successfully) decrypted with respect to the same object to a message m', and the involved associated-data strings are identical, then the involved users must be authorized and the messages m, m' identical. This is formalized via the SAFE game in Fig. 6. Intuitively, the scheme is *safe* if for all IFPs I the maximum advantage $\mathbf{Adv}^{\text{safe}}(I, \mathcal{A}) := \Pr[\text{SAFE}(I, \mathcal{A})]$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible. The scheme is perfectly safe if $\mathbf{Adv}^{\text{safe}}(I, \mathcal{A}) = 0$ for all $\mathcal{A}$.

Security. Our security notions demand that the messages associated with objects remain authentic and confidential. The notions are formalized in models supporting user corruptions. Authenticity is defined via the game INT in Fig. 6 and confidentiality via the left-or-right style $\text{IND}^0, \text{IND}^1$ games in Fig. 7. The latter depend on a parameter $t \in \mathbb{N}$ that specifies the maximum number of challenge pairs, and, akin to Sec. 5.1, on some equivalence relation $\equiv \subseteq \mathcal{M} \times \mathcal{M}$ on the message space. We say that the scheme provides *integrity* if for all IFPs I the maximum advantage $\mathbf{Adv}^{\text{int}}(I, \mathcal{A}) := \Pr[\text{INT}(I, \mathcal{A})]$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible. We say that the scheme provides t-challenge *indistinguishability* if for all IFPs I the maximum advantage

Game SAFE$(I, \mathcal{A})$
00 $\mathrm{C}[\cdot] \leftarrow \emptyset$
01 $\mathrm{M}[\cdot] \leftarrow \cdot$
02 $(\vec{\sigma}, \pi) \leftarrow \mathrm{setup}(I)$
03 $\mathcal{A}(I, \pi)$
04 Lose

Oracle Corrupt(u)
05 Return σ_u

Oracle Enc$(u, \bar{\pi}, o, ad, m)$
06 $c \leftarrow \mathrm{enc}(\sigma_u, \bar{\pi}, o, ad, m)$
07 Promise $o \leqslant u$
08 If $\bar{\pi} = \pi$:
09 Promise $c \notin \mathrm{C}[o, ad]$
10 $\mathrm{C}[o, ad] \xleftarrow{\cup} \{c\}$
11 $\mathrm{M}[o, ad, c] \leftarrow m$
12 Return c

Oracle Dec$(u, \bar{\pi}, o, ad, c)$
13 $m \leftarrow \mathrm{dec}(\sigma_u, \bar{\pi}, o, ad, c)$
14 Promise $o \leqslant u$
15 If $\bar{\pi} = \pi \wedge c \in \mathrm{C}[o, ad]$:
16 Promise $m = \mathrm{M}[o, ad, c]$
17 $m \leftarrow \diamond$
18 Return m

Game INT$(I, \mathcal{A})$
19 $\mathrm{C}[\cdot] \leftarrow \emptyset$
20 $\mathrm{CO} \leftarrow \emptyset$
21 $(\vec{\sigma}, \pi) \leftarrow \mathrm{setup}(I)$
22 $\mathcal{A}(I, \pi)$
23 Lose

Oracle Corrupt(u)
24 $O_u \leftarrow \{o : o \leqslant u\}$
25 $\mathrm{CO} \xleftarrow{\cup} O_u$
26 Return σ_u

Oracle Enc$(u, \bar{\pi}, o, ad, m)$
27 $c \leftarrow \mathrm{enc}(\sigma_u, \bar{\pi}, o, ad, m)$
28 If $\bar{\pi} = \pi$:
29 $\mathrm{C}[o, ad] \xleftarrow{\cup} \{c\}$
30 Return c

Oracle Dec$(u, \bar{\pi}, o, ad, c)$
31 $m \leftarrow \mathrm{dec}(\sigma_u, \bar{\pi}, o, ad, c)$
32 If $o \notin \mathrm{CO}$:
33 Reward $\bar{\pi} \neq \pi$
34 Reward $c \notin \mathrm{C}[o, ad]$
35 $m \leftarrow \diamond$
36 Return m

Figure 6: SAFE and INT games for RWES (the IND^b games are in Fig. 7). For all values $u, \bar{\pi}, o, ad, m, c$ provided by the adversary we require that $u \in U$, $\bar{\pi} \in \Pi$, $o \in O$, $ad \in \mathcal{AD}$, $m \in \mathcal{M}$, $c \in \mathcal{C}$. Read C like in ciphertext, M like in message, and CO like in corrupted object. Assuming $\diamond \notin \mathcal{M}$, we encode suppressed messages with $\diamond$. We refer the reader to Appendix F.5 for a further discussion.

$\mathbf{Adv}^{t\text{-ind}}(I, \mathcal{A}) := |\Pr[\mathrm{IND}^1_t(I, \mathcal{A})] - \Pr[\mathrm{IND}^0_t(I, \mathcal{A})]|$ that can be attained by realistic adversaries $\mathcal{A}$ is negligible.

The following result formally connects the t-challenge and single-challenge cases of indistinguishability. The proof is based on a simple hybrid argument and provided in Appendix G.2.

Lemma 2. *Let I be an IFP and $\mathcal{A}$ an adversary. Then for any $t \in \mathbb{N}$ there exists an adversary $\mathcal{A}'$ such that $\mathbf{Adv}^{t\text{-ind}}(I, \mathcal{A}) \leq t \cdot \mathbf{Adv}^{1\text{-ind}}(I, \mathcal{A}')$.*

6.2 Construction

In Fig. 8 we specify a construction of RWES that is secure according to our definitions. As generic building blocks we employ a KAS, an AEAD scheme, and a collision-resistant hash function. Alternatively, in Appendix E we leverage the associated-data input of KAS and provide a construction for which an AE scheme suffices as building block instead of AEAD.

We provide details of the construction. Numbers in brackets refer to line numbers in the figure. The setup procedure simply runs setup$'$ from KAS [00] and appends the hash value of the public state to each secret state [01–03]. Finally, setup returns the secret-state vector, which assigns a secret state to each user, and a global public state [04]. The enc and dec procedures mirror each other, we will describe dec. The dec procedure first verifies that the provided public state is correct [11]. Next, it derives an object-dependent key with derive$'$ from KAS [12] (but independently of the associated data). Subsequently, it uses this key to decrypt the ciphertext with associated data using dec$'$ from AEAD [13], and returns the message [14]. Note that the procedure fails if any of the steps in [11,12,13] fail.

We conduct the security analysis of our scheme in Sec. 7.2. Here we just provide shortened versions of the formal statements.

Game $\mathrm{IND}^b_t(I,\mathcal{A})$
00 $\mathrm{C}[\cdot] \leftarrow \emptyset$
01 $\mathrm{CO} \leftarrow \emptyset$
02 $\mathrm{CH} \leftarrow \emptyset$
03 $(\vec{\sigma},\pi) \leftarrow \mathrm{setup}(I)$
04 $b' \leftarrow \mathcal{A}(I,\pi)$
05 Stop with b'

Oracle $\mathrm{Enc}(u,\bar{\pi},o,ad,m)$
06 $c \leftarrow \mathrm{enc}(\sigma_u,\bar{\pi},o,ad,m)$
07 If $\bar{\pi}=\pi$:
08 $\quad \mathrm{C}[o,ad] \xleftarrow{\cup} \{c\}$
09 Return c

Oracle $\mathrm{Challenge}(u,\bar{\pi},o,ad,m^0,m^1)$
10 Require $m^0 \equiv m^1$
11 $c \leftarrow \mathrm{enc}(\sigma_u,\bar{\pi},o,ad,m^b)$
12 If $\bar{\pi}=\pi$:
13 $\quad$ Require $o \notin \mathrm{CO}$
14 $\quad \mathrm{C}[o,ad] \xleftarrow{\cup} \{c\}$
15 $\quad \mathrm{CH} \xleftarrow{\cup} \{o\}$
16 $\quad$ Require $|\mathrm{CH}| \le t$
17 Return c

Oracle $\mathrm{Dec}(u,\bar{\pi},o,ad,c)$
18 $m \leftarrow \mathrm{dec}(\sigma_u,\bar{\pi},o,ad,c)$
19 If $\bar{\pi}=\pi \wedge c \in \mathrm{C}[o,ad]$:
20 $\quad m \leftarrow \diamond$
21 Return m

Oracle $\mathrm{Corrupt}(u)$
22 $O_u \leftarrow \{o : o \leqslant u\}$
23 Require $\mathrm{CH} \cap O_u = \emptyset$
24 $\mathrm{CO} \xleftarrow{\cup} O_u$
25 Return σ_u

Figure 7: IND^b_t games for RWES (the SAFE and INT games are in Fig. 6). For all values $u,\bar{\pi},o,ad,m,m^0,m^1,c$ provided by the adversary we require that $u \in U$, $\bar{\pi} \in \Pi$, $o \in O$, $ad \in \mathcal{AD}$, $m,m^0,m^1 \in \mathcal{M}$, $c \in \mathcal{C}$. Read C like in ciphertext, CO like in corrupted object, and CH like in challenge. Assuming $\diamond \notin \mathcal{M}$, we encode suppressed messages with $\diamond$. We refer the reader to Appendix F.5 for a further discussion.

Proc $\mathrm{setup}(I)$
00 $(\vec{\sigma}',\pi) \leftarrow \mathrm{setup}'(I)$
01 $h' \leftarrow H(\pi)$
02 For all $u \in U$:
03 $\quad \sigma_u \leftarrow (h',\sigma'_u)$
04 Return $(\vec{\sigma},\pi)$

Proc $\mathrm{enc}(\sigma_u,\pi,o,ad,m)$
05 $(h',\sigma'_u) \leftarrow \sigma_u$
06 Require $H(\pi)=h'$
07 $k \leftarrow \mathrm{derive}'(\sigma'_u,\pi,o,\#)$
08 $c \leftarrow \mathrm{enc}'(k,ad,m)$
09 Return c

Proc $\mathrm{dec}(\sigma_u,\pi,o,ad,c)$
10 $(h',\sigma'_u) \leftarrow \sigma_u$
11 Require $H(\pi)=h'$
12 $k \leftarrow \mathrm{derive}'(\sigma'_u,\pi,o,\#)$
13 $m \leftarrow \mathrm{dec}'(k,ad,c)$
14 Return m

Figure 8: Our RWES construction rwes with procedures $\mathrm{setup},\mathrm{enc},\mathrm{dec}$ using procedures $\mathrm{setup}',\mathrm{derive}'$ of a generic KAS kas and procedures $\mathrm{enc}',\mathrm{dec}'$ of a generic AEAD scheme aead. In [07,12] we write # for any fixed associated-data string. If the conditions in [06,11] are not fulfilled, the respective algorithm fails.

Theorem 3 (informal version). Fix any IFP I. If the key assignment scheme kas and the AEAD scheme aead provide indistinguishability, then so does our RWES construction rwes. More precisely, for any adversary $\mathcal{A}$ that queries at most q_o different objects in the t-challenge indistinguishability game, there exist adversaries $\mathcal{A}',\mathcal{A}''$ of comparable efficiency such that

$$\mathbf{Adv}^{t\text{-ind}}_{\mathrm{rwes}}(I,\mathcal{A}) \le t \cdot q_o \cdot \big(\mathbf{Adv}^{1\text{-kind}}_{\mathrm{kas}}(I,\mathcal{A}') + \mathbf{Adv}^{\mathrm{ind}}_{\mathrm{aead}}(\mathcal{A}'')\big),$$

where the advantages are defined using the indistinguishability games corresponding to the primitive.

Theorem 4 (informal version). Fix any IFP I. If the key assignment scheme kas provides indistinguishability and the AEAD scheme aead provides integrity, then our RWES construction rwes provides integrity. More precisely, for any adversary $\mathcal{A}$ that queries at most q_o different objects there exist adversaries $\mathcal{A}',\mathcal{A}''$ of comparable efficiency such that

$$\mathbf{Adv}^{\mathrm{int}}_{\mathrm{rwes}}(I,\mathcal{A}) \le q_o \cdot \big(\mathbf{Adv}^{1\text{-kind}}_{\mathrm{kas}}(I,\mathcal{A}') + \mathbf{Adv}^{\mathrm{int}}_{\mathrm{aead}}(\mathcal{A}'')\big),$$

where the advantages are defined using the games corresponding to the primitive.

7 Security proofs

We prove our ROES and RWES constructions from Sections 5 and 6 secure in their respective models. By inspection one can readily verify that both constructions are safe. In this section we will answer the adversary's oracle queries by forwarding to oracles in a different game or running a procedure. If this results in an error message, the adversary receives the error symbol, $\perp$, as described in Sec. 3.1.

To keep our proofs concise and to the point, we do not explicitly mention the advantage of finding a collision in the hash function each time. Instead, we assume the function H in Fig. 5 and Fig. 8 is the identity function.

7.1 ROES security proofs

Theorem 1. *Let* roes *be the construction specified in Fig. 5, I an IFP, $t = |O|$, $\mathcal{A}$ an adversary, and* $\mathbf{Adv}^{\text{ind}}_{\text{roes}}(I, \mathcal{A})$ *the advantage that $\mathcal{A}$ has against construction* roes *in the ROES indistinguishability games of Fig. 4. For any adversary $\mathcal{A}$ there exist adversaries $\mathcal{A}', \mathcal{A}''$ of comparable efficiency such that*

$$\mathbf{Adv}^{\text{ind}}_{\text{roes}}(I, \mathcal{A}) \leq t \cdot \left(\mathbf{Adv}^{\text{1-kind}}_{\text{kas}}(I, \mathcal{A}') + \mathbf{Adv}^{\text{ind}}_{\text{enc}}(\mathcal{A}'')\right).$$

Proof. By Lemma 3 (below) we can replace the keys for each challenge object by an independent uniformly random key. What remains is exactly t independent instances of encryptment: $\mathcal{A}$ is provided with m^0, m^1, bt and c for each challenge object and has to guess b, which yields $\mathbf{Adv}^{\text{ind}}_{\text{roes}}(I, \mathcal{A}) \leq \mathbf{Adv}^{t\text{-kind}}_{\text{kas}} + t \cdot \mathbf{Adv}^{\text{ind}}_{\text{enc}}(\mathcal{A}'')$. The theorem statement immediately follows from applying Lemma 1 to reduce from the t-challenge to single-challenge KAS game. □

Lemma 3. *Let* roes *be the ROES construction from Theorem 1,* roes^t *the construction similar to* roes *with the exception that for each challenge object the* derive *algorithm is replaced by sampling a key $k \leftarrow \$(\mathcal{K})$, I an IFP and $t = |O|$. For any adversary $\mathcal{A}$ there exists an adversary $\mathcal{A}'$ with comparable efficiency such that*

$$\mathbf{Adv}^{\text{ind}}_{\text{roes}}(I, \mathcal{A}) \leq \mathbf{Adv}^{\text{ind}}_{\text{roes}^t}(I, \mathcal{A}) + \mathbf{Adv}^{t\text{-kind}}_{\text{kas}}(I, \mathcal{A}').$$

Proof. To prove the result we will show we can use an adversary $\mathcal{A}$ that can distinguish whether the game IND calls roes or roes^t to win KIND_t with non-negligible advantage. $\mathcal{A}'$ will initialize its own KIND_t game and simulate the IND game to $\mathcal{A}$. To do so, $\mathcal{A}'$ runs the setup procedure of roes. However, instead of calling the derive$'(\sigma'_{u'}, \pi, o, \#)$ procedure, it will query the $\text{Derive}(u', \pi, o, \#)$ oracle in its KIND_t game. The Derive oracle can only fail if its internal derive procedure fails and in this case $\mathcal{A}'$ will propagate the error, letting derive$'$ fail with the same error as derive. Otherwise, to obtain a key, $\mathcal{A}'$ will call its own $\text{Challenge}(o, \#)$ oracle for a challenge object and $\text{Reveal}(o, \#)$ for a non-challenge object. We remark these oracle queries will succeed because we derived the keys, there have not been any corruptions yet and $t = |O|$. For any $\text{Read}(u, \bar{\pi}, o, c)$ query $\mathcal{A}$ makes, $\mathcal{A}'$ will first check if $H(\bar{\pi}) = h'$ holds and fail otherwise. Next, $\mathcal{A}'$ will query $\text{Derive}(u, \bar{\pi}, o, c)$ because it needs to propagate an error if this query were to fail. If the query completes successfully, $\mathcal{A}'$ can use its key obtained during the setup phase to run the dec$'$ procedure and return the result (either a message or an error) to $\mathcal{A}$. We observe $\mathcal{A}$ is not allowed to corrupt any users that are authorized to access challenge objects so for any Corrupt query, $\mathcal{A}'$ can simply forward to its own oracle: both oracles have the exact same requirement. Now observe that the simulation always succeeds and $\mathcal{A}'$ simulates IND with roes if it is playing KIND^0_t and IND with roes^t if it is playing KIND^1_t. When $\mathcal{A}$ makes its guess, $\mathcal{A}'$ will make the corresponding guess in its own game. □

Theorem 2. *Let* roes *be the ROES construction specified in Fig. 5, I an IFP, $t = |O|$, $\mathcal{A}$ an adversary, and* $\mathbf{Adv}^{\text{int}}_{\text{roes}}(I, \mathcal{A})$ *the advantage that $\mathcal{A}$ has against construction* roes *in the ROES integrity game of Fig. 4. For any adversary $\mathcal{A}$ there exist an adversary $\mathcal{A}'$ of comparable efficiency such that*

$$\mathbf{Adv}^{\text{int}}_{\text{roes}}(I, \mathcal{A}) \leq t \cdot \mathbf{Adv}^{\text{int}}_{\text{enc}}(\mathcal{A}').$$

Proof. $\mathcal{A}'$ will simulate the INT game for ROES by running roes with the exception that each $\text{enc}'(k, m)$ call in setup is replaced by initializing an $\text{INT}(k, m, \mathcal{A}')$ game for Encryptment, which will return the required pair (bt, c). The initialization of INT runs enc, so if this fails $\mathcal{A}'$ will propagate the error, letting enc′ fail with the same error as enc. Since $\mathcal{A}'$ holds the secret state for each user, it can answer any Corrupt query by $\mathcal{A}$. When $\mathcal{A}$ makes a $\text{Read}(u, \bar{\pi}, o, c)$ query, $\mathcal{A}'$ will first check if $H(\bar{\pi}) = h'$ holds and fail otherwise. Next, it will run $\text{derive}(\sigma'_u, \bar{\pi}, o, \#)$ to obtain a key k. Because roes uses a safe KAS construction, k will be equal to the key $\mathcal{A}'$ initialized the game with. Hence $\mathcal{A}'$ can replace $\text{dec}'(k, bt, \bar{c})$ in read by a $\text{Dec}(\bar{c})$ query in the corresponding INT game. We conclude Dec will execute $\text{dec}(k, bt, \bar{c})$ and thus reward $\mathcal{A}'$ if $c \neq \bar{c}$. This is exactly the win condition for $\mathcal{A}$. We remark again that if any of the above procedures run by $\mathcal{A}'$ were to fail, it will propagate the error to $\mathcal{A}$. □

7.2 RWES security proofs

Theorem 3. *Let* rwes *be the RWES construction specified in Fig. 8, I an IFP, $\mathcal{A}$ an adversary, and* $\mathbf{Adv}^{t\text{-ind}}_{\text{rwes}}(I, \mathcal{A})$ *the advantage that $\mathcal{A}$ has against construction* rwes *in the RWES indistinguishability games of Fig. 7 that allows t* Challenge *queries. For any adversary $\mathcal{A}$ that queries at most q_o different objects (i.e., $q_o \leq |O|$) there exist adversaries $\mathcal{A}', \mathcal{A}''$ of comparable efficiency such that*

$$\mathbf{Adv}^{t\text{-ind}}_{\text{rwes}}(I, \mathcal{A}) \leq t \cdot q_o \cdot \big(\mathbf{Adv}^{1\text{-kind}}_{\text{kas}}(I, \mathcal{A}') + \mathbf{Adv}^{\text{ind}}_{\text{aead}}(\mathcal{A}'')\big).$$

Proof. The statement follows as a corollary from Lemma 2 (above), and Lemmas 4 and 5 (below). □

Lemma 4. *Let* rwes *be the RWES construction from Theorem 3,* rwes^i *the construction similar to* rwes *with the exception that for the i-th new object the* derive *algorithm is replaced by sampling a key $k \leftarrow \$(\mathcal{K})$, and I an IFP. For any adversary $\mathcal{A}$ that queries at most q_o different objects there exists an adversary $\mathcal{A}'$ with comparable efficiency such that*

$$\mathbf{Adv}^{1\text{-ind}}_{\text{rwes}}(I, \mathcal{A}) \leq \mathbf{Adv}^{1\text{-ind}}_{\text{rwes}^i}(I, \mathcal{A}) + q_o \cdot \mathbf{Adv}^{1\text{-kind}}_{\text{kas}}(I, \mathcal{A}').$$

Proof. To prove the result we will show we can use an adversary $\mathcal{A}$ that can distinguish whether the game IND^b_1 calls rwes or rwes^i to win KIND_1 with non-negligible advantage. $\mathcal{A}'$ will initialize its own KIND_1 game, pick $i \leftarrow \$(\{1, \ldots, q_o\})$, create an array of uninitialized keys $\text{K}[\cdot] \leftarrow \times$ and run $\mathcal{A}$. If $\mathcal{A}$ makes an $\text{Enc}(u, \bar{\pi}, o, ad, m)$ or $\text{Dec}(u, \bar{\pi}, o, ad, c)$ query $\mathcal{A}'$ will first check if $H(\bar{\pi}) = h'$ holds and fail otherwise. Next, $\mathcal{A}'$ will call $\text{Derive}(u, \pi, o, \#)$. If $\mathcal{A}'$ has already stored the key for the object in $\text{K}[o, \#]$, it will reuse the key. Otherwise, $\mathcal{A}'$ will call $\text{Reveal}(o, \#)$ if o is not the i-th object queried and $\text{Challenge}(o, \#)$ if o is the i-th object. The respective oracle will return a key k and $\mathcal{A}'$ will set $\text{K}[o, \#] \leftarrow k$. Finally, $\mathcal{A}'$ will use $\text{K}[o, \#]$ to encrypt or decrypt the message or ciphertext by running enc′ or dec′, respectively. For $\mathcal{A}$'s $\text{Challenge}(u, \pi, o, ad, m^0, m^1)$ query $\mathcal{A}'$ will abort if o is not the i-th object. Otherwise it will use $\text{K}[o, \#]$ to encrypt m^b. Note if $\text{K}[o, \#]$ is still uninitialized, $\mathcal{A}'$ will obtain the key described above for Enc queries, which $\mathcal{A}'$ can do because no users with access to the object have been corrupted. Finally, to answer $\text{Corrupt}(u)$ queries $\mathcal{A}'$ will check if user u can access the i-th object. If the user can access it, $\mathcal{A}'$ will abort. (Either immediately, or later when the i-th object becomes known.) Otherwise, it will

forward the Corrupt query to its own game and return σ_u to $\mathcal{A}$. We remark again that if any of the above procedures run by $\mathcal{A}'$ were to fail, it will propagate the error to $\mathcal{A}$. Now observe that the simulation succeeds if $\mathcal{A}'$ guessed correctly that the i-th object would be challenged. Moreover, $\mathcal{A}'$ simulates IND_1 with rwes if it is playing KIND_1^0 and IND_1 with rwes^i if it is playing KIND_1^1. When $\mathcal{A}$ makes its guess, $\mathcal{A}'$ will make the corresponding guess in its own game. □

Lemma 5. *Let* rwes^i *be the construction from Lemma 4 and I an IFP. For any adversary $\mathcal{A}$ that queries at most q_o different objects there exists an adversary $\mathcal{A}'$ with comparable efficiency such that*

$$\mathbf{Adv}^{\text{1-ind}}_{\text{rwes}^i}(I, \mathcal{A}) \leq q_o \cdot \mathbf{Adv}^{\text{ind}}_{\text{aead}}(\mathcal{A}').$$

Proof. $\mathcal{A}'$ picks $i \leftarrow \$(\{1, \ldots, q_o\})$, initializes rwes^i and maintains all secret states to answer $\mathcal{A}$'s oracle queries, with the exception of queries related to the i-th object. Corrupt queries are not allowed for any u that have access to the i-th object as $\mathcal{A}'$ guessed it is the challenge object. So in this case $\mathcal{A}'$ can abort (either during the corrupt query or later when the i-th object becomes known). For any Enc, Dec or Challenge queries related to the i-th object, $\mathcal{A}'$ will forward the query to the oracles in its own IND game and return the result to $\mathcal{A}$. (Note $\mathcal{A}'$ can call the Enc oracle in its own game with the same messages to answer $\mathcal{A}$'s Enc queries.) We remark again that if any of the above oracle queries made by $\mathcal{A}'$ were to fail, it will propagate the error to $\mathcal{A}$. Recall in the IND_1 game with rwes^i a uniformly random key gets selected for encryptions and decryptions related to the i-th object, so $\mathcal{A}'$ perfectly simulates to $\mathcal{A}$ if it guessed correctly that the i-th object would be challenged. We conclude $\mathcal{A}'$ simulates IND_1^b with rwes^i if and only if it is playing IND^b. When $\mathcal{A}$ makes its guess, $\mathcal{A}'$ will make the corresponding guess in its own game. □

Theorem 4. *Let* rwes *be the RWES construction given in Fig. 8, I an IFP, $\mathcal{A}$ an adversary, and* $\mathbf{Adv}^{\text{int}}_{\text{rwes}}(I, \mathcal{A})$ *the advantage that $\mathcal{A}$ has against construction* rwes *in the RWES integrity game of Fig. 6. For any adversary $\mathcal{A}$ that queries at most q_o different objects (i.e., $q_o \leq |O|$) there exist adversaries $\mathcal{A}', \mathcal{A}''$ of comparable efficiency such that*

$$\mathbf{Adv}^{\text{int}}_{\text{rwes}}(I, \mathcal{A}) \leq q_o \cdot \big(\mathbf{Adv}^{\text{1-kind}}_{\text{kas}}(I, \mathcal{A}') + \mathbf{Adv}^{\text{int}}_{\text{aead}}(\mathcal{A}'')\big).$$

Proof. The proof follows from Lemmas 6 and 7 (below). □

Lemma 6. *Let* rwes *be the RWES construction from Theorem 4,* rwes^i *the construction similar to* rwes *with the exception that for the i-th new object the* derive *algorithm is replaced by sampling a key $k \leftarrow \$(\mathcal{K})$, and I an IFP. For any adversary $\mathcal{A}$ that queries at most q_o different objects there exists an adversary $\mathcal{A}'$ with comparable efficiency such that*

$$\mathbf{Adv}^{\text{int}}_{\text{rwes}}(I, \mathcal{A}) \leq \mathbf{Adv}^{\text{int}}_{\text{rwes}^i}(I, \mathcal{A}) + q_o \cdot \mathbf{Adv}^{\text{1-kind}}_{\text{kas}}(I, \mathcal{A}').$$

Proof. To prove the result we will show we can use an adversary $\mathcal{A}$ that can distinguish whether the game INT calls rwes or rwes^i to win KIND_1 with non-negligible advantage. $\mathcal{A}'$ will initialize its own KIND_1 game, pick $i \leftarrow \$(\{1, \ldots, q_o\})$ create an array of uninitialized keys $\text{K}[\cdot] \leftarrow \times$ and run $\mathcal{A}$. If $\mathcal{A}$ makes an $\text{Enc}(u, \bar{\pi}, o, ad, m)$ or $\text{Dec}(u, \bar{\pi}, o, ad, c)$ query $\mathcal{A}'$ will first check if $H(\bar{\pi}) = h'$ holds and fail otherwise. Next, $\mathcal{A}'$ will call $\text{Derive}(u, \pi, o, \#)$. If $\mathcal{A}'$ has already stored the key for the object in $\text{K}[o, \#]$, it will reuse the key. Otherwise, $\mathcal{A}'$ will call $\text{Reveal}(o, \#)$ if o is not the i-th object queried and $\text{Challenge}(o, \#)$ if o is the i-th object. The respective oracle will return a key k and $\mathcal{A}'$ will set $\text{K}[o, \#] \leftarrow k$. Finally, $\mathcal{A}'$ will use $\text{K}[o, \#]$ to encrypt or decrypt the message or ciphertext by running enc' or dec', respectively. To answer $\text{Corrupt}(u)$ queries $\mathcal{A}'$ will check if user u can access the i-th object. If the user can access it, $\mathcal{A}'$ will abort as $\mathcal{A}$ will not win with a forgery for object o. ($\mathcal{A}'$ will either abort immediately, or later when the i-th object becomes known.)

Otherwise, it will forward the Corrupt query to its own game and return σ_u to $\mathcal{A}$. We remark again that if any of the above procedures run by $\mathcal{A}'$ were to fail, it will propagate the error to $\mathcal{A}$. Now observe that the simulation succeeds if $\mathcal{A}'$ guessed correctly that a ciphertext for the i-th object would be forged. Moreover, $\mathcal{A}'$ simulates INT with rwes if it is playing KIND_1^0 and INT with rwes^i if it is playing KIND_1^1. When $\mathcal{A}$ makes its guess, $\mathcal{A}'$ will make the corresponding guess in its own game. □

Lemma 7. *Let* rwes^i *be the construction from Lemma 6 and* I *an IFP. For any adversary* $\mathcal{A}$ *that queries at most* q_o *different objects there exists an adversary* $\mathcal{A}'$ *with comparable efficiency such that*

$$\mathbf{Adv}^{\text{int}}_{\text{rwes}^i}(I, \mathcal{A}) \leq q_o \cdot \mathbf{Adv}^{\text{int}}_{\text{aead}}(\mathcal{A}').$$

Proof. $\mathcal{A}'$ picks $i \leftarrow \$(\{1, \ldots, q_o\})$, initializes rwes^i and maintains all secret states to answer $\mathcal{A}$'s oracle queries, with the exception of queries related to the i-th object. Corrupt queries are not allowed for any u that have access to the i-th object as $\mathcal{A}'$ guessed $\mathcal{A}$ wins the game with a forgery for this object. So in this case $\mathcal{A}'$ can abort (either during the corrupt query or later when the i-th object becomes known). For any Enc or Dec queries related to the i-th object, $\mathcal{A}'$ will forward the query to the oracles in its own INT game and return the result to $\mathcal{A}$. We remark again that if any of the above oracle queries made by $\mathcal{A}'$ were to fail, it will propagate the error to $\mathcal{A}$. Recall in the INT game with rwes^i a uniformly random key gets selected for encryptions and decryptions related to the i-th object, so $\mathcal{A}'$ perfectly simulates to $\mathcal{A}$ if it guessed correctly that the i-th object would be used for a forgery. When $\mathcal{A}$ forges, $\mathcal{A}'$ will have forwarded the query to the Dec oracle and won in its own game. □

8 Conclusion

The cryptographic primitive that we consider has a long history in practically implementing hierarchical access control. After giving a careful critique of recent prior work (published in ToSC 2018(4), presented at FSE 2019), we note that there are two main profiles in access control: read-only access and read-write access. These call for an individual treatment with dedicated models and constructions. We deliver precisely this, and observe that while the one primitive is naturally built from regular AEAD, the other requires a stronger symmetric building block, Encryptment, that has recently been proposed in a considerably different context.

Acknowledgments

The research of Pijnenburg was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1). We thank the reviewers of ToSC for their particularly detailed and helpful comments.

References

[ABFF09] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):18:1–18:43, January 2009.

[ADFM06] Giuseppe Ateniese, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Provably-secure time-bound hierarchical key assignment schemes. In Ari Juels,

Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 288–297. ACM Press, October / November 2006.

[ADFM12] Giuseppe Ateniese, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Provably-secure time-bound hierarchical key assignment schemes. *Journal of Cryptology*, 25(2):243–270, April 2012.

[AFB05] Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. Dynamic and efficient key management for access hierarchies. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 2005*, pages 190–202. ACM Press, November 2005.

[AS87] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

[AT83] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.

[CC02] Tzer-Shyong Chen and Yu-Fang Chung. Hierarchical access control based on Chinese Remainder Theorem and symmetric algorithm. *Computers & Security*, 21(6):565–570, 2002.

[CDM10] Jason Crampton, Rosli Daud, and Keith M. Martin. Constructing key assignment schemes from chain partitions. In Sara Foresti and Sushil Jajodia, editors, *Data and Applications Security and Privacy XXIV, 24th Annual IFIP WG 11.3 Working Conference, Rome, Italy, June 21-23, 2010. Proceedings*, volume 6166 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.

[CDM+16] Arcangelo Castiglione, Alfredo De Santis, Barbara Masucci, Francesco Palmieri, and Aniello Castiglione. On the relations between security notions in hierarchical key assignment schemes for dynamic structures. In Joseph K. Liu and Ron Steinfeld, editors, *ACISP 16, Part II*, volume 9723 of *LNCS*, pages 37–54. Springer, Heidelberg, July 2016.

[CFG+15] Jason Crampton, Naomi Farley, Gregory Gutin, Mark Jones, and Bertram Poettering. Cryptographic enforcement of information flow policies without public information. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 389–408. Springer, Heidelberg, June 2015.

[CFG+17] Jason Crampton, Naomi Farley, Gregory Z. Gutin, Mark Jones, and Bertram Poettering. Cryptographic enforcement of information flow policies without public information via tree partitions. *Journal of Computer Security*, 25(6):511–535, 2017.

[CMW06] Jason Crampton, Keith M. Martin, and Peter R. Wild. On key assignment for hierarchical access control. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*, pages 98–111. IEEE Computer Society, 2006.

[CT17] Yi-Ruei Chen and Wen-Guey Tzeng. Hierarchical key assignment with dynamic read-write privilege enforcement and extended KI-security. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 165–183. Springer, Heidelberg, July 2017.

[DDFM09] Paolo D'Arco, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Security and tradeoffs of the Akl-Taylor scheme and its variants. In Rastislav Královic and Damian Niwinski, editors, *Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009, Novy Smokovec, High Tatras, Slovakia, August 24-28, 2009. Proceedings*, volume 5734 of *Lecture Notes in Computer Science*, pages 247–257. Springer, 2009.

[DDFM10] Paolo D'Arco, Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Variations on a theme by Akl and Taylor: Security and tradeoffs. *Theor. Comput. Sci.*, 411(1):213–227, 2010.

[DFM07] Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Efficient provably-secure hierarchical key assignment schemes. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 2007.

[DFM11] Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Efficient provably-secure hierarchical key assignment schemes. *Theor. Comput. Sci.*, 412(41):5684–5699, 2011.

[DGRW18] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186. Springer, Heidelberg, August 2018.

[FP11] Eduarda S. V. Freire and Kenneth G. Paterson. Provably secure key assignment schemes from factoring. In Udaya Parampalli and Philip Hawkes, editors, *ACISP 11*, volume 6812 of *LNCS*, pages 292–309. Springer, Heidelberg, July 2011.

[FPP13] Eduarda S. V. Freire, Kenneth G. Paterson, and Bertram Poettering. Simple, efficient and strongly KI-secure hierarchical key assignment schemes. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 101–114. Springer, Heidelberg, February / March 2013.

[Gud80] Ehud Gudes. The design of a cryptography based secure file system. *Software Engineering, IEEE Transactions on*, SE-6:411–420, 10 1980.

[HL90] Lein Harn and Hung-Yu Lin. A cryptographic key generation scheme for multilevel data security. *Computers & Security*, 9(6):539–546, 1990.

[KP18a] Suyash Kandele and Souradyuti Paul. Key assignment scheme with authenticated encryption. *IACR Trans. Symm. Cryptol.*, 2018(4):150–196, 2018.

[KP18b] Suyash Kandele and Souradyuti Paul. Key assignment scheme with authenticated encryption. Cryptology ePrint Archive, Report 2018/1233, 2018. https://eprint.iacr.org/2018/1233.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer, Heidelberg, August 2002.

[MTMA85] Stephen J. MacKinnon, Peter D. Taylor, Henk Meijer, and Selim G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Trans. Computers*, 34(9):797–802, 1985.

[PP20] Jeroen Pijnenburg and Bertram Poettering. Key assignment schemes with authenticated encryption, revisited. Cryptology ePrint Archive, Report 2020/612, 2020. https://eprint.iacr.org/2020/612.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, November 2002.

[Shi07] R. Shirey. Internet Security Glossary, Version 2. RFC 4949, August 2007.

[Sho04] Victor Shoup. ISO 18033-2: An emerging standard for public-key encryption. http://shoup.net/iso/std6.pdf, December 2004. Final Committee Draft.

[Tze06] Wen-Guey Tzeng. A secure system for data access based on anonymous authentication and time-dependent hierarchical keys. In Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Lin, Shiuhpyng Shieh, and Sushil Jajodia, editors, *ASIACCS 06*, pages 223–230. ACM Press, March 2006.

[WC01] Tzong-Chen Wu and Chin-Chen Chang. Cryptographic key assignment scheme for hierarchical access control. *International Journal of Computer Systems Science and Engineering*, 16(1), 2001.

A Support Material for Section 2

A.1 Scheme A

The encryption scheme specified in Fig. 9 is formally secure according to the IND-PRV and INT notions from [KP18a, Fig. 2]. However, as quite simple attacks show, the scheme offers neither confidentiality (against passive let alone active adversaries) nor authenticity in an intuitive sense. The ideas behind the construction are: (a) encryption keys are additively made up of two shares, k^0 and k^1, and each encryption operation leaks (via the tag) one of the two shares to the public, meaning that key recovery may be possible after seeing just two ciphertexts, and (b) the essential part of the tag is computed using a public function.

Proc gen	**Proc** enc(k, m)	**Proc** dec(k, c, t)
00 $k^0 \leftarrow \$(\mathcal{K})$	05 $(k^0, k^1, k^+) \leftarrow k$	11 $(k^0, k^1, k^+) \leftarrow k$
01 $k^1 \leftarrow \$(\mathcal{K})$	06 $c \leftarrow \text{enc}'(k^+, m)$	12 $(u, v) \leftarrow t$
02 $k^+ \leftarrow k^0 \oplus k^1$	07 $b \leftarrow \lvert m \rvert \bmod 2$	13 $v' \leftarrow H(c)$
03 $k \leftarrow (k^0, k^1, k^+)$	08 $(u, v) \leftarrow (k^b, H(c))$	14 Require $v' = v$
04 Return k	09 $t \leftarrow (u, v)$	15 $m \leftarrow \text{dec}'(k^+, c)$
	10 Return (c, t)	16 Return m

Figure 9: We let $\mathcal{K} = \{0,1\}^{256}$, write $\oplus$ for the bit-wise exclusive-or combination of two same-length strings, write $\lvert\cdot\rvert$ for the length of a string (e.g., in bits), and use enc', dec' and H as placeholders for CPA-secure encryption/decryption algorithms with key space $\mathcal{K}$, and a collision-resistant hash function, respectively. If the condition in line 14 is not fulfilled, the dec algorithm fails.

A.2 Scheme B

The encryption scheme specified in Fig. 10 is an encrypt-then-mac design where the MAC component is itself composed of a universal hash function (UHF) followed by a

pseudorandom function. This construction is formally secure according to the IND-PRV and INT notions from [KP18a, Fig. 2], and also according to the classic notions. It is a standard property of the most efficient UHFs that collisions can be efficiently computed if the hashing key is known. For instance, if the UHF is instantiated via polynomial hashing (as in GCM or Poly1305), a couple of field operations are sufficient for this. This shows that the goal of the INT notion, namely to ensure that for any tag at most one valid ciphertext can be found, immediately has to be given up once key k becomes public.

Proc gen	**Proc** enc(k, m)	**Proc** dec(k, c, t)
00 $k_e \leftarrow \$(\mathcal{K})$	05 $(k_e, k_h, k_t) \leftarrow k$	10 $(k_e, k_h, k_t) \leftarrow k$
01 $k_h \leftarrow \$(\mathcal{K})$	06 $c \leftarrow \text{enc}'(k_e, m)$	11 $h \leftarrow \text{UHF}(k_h, c)$
02 $k_t \leftarrow \$(\mathcal{K})$	07 $h \leftarrow \text{UHF}(k_h, c)$	12 $t' \leftarrow F(k_t, h)$
03 $k \leftarrow (k_e, k_h, k_t)$	08 $t \leftarrow F(k_t, h)$	13 Require $t' = t$
04 Return k	09 Return (c, t)	14 $m \leftarrow \text{dec}'(k_e, c)$
		15 Return m

Figure 10: We let $\mathcal{K} = \{0,1\}^{256}$, and use enc′, dec′ and UHF and F as placeholders for CPA-secure encryption/decryption algorithms, a universal hash function, and a pseudorandom function, respectively, all with key space $\mathcal{K}$. If the condition in line 13 is not fulfilled, the dec algorithm fails.

B Macros for game termination

Table 1 reproduces definitions from Sec. 3.1 in tabular form.

Table 1: Macros for game termination

Win	—	Stop with `T`
Lose	—	Stop with `F`
Reward *cond*	—	If *cond*: Win
Penalize *cond*	—	If *cond*: Lose
Promise *cond*	—	If ¬*cond*: Win
Require *cond*	—	If ¬*cond*: Lose

C Encryptment from One-Time Encryption and Hashing

In Sec. 3.3 we recall the definition of encryptment from [DGRW18]. Quite obviously, this primitive can be realized by combining regular symmetric encryption with a cryptographic hash function. For completeness, and without claiming novelty, we specify the details of this construction in Fig. 11. To obtain an encryptment scheme enc, dec for a message space $\mathcal{M}$, the required building blocks are a one-time passively secure symmetric encryption scheme enc′, dec′ for $\mathcal{M}$ and a collision resistant hash function $H\colon \{0,1\}^* \to \{0,1\}^l$, for a suitable value l. The security argument is trivial.

D Simple KAS transforms

In Fig. 12 we indicate two simple transforms that illustrate that the syntactical and semantical changes to KAS that we carried out in Sec. 4.2 are minor. Concretely, the transform in Fig. 12 (top) shows how support for associated data can be retrofitted

Proc $\mathrm{enc}(k, m)$	**Proc** $\mathrm{dec}(k, bt, c)$
00 $c \leftarrow \mathrm{enc}'(k, m)$	03 Require $H(c) = bt$
01 $bt \leftarrow H(c)$	04 $m \leftarrow \mathrm{dec}'(k, c)$
02 Return (bt, c)	05 Return m

Figure 11: Construction of encryptment. If the condition in line 03 is not fulfilled, the dec algorithm fails.

into a classical KAS by outputting as the derived (ad-dependent) key the output of a pseudorandom function F keyed with a classical (ad-independent) key and evaluated on the ad input. The second transform in Fig. 12 (bottom) shows how a collision-resistant hash function H can be used to protect a KAS with explicit input of a public state against attacks where the adversary tricks users to derive keys using an unauthentic public state.

Proc $\mathrm{setup}(I)$	**Proc** $\mathrm{derive}(\sigma, \pi, o, ad)$
00 $(\vec{\sigma}, \pi) \leftarrow \mathrm{setup}'(I)$	02 $k' \leftarrow \mathrm{derive}'(\sigma, \pi, o)$
01 Return $(\vec{\sigma}, \pi)$	03 $k \leftarrow F(k', ad)$
	04 Return k
Proc $\mathrm{setup}(I)$	**Proc** $\mathrm{derive}(\sigma, \pi, o)$
05 $(\vec{\sigma}', \pi) \leftarrow \mathrm{setup}'(I)$	10 $(h', \sigma') \leftarrow \sigma$
06 $h' \leftarrow H(\pi)$	11 Require $H(\pi) = h'$
07 For all $u \in U$:	12 $k' \leftarrow \mathrm{derive}'(\sigma', \pi, o)$
08 $\quad \sigma_u \leftarrow (h', \sigma'_u)$	13 Return k'
09 Return $(\vec{\sigma}, \pi)$	

Figure 12: Two constructions of a KAS $\mathrm{setup}, \mathrm{derive}$ from a KAS $\mathrm{setup}', \mathrm{derive}'$. We assume building blocks $F\colon \mathcal{K} \times \mathcal{AD} \to \mathcal{K}$ and $H\colon \{0,1\}^* \to \{0,1\}^{256}$. If the condition in line 11 is not fulfilled, the derive algorithm fails.

E RWES Construction from KAS and AE

In Fig. 13 we specify a construction of RWES that is secure according to our definitions. It is very similar to that of Fig. 8 but leverages the associated-data input of KAS such that the construction could be implemented with an AE scheme instead of an AEAD scheme.

Proc $\mathrm{setup}(I)$	**Proc** $\mathrm{enc}(\sigma_u, \pi, o, ad, m)$	**Proc** $\mathrm{dec}(\sigma_u, \pi, o, ad, c)$
00 $(\vec{\sigma}', \pi) \leftarrow \mathrm{setup}'(I)$	05 $(h', \sigma'_u) \leftarrow \sigma_u$	10 $(h', \sigma'_u) \leftarrow \sigma_u$
01 $h' \leftarrow H(\pi)$	06 Require $H(\pi) = h'$	11 Require $H(\pi) = h'$
02 For all $u \in U$:	07 $k \leftarrow \mathrm{derive}'(\sigma'_u, \pi, o, ad)$	12 $k \leftarrow \mathrm{derive}'(\sigma'_u, \pi, o, ad)$
03 $\quad \sigma_u \leftarrow (h', \sigma'_u)$	08 $c \leftarrow \mathrm{enc}'(k, \#, m)$	13 $m \leftarrow \mathrm{dec}'(k, \#, c)$
04 Return $(\vec{\sigma}, \pi)$	09 Return c	14 Return m

Figure 13: Our alternative RWES construction with procedures $\mathrm{setup}, \mathrm{enc}, \mathrm{dec}$ using procedures $\mathrm{setup}', \mathrm{derive}'$ of a generic KAS kas, procedures $\mathrm{enc}', \mathrm{dec}'$ of a generic AEAD scheme aead, and a collision-resistant hash function H. In [08,13] we write # for any fixed associated-data string; in particular this can be the empty string. If the conditions in [06,11] are not fulfilled, the respective algorithm fails.

F Discussion of Security Games

In this section we will refer to line numbers in the security games using square brackets.

F.1 AEAD

In Fig. 1 we provide security games for AEAD. Here we discuss some of the subtleties in the games. We remark in the SAFE game in [06] we promise $c \notin \mathrm{C}[ad]$, implying encryption is randomized. It should be obvious the promise in [12] ensures that the dec procedures outputs the correct message if the ciphertext was output of the Enc oracle. In this case we can also overwrite the message, as it is already known to the adversary. Next, we recall an algorithm may fail. In particular, the integrity game makes critical use of this: the dec procedure must reject unauthentic ciphertexts such that [23], which rewards the adversary, is not executed. Again, for consistency, we can overwrite the message because the game would have ended if it was not output of the Enc oracle. Finally, in the indistinguishability games in [36] we crucially overwrite the message if the ciphertext was output of the Enc oracle such that the adversary does not trivially learn which message was encrypted.

F.2 Encryptment

In Fig. 2 we provide security games for encryptment. Here we discuss some of the subtleties in the games. We remark the games are now specified for a specific message, but this is not restrictive as the advantage is defined as a maximum over all $m \in \mathcal{M}$. Similarly, the SAFE and INT games are specified for a specific key with the advantage defined as the maximum over all keys. Moreover, the adversary is provided with the key, as safety and integrity should be maintained even when the key becomes public. For the indistinguishability games we do not provide the key to the adversary as this would allow for trivial decryption. In addition, we must sample a key uniformly at random. Otherwise, an adversary which always attempts decryption for a hard-coded key would have an advantage of 1 when we take the maximum over all keys. Finally, recall an algorithm may fail. In particular, the integrity game crucially depends on this: the dec procedure must reject unauthentic ciphertexts such that [12], which rewards the adversary, is not executed. Similarly to F.1, we overwrite the decrypted message if it was generated by the game such that in the indistinguishability games the adversary does not trivially learn which message was encrypted.

F.3 KAS

In Fig. 3 we provide security games for KAS. Here we discuss some of the key ideas in the games. In the SAFE game we promise the user has access to the object [05]. This implies for any safe construction the derive procedure will fail when it attempts to derive a key for an object the user cannot access. If an authentic public state is used, the SAFE game promises each user derives identical keys for each object given they use identical associated data [08]. Note the SAFE game makes no such promise for unauthentic public states. In the KIND games we remark an adversary is only allowed to either Reveal or Challenge a key for a specific object o and associated data ad, but not both. We encode this by requiring the queried key is in the key space [17,31]. Correspondingly, we overwrite $\mathrm{K}[o, ad]$ in [19,35] with $\diamond$ to encode revealed/challenged keys. Since uninitialized keys are also not in the key space, the adversary must first Derive the keys it wishes to use. For any key derived with an authentic public state the adversary can decide to reveal or challenge it. Clearly, the Derive oracle must not output challengeable keys. So, if an authentic public state was provided, we overwrite the key with $\star$ to encode this [29]. If an unauthentic public state was provided, and the derive procedure outputs a key, the

Derive oracle will simply return the key to the adversary. Finally, we note to avoid trivial attacks the adversary is not allowed to Challenge and Corrupt (a user that can access) the same object. The remaining lines in these oracles actively track queries and exclude these attacks.

F.4 ROES

In Fig. 4 we provide security games for ROES. Here we discuss some of the subtleties in the games. For the SAFE game it is again important to recall an algorithm is allowed to fail. In particular, if a user is unauthorized the read procedure must fail. Indeed, it is promised read will only retrieve messages for authorized users [04]. The integrity game rewards the adversary (for any object) if the read procedure accepts any ciphertext that was not generated by the game for the specified object. The corruption of any user is allowed, capturing the fact that even insiders should not be able to create different, valid ciphertexts. The indistinguishability games are specified for all functions from $\mathcal{M}$ to O. In a similar fashion to our other indistinguishability notions, we only consider the combination of two functions if each object is mapped to equivalent messages under them [20]. We mark messages as challenge messages if they are different [21]. The Corrupt oracle prevents trivial attacks where the adversary corrupts a user authorized to access a challenge message [30]. The corruption of users which do not have access to challenge messages precisely captures that unauthorized users should have no information that can help distinguish these messages.

F.5 RWES

In Fig. 6 and Fig. 7 we provide security games for RWES. Here we discuss some of the subtleties in the games. In the SAFE game, we first remark we promise only authorized users are able to encrypt and decrypt messages. This implies the enc and dec procedures must fail for unauthorized users. Next, we promise in [09] that $c \notin \mathrm{C}[o, ad]$, implying encryption is randomized. It should be obvious the promise in [16] ensures that the dec procedures outputs the correct message if the ciphertext was output of the Enc oracle when provided with an authentic public state. Note that the INT game also makes critical use of the fact procedures can fail: the dec procedure must reject unauthentic ciphertexts such that the oracle aborts and the adversary is not rewarded. Of course, we only reward the adversary for a forgery if the adversary had not corrupted a user with access to the object. Finally, in the IND game in [20] we overwrite the message if the ciphertext was output of the Enc or Challenge oracle such that the adversary does not trivially learn which message was encrypted in the case of a Challenge query. We note that in the case of an Enc query, the adversary already knows the message anyway, so it does not need to learn this from the Dec oracle. The remaining lines in the Challenge and Corrupt oracles are there to track these queries and exclude trivial attacks where the adversary challenges and corrupts the same object.

G One-time to t-time reductions

G.1 KAS

Proof of Lemma 1. Let H_i denote the hybrid of the KIND_t game where the first i challenge queries output the real key and the remaining challenge queries output a random key. Clearly $H_0 = \mathrm{KIND}_t^1$ and $H_t = \mathrm{KIND}_t^0$. We define $\mathbf{Adv}^{\mathrm{hyb}}_{i,i+1}(I, \mathcal{A}) := |\Pr[H_i(I, \mathcal{A})] - \Pr[H_{i+1}(I, \mathcal{A})]|$. By the triangle inequality we have $\mathbf{Adv}^{t\text{-kind}}(I, \mathcal{A}) \leq \sum_{i=0}^{t-1} \mathbf{Adv}^{\mathrm{hyb}}_{i,i+1}(I, \mathcal{A})$, so there must exist a j s.t. $0 \leq j < t$ and $\mathbf{Adv}^{t\text{-kind}}(I, \mathcal{A}) \leq$

$t \cdot \mathbf{Adv}^{\text{hyb}}_{j,j+1}(I, \mathcal{A})$. To prove the result it remains to show we can use an adversary $\mathcal{A}$ that can distinguish between H_i and H_{i+1} to win the KIND_1 game with non-negligible advantage. $\mathcal{A}'$ will initialise its own game KIND_1 and run $\mathcal{A}$. Any queries that $\mathcal{A}$ makes to the Corrupt, Derive and Reveal oracles are forwarded to $\mathcal{A}'$'s own oracles after checking for the requirements in the game oracles. ($\mathcal{A}'$ will abort if the requirements are not met as $\mathcal{A}$ would lose the game anyway.) When $\mathcal{A}$ makes a Challenge(o, ad) query $\mathcal{A}'$ will first check it is a valid challenge. If it is not valid $\mathcal{A}'$ will abort as $\mathcal{A}$ would lose the game anyway. Otherwise $\mathcal{A}'$ will proceed as follows, where q_c counts the number of preceding challenge queries:

- If $q_c < j$: $\mathcal{A}'$ makes the corresponding Reveal (and Derive if necessary) query, adds (o, ad) to CH and returns k to $\mathcal{A}$.
- If $q_c = j$: $\mathcal{A}'$ makes the corresponding Challenge (and Derive if necessary) query, adds (o, ad) to CH and returns k to $\mathcal{A}$.
- If $q_c > j$: $\mathcal{A}'$ makes the corresponding Derive query if necessary, picks $k \leftarrow_{\$} \mathcal{K}$, adds (o, ad) to CH and returns k to $\mathcal{A}$.

Now observe $\mathcal{A}'$ simulates H_j to $\mathcal{A}$ in the case $\mathcal{A}'$ is playing KIND^1_1 and H_{j+1} in the case $\mathcal{A}'$ is playing KIND^0_1. When $\mathcal{A}$ makes its guess, $\mathcal{A}'$ will make the corresponding guess in its own game. We conclude $\mathbf{Adv}^{\text{hyb}}_{j,j+1}(I, \mathcal{A}) \leq \mathbf{Adv}^{\text{1-kind}}(I, \mathcal{A}')$.

G.2 RWES

Proof of Lemma 2. Let H_i denote the hybrid of the IND_t game where the first i challenge queries output the encryption of m_0 and the remaining challenge queries output the encryption of m_1. Clearly $H_0 = \text{IND}^1_t$ and $H_t = \text{IND}^0_t$. We define $\mathbf{Adv}^{\text{hyb}}_{i,i+1}(I, \mathcal{A}) := |\Pr[H_i(I, \mathcal{A})] - \Pr[H_{i+1}(I, \mathcal{A})]|$. By the triangle inequality we have $\mathbf{Adv}^{t\text{-ind}}(I, \mathcal{A}) \leq \sum_{i=0}^{t-1} \mathbf{Adv}^{\text{hyb}}_{i,i+1}(I, \mathcal{A})$, so there must exist a j s.t. $0 \leq j < t$ and $\mathbf{Adv}^{t\text{-ind}}(I, \mathcal{A}) \leq t \cdot \mathbf{Adv}^{\text{hyb}}_{j,j+1}(I, \mathcal{A})$. To prove the result it remains to show we can use an adversary $\mathcal{A}$ that can distinguish between H_i and H_{i+1} to win the IND_1 game with non-negligible advantage. $\mathcal{A}'$ will initialise its own game IND_1 and run $\mathcal{A}$. Any queries that $\mathcal{A}$ makes to the Corrupt, Enc and Dec oracles are forwarded to $\mathcal{A}'$'s own oracles after checking for the requirements in the game oracles. ($\mathcal{A}'$ will abort if the requirements are not met as $\mathcal{A}$ would lose the game anyway.) When $\mathcal{A}$ makes a Challenge$(u, \pi, o, ad, m^0, m^1)$ query $\mathcal{A}'$ will first check it is a valid challenge. If it is not valid $\mathcal{A}'$ will abort as $\mathcal{A}$ would lose the game anyway. Otherwise $\mathcal{A}'$ will proceed as follows, where q_c counts the number of preceding challenge queries:

- If $q_c < j$: $\mathcal{A}'$ makes an Enc(u, π, o, ad, m^0) query, adds (o, ad) to CH and returns c to $\mathcal{A}$.
- If $q_c = j$: $\mathcal{A}'$ forwards the Challenge query, adds (o, ad) to CH and returns c to $\mathcal{A}$.
- If $q_c > j$: $\mathcal{A}'$ makes an Enc(u, π, o, ad, m^1) query, adds (o, ad) to CH and returns c to $\mathcal{A}$.

Now observe $\mathcal{A}'$ simulates H_j to $\mathcal{A}$ in the case $\mathcal{A}'$ is playing IND^1_1 and H_{j+1} in the case $\mathcal{A}'$ is playing IND^0_1. When $\mathcal{A}$ makes its guess, $\mathcal{A}'$ will make the corresponding guess in its own game. We conclude $\mathbf{Adv}^{\text{hyb}}_{j,j+1}(I, \mathcal{A}) \leq \mathbf{Adv}^{\text{1-ind}}(I, \mathcal{A}')$.