An extended abstract of this article was published with DOI 10.1007/978-3-031-30731-7_3 in the proceedings of SSR'23.
This is the full version, and available in the IACR eprint archive as article 2023/272.

A study of KEM generalizations

Bertram Poettering[1] and Simon Rastikian[1,2]

[1] IBM Research Europe – Zurich, Rüschlikon, Switzerland
[2] ETH Zurich, Zurich, Switzerland

Abstract. The NIST, in its recent competition on quantum-resilient confidentiality primitives, requested the submission of exclusively KEMs. The task of KEMs is to establish secure session keys that can drive, amongst others, public key encryption and TLS-like secure channels. In this work we test the KEM abstraction in the context of constructing cryptographic schemes that are not subsumed in the PKE and secure channels categories. We find that, when used to construct a key transport scheme or when used within a secure combiner, the KEM abstraction imposes certain inconvenient limits, the settling of which requires the addition of auxiliary symmetric primitives.

We hence investigate generalizations of the KEM abstraction that allow a considerably simplified construction of the above primitives. In particular, we study VKEMs and KDFEMs, which augment classic KEMs by label inputs, encapsulation handle outputs, and key derivation features, and we demonstrate that they can be transformed into KEM combiners and key transport schemes *without* requiring auxiliary components. We finally show that all four finalist KEMs of the NIST competition are effectively KDFEMs. Our conclusion is that only very mild adjustments are necessary to significantly increase their versatility.

1 Introduction

Hybrid Encryption. The contemporary approach to construct public key encryption (PKE) is via the KEM+DEM paradigm [11]: To encrypt a message $m \in \mathcal{M}$, first a key encapsulation mechanism (KEM) is used to establish a session key $k \in \mathcal{K}$, then a data encapsulation mechanism (DEM) is used to symmetrically encrypt message m with session key k. The fundamental lemma of hybrid encryption guarantees that if both KEM and DEM are secure against active adversaries, then also the resulting PKE scheme is secure against active adversaries [11].

A main advantage of constructing PKE from two separate primitives is the gain in flexibility: The KEM can be chosen to meet one specific set of conditions (e.g. related to ciphertext size/expansion, resilience against quantum adversaries, level of standardization, ROM vs. standard model, ...), and the DEM can be chosen to meet a different set of conditions (e.g. related to its performance on the expected computing architecture, the type of underlying primitive, ...). While the KEM+DEM paradigm is now about two decades old, its attractiveness was recently confirmed when the NIST opened their call for quantum resilient cryptographic schemes, where all encryption primitive submissions were explicitly required to be of the KEM type [1].

Key Transport. A key transport (KT) scheme is a public-key primitive that allows users to securely transport 'symmetric' keys to other users. More specifically, KT can be seen as a special case of PKE where the message space $\mathcal{M}$ is restricted to a payload key space of the form $\bar{\mathcal{K}} = \{0,1\}^\kappa$, commonly instantiated with $\kappa = 128$ or $\kappa = 256$. Standard applications of KT include OpenPGP email encryption [9] where for each email that is encrypted a fresh session key $\bar{k}$ is randomly sampled from $\bar{\mathcal{K}}$ and then transported, via KT, to all recipients of the email. The latter involves one KT operation per receiver, and implicitly represents a multi-recipient PKE construction [19].

If one wants to construct a KT scheme from a KEM, the simple approach of first establishing a session key k with the KEM and then appending the one-time pad encryption $\bar{k} \oplus k$ of $\bar{k}$ to its ciphertext is, due to the obvious malleability condition, not secure against active adversaries. Rather, it appears that a stronger encryption primitive is necessary. For instance, $\bar{k}$ could be encrypted via $c \leftarrow \mathrm{enc}_k(\bar{k})$ where enc is a DEM encapsulation routine that is secure against active adversaries. In practice, the natural options for instantiating such a DEM would be using either EtM (encrypt-then-mac, [5]) or authenticated encryption (AE/AEAD, [20]). Unfortunately, these approaches imply overheads that are inconvenient in two independent dimensions: (1) At least one auxiliary symmetric algorithm has to be agreed on and implemented (two in the case of EtM), and the effective price of this should not be underestimated.[3] (2) AE/AEAD schemes expect auxiliary inputs like nonces [21] and associated data strings [20], the processing of which requires additional resources.[4] Note that the nonce processing is demanded by the AE/AEAD interface [18], while our KT application itself wouldn't require it (and could fix the nonce to the all-zero string). The price of processing the nonce has to be paid anyway.

Starting from (a generalized form of) a KEM, this article contributes a KT construction that does not require any auxiliary symmetric algorithm. That is, our KT scheme completely removes the two overhead categories discussed above.

KEM Combiners. A KEM combiner merges two ingredient KEMs into a single (hybrid) KEM such that if at least one of the ingredient KEMs is secure then so is the hybrid. Interest in KEM combiners increased recently [2] with the availability of KEMs that are potentially resilient against quantum adversaries: While hardness assumptions in the domain of lattices and codes can be considered less tested than RSA/DL, only the former have the potential to provide security once quantum computers become available; hence, combining a classic KEM with a lattice or code based KEM promises to achieve security in more scenarios. Similarly to above (see KT discussion), the simple construction of first letting the ingredient KEMs establish session keys k_1, k_2 independently of each other and then combining these keys to a common key via $k \leftarrow k_1 \oplus k_2$ does, due to malleability issues, not provide security against active adversaries.

KEM combiners secure against active adversaries have been investigated in [15,8]. All known constructions require auxiliary symmetric primitives, namely either blockciphers, PRFs, or hash functions. For instance, the likely most elegant hybrid from [15] has an encapsulation routine that lets the ingredient algorithms $\mathrm{enc}_1, \mathrm{enc}_2$ establish session keys k_1, k_2 independently of each other, and then computes the hybrid key as per $k \leftarrow F(k_1, c_2) \oplus F(k_2, c_1)$, where F is an auxiliary PRF and c_1, c_2 are the ingredient KEMs' ciphertexts. Another example of a KEM combiner would derive the combined key k as per $k \leftarrow H(k_1, k_2, c_1, c_2)$, for a (quantum) random oracle H. As we discussed in the KT context, the use of auxiliary symmetric components comes with a price that should not be underestimated.

Starting from (a generalized form of) a KEM, this article contributes a KEM combiner that does not require any auxiliary symmetric algorithm.

1.1 Existing KEM generalizations

As discussed above, neither key transport nor KEM combiners seem to be constructable from KEMs directly, i.e., without adding an auxiliary symmetric primitive of some kind. The goal of this article is to study whether a relatively small strengthening of the KEM primitive might suffice to enable the construction of key transport or KEM combiners without adding extra primitives. We are not the first authors to consider strengthenings of the KEM primitive. In the following we review three prior approaches (all of which were originally explored with an overall different focus).

Labeled PKE/KEMs. In labeled PKE [23], the encryption and decryption algorithms take, in addition to their standard inputs (public or secret key, message or ciphertext), an auxiliary label input L which may consist of an arbitrary string. Correctness is provided if and only if encryption and decryption use the same label. That is, intuitively, $\forall L, m\colon \mathrm{dec}(sk, L, \mathrm{enc}(pk, L, m)) = m$. The adapted security definition, which is a straightforward variant of the standard PKE security definition, implies that for $L_1 \neq L_2$ the value of $\mathrm{dec}(sk, L_2, \mathrm{enc}(pk, L_1, m))$ is not correlated with m. (Assuming that dec doesn't reject the ciphertext in the first place.) The main application of the auxiliary label input is that it easily allows to implement domain separation. For instance, if the same PKE instance is relied on both for receiving encrypted emails and for authenticating to services (by proving the ability to decrypt challenge

[3] Firstly, agreeing on an auxiliary component will likely require dedicated standardization efforts. Secondly, side-channel resilient implementations of cryptographic algorithms require knowledge of the target machine and hence, in the worst case, one dedicated implementation per computing architecture.

[4] For instance, the nonce handling of most AES-based AE/AEAD schemes requires one additional blockcipher invocation.

ciphertexts), if the labels `"enc"` and `"auth"` are used to logically separate to two applications, it is ensured that the otherwise obvious attacks are not possible.

The idea of adding a label input to the PKE interface was formalized in [23]. Translating the idea to the KEM world is immediate: Intuitively, for correctness we now would demand that $\forall L\colon \mathrm{enc}(pk, L) = (c, k) \implies \mathrm{dec}(sk, L, c) = k$. Also the adaptation of the security notions is straightforward.

TagKEMs. It was observed by Abe *et al.* [3] that certain IND-CCA secure KEM constructions (e.g., in the spirit of Cramer–Shoup encryption [11]) contain an internal mechanism that authenticates ciphertexts in such a way that the decryptor can detect and reject malicious ciphertext manipulations. One idea behind their TagKEM primitive is to use the same authentication mechanism to also protect the DEM ciphertext of a KEM+DEM hybrid. To make this practical, the KEM encapsulation is split into two algorithms, enc_1 and enc_2, such that first enc_1 is executed on input the public key and with output the session key k plus some state information, then session key k is used with a DEM to encrypt the payload message $\bar{m}$ which results in a DEM ciphertext $\bar{c}$, and finally enc_2 is executed on input the state information and $\bar{c}$ as a *tag*, and with output the KEM ciphertext c. The TagKEM decapsulation routine is not split, and would recover k from sk, c and tag $\bar{c}$, so that then $\bar{m}$ can be recovered from $\bar{c}$ via DEM decapsulation.

While the TagKEM concept was specifically developed to allow the construction of efficient PKE schemes, the fact that its encapsulation routine is split and can authenticate the *use* of the session keys might find more general applications. This article will draw on a very similar concept.

We note that while labels (see above) and tags (see here) serve slightly different purposes, both of them represent arbitrary strings that are known to both sender and receiver, and significantly control the behavior of the corresponding algorithms. This concept also appears in other areas of cryptography, e.g., in the form of associated-data strings in AEAD [20], or as a tweak input for blockciphers [17]. As it will become clear in the course of this paper, it is meaningful in our generalizations to use one single term for the label/tag inputs of KEMs; we chose to consistently use the term 'label'.

KEMs with Handles. A KEM can be seen as a special form of a one-pass key establishment (KE) protocol for two parties where only one party is authenticated (via a public key). Early models for key establishment [7] define security via session transcripts that would match (or not) on both sides. To side-step drawbacks implied by this purely syntactical approach, later models, e.g. [6], adopted the idea of letting protocol instances also output an explicit *session id*, the matching of which would replace the matching of transcripts. More concretely, if participant Alice establishes session id/key pair (sid_A, k_A) and Bob establishes pair (sid_B, k_B), then, intuitively, correctness would demand that $sid_A = sid_B \implies k_A = k_B$ ("same session, same key"), while the security definition would demand that if $sid_A \neq sid_B$ then k_A, k_B are not correlated ("different session, independent key"). The main advantage of models with session id is that the concept of matching sessions is made explicit and clear, and that obviously correct protocols that couldn't be proven in the model of [7] (for purely syntactical reasons) suddenly become tractable.

Given that KEMs represent a special KE case, it makes sense to explore introducing the session id concept also to KEMs. As in the KE world, this can only increase the number of tractable constructions. However, as establishing a shared key using a KEM doesn't really involve creating a 'session', in this article we use the term 'encapsulation handle' instead of 'session id'; we often just write *handle* for short. Syntactically, a KEM supporting handles encapsulates via $(c, hd, k) \leftarrow \mathrm{enc}(pk)$ and decapsulates via $(hd', k') \leftarrow \mathrm{dec}(sk, c)$. The KE correctness condition translates to $hd' = hd \implies k' = k$ ("same handle, same key"), and for security we demand that if $hd' \neq hd$ then k', k are not correlated ("different handle, independent key").

We observe that the classic KEM notion also provides a handle concept, but only implicitly: In standard correctness and security definitions for KEMs [11], the ciphertext takes a dual role: (1) It conveys the information necessary for the decryptor to reconstruct the session key, and (2) it serves as a handle for the encapsulation operation: Each KEM ciphertext uniquely identifies the invocation of the encapsulation algorithm that created it. Also the "same ciphertext, same key" and "different ciphertext, independent key" principles hold for (IND-CCA secure) KEMs.

While all formalizations in this article consistently use handle based definitions for KEMs and PKE, readers unfamiliar or uncomfortable with this concept can, whenever a handle is mentioned, instead think of the ciphertext. This way of thinking *does* reduce the generality of our results, but only mildly so. We will make those cases explicit where the difference is significant.

1.2 Our approach

The goal of this article is to find and study natural generalizations of the KEM primitive such that intuitively simple applications like key transport (KT) and KEM combiners can be constructed without having to rely on auxiliary symmetric building blocks. In our search we considered it a necessary condition that the KEM generalization wouldn't change the main character profile of a KEM too much. For instance, we insisted on the overall communication from sender to receiver remaining one-pass. In the end our search identified two different KEM generalizations, dubbed VKEM and KDFEM, that we briefly present in the upcoming paragraphs. We found in particular the KDFEM primitive suitable for our purposes.

VKEMs. In Sect. 1.1 we discussed three already existing KEM generalizations from prior work: KEMs with labels, with tags, and with handles. What we call a versatile key encapsulation mechanism (VKEM) is a KEM variant that combines all three of these approaches, in a clean and unified way, with the ultimate goal of maximum versatility: A VKEM has *both* the encapsulation and decapsulation routine split into two phases each, where the algorithms of both phases take labels on input and generate keys and handles on output. (See Fig. 4 for a high-level illustration of the syntax.)

After defining the precise syntax and security of VKEMs, we study how a KEM combiner and/or KT scheme can be constructed from this primitive. The encapsulation routine of our VKEM combiner from Sect. 5 is illustrated in Fig. 1. As the red crosses suggest, the label input and the session key output of the two first-phase VKEM invocations (top left and top right) are not used. In contrast, the first-phase encapsulation handles, serving as identifiers for the respective encapsulation invocations, are fed, in form of labels, into the second phase of the *other* VKEM instance. The idea behind this cross-over is to cryptographically tie the two VKEM instances together, so that an attack against the one can be noticed, and reacted to, by the other. The hybrid KEM's key k is the XOR of the two second-phase session keys, while the hybrid's handle hd is the concatenation of the two second-phase handles. We formally confirm the security of this construction in Sect. 5.

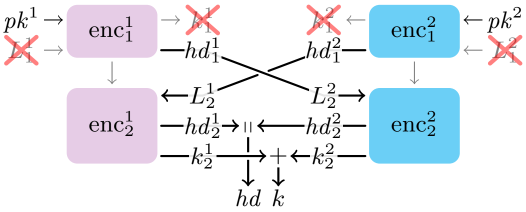

Fig. 1. Combiner of two VKEMs (left and right) to obtain one secure KEM. We only show the encapsulation process, and, for clarity, omit drawing the arrows transporting ciphertexts.

We also succeeded with transforming a VKEM into a KT scheme. The construction is a little odd for allowing empty second-phase ciphertexts and using the second-phase session keys exclusively, in the style of MAC tags, in cleartext for authentication. While this was confusing at first, we eventually noticed that the very same key transport scheme could also be instantiated with a KDFEM (see below) instead of with a VKEM, meaning that its requirements are located in the small intersection of VKEMs and KDFEMs. As the KDFEM notation is substantially cleaner when it comes to defining a KT scheme, we decided to present our KT solution and its analysis exclusively in the KDFEM setting.

KDFEMs. Our second approach to generalizing KEMs is based on the observation that many real-world KEM constructions internally derive the output session key with a dedicated key derivation function (KDF) like HKDF [16]. While KDFs allow for deriving many keys from a single seed, we are not aware of a KEM construction that would evaluate its KDF at more than one point. Our approach is to remove this restriction and to enable the evaluation of the (seeded) KDF on arbitrarily many points. Very briefly, what we refer to as a key derivation function encapsulation mechanism (KDFEM) consists of encapsulation/decapsulation algorithms $(c, st) \leftarrow \mathrm{enc}(pk)$ and $st' \leftarrow \mathrm{dec}(sk, c)$ and a KDF evaluation algorithm eval such that $k \leftarrow \mathrm{eval}(st, L)$ and $k' \leftarrow \mathrm{eval}(st', L)$ lead to the same result $k = k'$.

We observe that the KDFEM primitive allows for constructing both a KEM combiner and a KT scheme in an extremely straightforward manner: The $k \leftarrow F(k_1, c_2) \oplus F(k_2, c_1)$ construction of [15] discussed earlier can be salvaged by replacing the (auxiliary) PRF with the KDFEM's eval routine: Using our handle-based notation, the instruction becomes $k \leftarrow \mathrm{eval}(st_1, hd_2) \oplus \mathrm{eval}(st_2, hd_1)$. Pronto. Our key transport construction is as simple: A first eval invocation establishes a mask that is used to one-time pad encrypt the payload key, and a second eval invocation is used on the resulting ciphertext to protect its integrity in an encrypt-then-mac fashion. We formally confirm the security of these constructions in Sect. 7 and Sect. 9.

Discussion. Our approach to expect of a generalized KEM that it expose a new kind of auxiliary KDF functionality may at first seem moot given that our overall goal was to *reduce* the number of auxiliary symmetric primitives (including KDFs) required to construct KEM combiners and KT. It's not. The key insight is that many KEMs already have that KDF functionality built into them, so we can re-use it for free. The cost reduction of our approach is not necessarily visible in computation time or the like, but in the removed requirement to agree on an additional primitive. Concretely, in Sect. 10 we demonstrate that all four KEM finalists of the recent NIST competition [1] can be turned into KDFEMs with almost no modification.

1.3 Related work

We already gave numerous references to related work inline in the above paragraphs. This includes work on KEMs with labels, with tag inputs, and of primitives that establish keys together with handles. We also mentioned relevant standardization efforts like ETSI TS 103 744 [2] and the ongoing, soon-to-be-completed efforts by NIST [1]. The public interest in KEM combiners is also visible in the existence of an RFC draft that explicitly targets this primitive (tolerating an auxiliary random oracle).[5]

The works of Zhang et al. [24], Dodis and Katz [13], Giacon et al. [15], as well as Bindel et al. [8] consider combiners for public key encryption and key encapsulation mechanisms. While the former two works consider PKE and their results cannot be translated to the KEM setting, the latter two combine KEMs but require additional building blocks. In this sense, they don't present solutions to our challenge.

Numerous practical protocols, including development versions of TLS and MLS, employ KEM combiners or KT schemes only implicitly. This is typically done via key mixing, using auxiliary symmetric primitives like hash functions or KDFs. A difference to our setting is that TLS and MLS are generously using such primitives anyway, so that the advantages offered by our approach become less considerable.

2 Preliminaries

2.1 Notation

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. If var is a set variable and exp evaluates to a set, we write $var \overset{\cup}{\leftarrow} exp$ shorthand for $var \leftarrow var \cup exp$. A (row) vector variable can be appended to another vector variable with the (associative) concatenation operator $\mathbin{\|}$, and we write $var \overset{\|}{\leftarrow} exp$ shorthand for $var \leftarrow var \mathbin{\|} exp$. We do *not* overload the $\mathbin{\|}$ operator to also indicate string concatenation, i.e., the objects $\mathtt{a} \mathbin{\|} \mathtt{b}$ and $\mathtt{ab}$ are not the same. We use $[\,]$ notation for associative arrays (i.e., the 'dictionary' data structure): Once the instruction $A[\cdot] \leftarrow exp$ initialized all items of array A to the default value exp, individual items can be accessed as per $A[idx]$, e.g., updated and extracted via $A[idx] \leftarrow exp$ and $var \leftarrow A[idx]$, respectively, for any expression idx.

To keep our games compact, we use the alias-creating operator ":=" where convenient. The instruction '$A := B$' introduces A as a symbolic alias for the expression B. This crucially differs from $A \leftarrow B$ which is an assignment that evaluates expression B and stores the result in variable A. For instance, if $D[\,]$ is a

[5] https://datatracker.ietf.org/doc/draft-ounsworth-cfrg-kem-combiners/.

dictionary and $D[\texttt{"x"}]$ an integer entry, and an alias is created as per $A := D[\texttt{"x"}]$, then the instruction $A \leftarrow A + 1$ expands to $D[\texttt{"x"}] \leftarrow D[\texttt{"x"}] + 1$ and thus modifies the value of $D[\texttt{"x"}]$.

Unless explicitly noted, any scheme algorithm may be randomized. We use $\langle\,\rangle$ notation for stateful algorithms: If *alg* is a (stateful) algorithm, we write $y \leftarrow \mathit{alg}\langle \mathit{st}\rangle(x)$ shorthand for $(\mathit{st}, y) \leftarrow \mathit{alg}(\mathit{st}, x)$ to denote an invocation with input x and output y that updates its state *st*. (Depending on the algorithm, x and/or y may be missing.) If in a specific context one of the output elements of an algorithm shall be ignored, we annotate this by assigning it to the symbol _. Importantly, and in contrast to most prior works, we assume that *any* algorithm of a cryptographic scheme may fail or abort, even if this is not explicitly specified in the syntax definition. This approach is inspired by how modern programming languages deal with error conditions via *exceptions*: Any code can at any time 'throw an exception' which leads to an abort of the current code and is passed on to the calling instance. In particular, if in our game definitions a scheme algorithm aborts, the corresponding game oracle immediately aborts as well (and returns to the adversary).

Security games are parameterized by an adversary, and consist of a main game body plus zero or more oracle specifications. The adversary is allowed to call any of the specified oracles. The execution of the 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 (over the random coins of G and $\mathcal{A}$) that an execution of G with adversary $\mathcal{A}$ results in the outcome 1. We define shorthand notation for specific combinations of game-ending instructions: While in computational games we write 'Win' for 'Stop with 1', in distinguishing games we write 'Win' for 'Stop with b' (where b is the challenge bit). In any case we write 'Lose' for 'Stop with 0'. Further, for a Boolean condition C, we write '**Require** C' for 'If $\neg C$: Lose', 'Penalize C' for 'If C: Lose', 'Reward C' for 'If C: Win', and '**Promise** C' for 'If $\neg C$: Win'.

Many of the oracles specified in a security game will produce information that is considered public and to be shared with the adversary. This holds for instance for a ciphertext c created within an encryption oracle. Instead of collecting such information in an explicit data structure and returning it to the adversary when the processing of the oracle finishes, we use the **Share** shortcut notation to perform the same job implicitly. (In the above case we would write 'Share c'.) If required, this concept could be formalized by initializing a list $\mathrm{L} \leftarrow \epsilon$ when the game starts, by appending the arguments of any Share instruction to this list (e.g., $\mathrm{L} \overset{\shortparallel}{\leftarrow} c$), and to return L from any oracle query. We chose our implicit notation as it uses less symbols and makes the game mechanics more clear.

2.2 Key establishment games

Most of the cryptographic primitives considered in this work (KEMs, VKEMs, KDFEMs) are key establishing primitives: Their goal is to establish fresh session keys that can be used with arbitrary applications. While, not surprisingly, each such primitive is covered by individual security definitions and games, some parts of these definitions overlap and are common across all formalizations. Instead of specifying the same game components over and over again, we define and describe the common parts here and refer to them from the main body of our treatment.

In Fig. 2 we define the core part that the formalizations of all our key establishing primitives have in common. The game body (lines 00–03) initializes a secret/public key pair, invokes the adversary on input the public key, checks for trivial win conditions (see below), and terminates the game with the output provided by the adversary. The adversary can invoke a number of oracles (depending on the modeled primitive), among which are always the Reveal and Challenge oracles specified here. (Some works in the key establishment literature may refer to our Challenge oracle as the Test oracle.) Both oracles provide access to a key that was priorly accepted (see lines 04,08; entries will be added to set A by other oracles). The Reveal oracle always returns the real key (stored in array K, line 06), and the Challenge oracle either returns the real key or a random key (line 10). (Array R is initialized to random keys, see the Initializations: line at the top of the figure.)

Intuitively, if the adversary reveals a specific key, the latter becomes exposed. We record this in set X (line 05). If however the adversary tests a key by invoking the Challenge oracle, the key is thereby declared fresh. We record this in set F (line 09). It is a trivial attack to first reveal a key and then test it (or vice versa); hence, in line 02, the game aborts (Stops with 0) if this condition is identified. Based on the $\mathbf{KE}^0, \mathbf{KE}^1$ games specified in Fig. 2 (plus additional scheme specific oracles), a typical advantage of an adversary would be defined as $\mathbf{Adv}(\mathcal{A}) := |\Pr[\mathbf{KE}^0(\mathcal{A})] - \Pr[\mathbf{KE}^1(\mathcal{A})]|$.

```
INITIALIZATIONS: $\mathrm{A}, \mathrm{X}, \mathrm{F} \leftarrow \emptyset$; $\mathrm{K}[\cdot] \leftarrow \diamond$; $\mathrm{R}[\cdot] \leftarrow \$(\mathcal{K})$

Game $\mathbf{KE}^b(\mathcal{A})$
00 $(sk, pk) \leftarrow \mathrm{gen}$
01 $b' \leftarrow \mathcal{A}(pk)$
02 Require $\mathrm{X} \cap \mathrm{F} = \emptyset$
03 Stop with $b'$

Oracle Reveal$(hd)$
04 Require $hd \in \mathrm{A}$
05 $\mathrm{X} \overset{\cup}{\leftarrow} \{hd\}$
06 $k \leftarrow \mathrm{K}[hd]$
07 Return $k$

Oracle Challenge$(hd)$
08 Require $hd \in \mathrm{A}$
09 $\mathrm{F} \overset{\cup}{\leftarrow} \{hd\}$
10 $k \leftarrow b \,?\, \mathrm{K}[hd] : \mathrm{R}[hd]$
11 Return $k$
```

Fig. 2. Game components for general key establishment. Legend: A: accepted; X: exposed; F: fresh; K: key; R: random.

3 Key Encapsulation Mechanisms (KEM)

As a warm-up we define a KEM variant that supports encapsulation handles: Each encapsulation generates a fresh such handle, and a corresponding decapsulation operation can recover it from the ciphertext. In contrast to the established session key, the handle is considered public information. As discussed in Sect. 1.1, the handle concept is borrowed from the key establishment literature where handles reside under the name of *session id* [6].

Definition 1. *A* key encapsulation mechanism (KEM) *for (session) key space $\mathcal{K}$ consists of a secret key space $\mathcal{SK}$, a public key space $\mathcal{PK}$, a ciphertext space $\mathcal{C}$, an encapsulation handle space $\mathcal{HD}$, a key generation algorithm* $\mathrm{gen} \to \mathcal{SK} \times \mathcal{PK}$*, and algorithms* enc, dec *as follows:*

$$\mathcal{PK} \to \mathrm{enc} \to \mathcal{C} \times \mathcal{HD} \times \mathcal{K} \qquad \mathcal{SK} \times \mathcal{C} \to \mathrm{dec} \to \mathcal{HD} \times \mathcal{K}$$

Intuitively, for correctness we demand that after $(sk, pk) \leftarrow \mathrm{gen}$ and $(c, hd, k) \leftarrow \mathrm{enc}(pk)$ and $(hd', k') \leftarrow \mathrm{dec}(sk, c')$ we have (1) *handle freshness:* the handle hd output by enc is unique (doesn't collide with other handles output by enc); and (2) *key recovery:* $hd' = hd \implies k' = k$.[6] We formalize this in the following.

Definition 2. *A KEM is correct if for every considerable adversary $\mathcal{A}$ the advantage function* $\mathbf{Adv}^{\mathrm{cor-kem}}(\mathcal{A}) := \Pr[(sk, pk) \leftarrow \mathrm{gen}; \text{Invoke } \mathcal{A}(pk); \text{Lose}]$ *is negligible, where the adversary has access to the oracles of Fig. 3, and the game variables* A, K *are initialized as in Fig. 2. The KEM is secure (against active adversaries) if for every considerable adversary $\mathcal{A}$ the advantage function* $\mathbf{Adv}^{\mathrm{ke-kem}}(\mathcal{A}) := |\Pr[\mathbf{KE}^0(\mathcal{A})] - \Pr[\mathbf{KE}^1(\mathcal{A})]|$ *is negligible, where the* $\mathbf{KE}^0, \mathbf{KE}^1$ *games consist of the components specified in Fig. 2 and Fig. 3.*

Note that the security definition also covers correctness (as the same Promise lines are present in both games). Observe how lines 23,24 formalize *handle freshness* while lines 26,30,32 formalize the *key recovery* demand. Lines 22,29,34 model that ciphertexts and handles and dishonestly generated session keys are not considered secret but public information. While Def. 2, as is, specifies security against active adversaries, a strengthening to IND-CCA security can be achieved by activating the gray components including lines 25,31. (For the results of this article, this will not be necessary.)

4 Versatile key encapsulation: VKEM

We formalize the first of our two KEM generalizations. As discussed in Sect. 1.1, VKEMs combine and extend the features of earlier KEM generalizations: They are two-phased as in Abe *et al.* [3], they support labels as in ISO 18033-2, and they support handles as already used in Sect. 3. We illustrate the syntax of VKEMs in Fig. 4.

Definition 3. *A* versatile key encapsulation mechanism (VKEM) *for label spaces $\mathcal{L}_1, \mathcal{L}_2$ and (session) key spaces $\mathcal{K}_1, \mathcal{K}_2$ consists of a secret key space $\mathcal{SK}$, a public key space $\mathcal{PK}$, state spaces $\mathcal{ST}_E, \mathcal{ST}_D$, ciphertext spaces $\mathcal{C}_1, \mathcal{C}_2$, encapsulation handle spaces $\mathcal{HD}_1, \mathcal{HD}_2$, a key generation algorithm* $\mathrm{gen} \to \mathcal{SK} \times \mathcal{PK}$*, and algorithms* $\mathrm{enc}_1, \mathrm{enc}_2, \mathrm{dec}_1, \mathrm{dec}_2$ *as follows:*

[6] It might be tempting to additionally require that $c' = c \implies hd' = hd$. However, as no part of our article logically depends on such a property, we abstain from *formally demanding* it.

```
INITIALIZATIONS: C[·] ← ◇

Oracle Enc()
20 $(c, hd, k) \leftarrow \mathrm{enc}(pk)$
21 $\mathrm{Accept_E}(c, hd, k)$
22 Share $c, hd$

Oracle Dec($c$)
27 $(hd, k) \leftarrow \mathrm{dec}(sk, c)$
28 $\mathrm{Accept_D}(c, hd, k)$
29 Share $hd$

Proc $\mathrm{Accept_E}(c, hd, k)$
23 Promise $hd \notin \mathrm{A}$
24 $\mathrm{A} \overset{\cup}{\leftarrow} \{hd\}$
25 $\mathrm{C}[hd] \leftarrow c$
26 $\mathrm{K}[hd] \leftarrow k$

Proc $\mathrm{Accept_D}(c, hd, k)$
30 If $hd \in \mathrm{A}$:
31     Promise $\mathrm{C}[hd] = c$
32     Promise $\mathrm{K}[hd] = k$
33 Else:
34     Share $k$
```

Fig. 3. KEM-specific oracles required by Def. 2. (By default ignore the gray components, in particular lines 25,31.) In the $\mathbf{KE}^0, \mathbf{KE}^1$ games, the adversary can query the Reveal, Challenge oracles of Fig. 2 and the Enc, Dec oracles specified here. The $\mathrm{Accept_E}, \mathrm{Accept_D}$ procedures are invoked (exclusively) from lines 21,28. See Sect. 2.1 for the meaning of instructions 'Share' and 'Promise'.

$$\mathcal{PK} \times \mathcal{L}_1 \to \mathrm{enc}_1 \to \mathcal{C}_1 \times \mathcal{HD}_1 \times \mathcal{K}_1 \times \mathcal{ST}_E$$
$$\mathcal{ST}_E \times \mathcal{L}_2 \to \mathrm{enc}_2 \to \mathcal{C}_2 \times \mathcal{HD}_2 \times \mathcal{K}_2$$
$$\mathcal{SK} \times \mathcal{L}_1 \times \mathcal{C}_1 \to \mathrm{dec}_1 \to \mathcal{HD}_1 \times \mathcal{K}_1 \times \mathcal{ST}_D$$
$$\mathcal{ST}_D \times \mathcal{L}_2 \times \mathcal{C}_2 \to \mathrm{dec}_2 \to \mathcal{HD}_2 \times \mathcal{K}_2$$

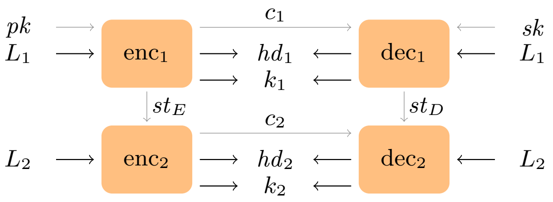

Fig. 4. Interplay of VKEM algorithms. The thick arrows are relevant for functionality/applications. The thin arrows are for technical artifacts.

In a nutshell, for correctness we demand that if the encapsulation and decapsulation algorithms are invoked and the labels and handles are consistent, then so are the established session keys. More precisely, we demand that for all $(L_1, L_2), (L'_1, L'_2) \in \mathcal{L}_1 \times \mathcal{L}_2$, after $(sk, pk) \leftarrow \mathrm{gen}$ and $(c_1, hd_1, k_1, st_E) \leftarrow \mathrm{enc}_1(pk, L_1)$ followed by $(c_2, hd_2, k_2) \leftarrow \mathrm{enc}_2(st_E, L_2)$ and $(hd'_1, k'_1, st_D) \leftarrow \mathrm{dec}_1(sk, L'_1, c'_1)$ followed by $(hd'_2, k'_2) \leftarrow \mathrm{dec}_2(st_D, L'_2, c'_2)$, we have (1) *handle freshness:* the handles hd_1, hd_2 output by $\mathrm{enc}_1, \mathrm{enc}_2$ are unique (don't collide with other handles output by enc_1 and enc_2); (2) *history matching:* $hd'_1 = hd_1 \implies L'_1 = L_1$ and $hd'_2 = hd_2 \implies (L'_1, hd'_1, L'_2) = (L_1, hd_1, L_2)$; and (3) *key recovery:* $hd'_1 = hd_1 \implies k'_1 = k_1$ and $hd'_2 = hd_2 \implies k'_2 = k_2$.[7] Before we formalize this, note that history matching is equivalent with the possibly more intuitive demand for (2') *handle divergence:* $L'_1 \neq L_1 \implies hd'_1 \neq hd_1$ and $(L'_1, hd'_1, L'_2) \neq (L_1, hd_1, L_2) \implies hd'_2 \neq hd_2$.

The oracles required by our formal definitions of correctness and security are considerably more involved than those for KEMs in Fig. 3. This is primarily because the splitting of enc, dec into two phases requires infrastructure for session management: In practice, multiple enc/dec sessions might be invoked in parallel, meaning that an expressive definition has to support concurrency. We provide further discussion after the definition.

[7] Analogously to Footnote 6, it might be tempting to additionally require $c'_1 = c_1 \implies hd'_1 = hd_1$ and $(c'_1, c'_2) = (c_1, c_2) \implies hd'_2 = hd_2$. However, as no part of our article logically depends on such a property, we once more abstain from *formally demanding* it.

Definition 4. *A VKEM is correct if for every considerable adversary $\mathcal{A}$ the advantage function* $\mathbf{Adv}^{\text{cor-vkem}}(\mathcal{A}) := \Pr[(sk, pk) \leftarrow \text{gen}; \text{Invoke } \mathcal{A}(pk); \text{Lose}]$ *is negligible, where the adversary has access to the oracles of Fig. 5, and the game variables* A, K *are initialized as in Fig. 2. The VKEM is secure (against active adversaries) if for every considerable adversary $\mathcal{A}$ the advantage function* $\mathbf{Adv}^{\text{ke-vkem}}(\mathcal{A}) := |\Pr[\mathbf{KE}^0(\mathcal{A})] - \Pr[\mathbf{KE}^1(\mathcal{A})]|$ *is negligible, where the* $\mathbf{KE}^0, \mathbf{KE}^1$ *games consist of the components specified in Fig. 2 and Fig. 5.*

```
INITIALIZATIONS: $\mathrm{ST_E}[\cdot], \mathrm{ST_D}[\cdot] \leftarrow \triangleright$; $\mathrm{H_E}[\cdot], \mathrm{C_E}[\cdot], \mathrm{H_D}[\cdot], \mathrm{C_D}[\cdot] \leftarrow \epsilon$; $\mathrm{H}[\cdot], \mathrm{C}[\cdot] \leftarrow \diamond$

Oracle $\mathrm{Enc}_1(sid, L)$
20 Require $\mathrm{ST_E}[sid] = \triangleright$
21 $(c,hd,k,st) \leftarrow \mathrm{enc}_1(pk,L)$
22 $\mathrm{ST_E}[sid] \leftarrow st$
23 $\mathrm{H_E}[sid] \overset{\shortparallel}{\leftarrow} L \shortparallel hd$
24 $\mathrm{C_E}[sid] \overset{\shortparallel}{\leftarrow} c$
25 $\mathrm{Accept_E}(sid\colon hd, k)$
26 Share $c, hd$

Oracle $\mathrm{Enc}_2(sid, L)$
27 Require $\mathrm{ST_E}[sid] \neq \triangleright, \triangleleft$
28 $st \leftarrow \mathrm{ST_E}[sid]$
29 $(c, hd, k) \leftarrow \mathrm{enc}_2(st, L)$
30 $\mathrm{ST_E}[sid] \leftarrow \triangleleft$
31 $\mathrm{H_E}[sid] \overset{\shortparallel}{\leftarrow} L \shortparallel hd$
32 $\mathrm{C_E}[sid] \overset{\shortparallel}{\leftarrow} c$
33 $\mathrm{Accept_E}(sid\colon hd, k)$
34 Share $c, hd$

Proc $\mathrm{Accept_E}(sid\colon hd, k)$
35 Promise $hd \notin \mathrm{A}$
36 $\mathrm{A} \overset{\cup}{\leftarrow} \{hd\}$
37 $\mathrm{H}[hd] \leftarrow \mathrm{H_E}[sid]$
38 $\mathrm{C}[hd] \leftarrow \mathrm{C_E}[sid]$
39 $\mathrm{K}[hd] \leftarrow k$

Oracle $\mathrm{Dec}_1(sid, L, c)$
40 Require $\mathrm{ST_D}[sid] = \triangleright$
41 $(hd,k,st) \leftarrow \mathrm{dec}_1(sk,L,c)$
42 $\mathrm{ST_D}[sid] \leftarrow st$
43 $\mathrm{H_D}[sid] \overset{\shortparallel}{\leftarrow} L \shortparallel hd$
44 $\mathrm{C_D}[sid] \overset{\shortparallel}{\leftarrow} c$
45 $\mathrm{Accept_D}(sid\colon hd, k)$
46 Share $hd$

Oracle $\mathrm{Dec}_2(sid, L, c)$
47 Require $\mathrm{ST_D}[sid] \neq \triangleright, \triangleleft$
48 $st \leftarrow \mathrm{ST_D}[sid]$
49 $(hd, k) \leftarrow \mathrm{dec}_2(st, L, c)$
50 $\mathrm{ST_D}[sid] \leftarrow \triangleleft$
51 $\mathrm{H_D}[sid] \overset{\shortparallel}{\leftarrow} L \shortparallel hd$
52 $\mathrm{C_D}[sid] \overset{\shortparallel}{\leftarrow} c$
53 $\mathrm{Accept_D}(sid\colon hd, k)$
54 Share $hd$

Proc $\mathrm{Accept_D}(sid\colon hd, k)$
55 If $hd \in \mathrm{A}$:
56   Promise $\mathrm{H}[hd] = \mathrm{H_D}[sid]$
57   Promise $\mathrm{C}[hd] = \mathrm{C_D}[sid]$
58   Promise $\mathrm{K}[hd] = k$
59 Else:
60   Share $k$
```

Fig. 5. VKEM-specific oracles required by Def. 4. (By default ignore the gray components.) In the $\mathbf{KE}^0, \mathbf{KE}^1$ games, the adversary can query the $\mathrm{Reveal}, \mathrm{Challenge}$ oracles of Fig. 2 and the $\mathrm{Enc}_1, \mathrm{Enc}_2, \mathrm{Dec}_1, \mathrm{Dec}_2$ oracles specified here. The $\mathrm{Accept_E}, \mathrm{Accept_D}$ procedures are invoked (exclusively) from lines 25,33,45,53. See Sect. 2.1 for the meaning of instructions 'Share' and 'Promise' and 'Require'.

In Fig. 5 we store the states of enc/dec sessions in the arrays $\mathrm{ST_E}, \mathrm{ST_D}$, and use the $\triangleright, \triangleleft$ symbols to identify freshly initialized and completed sessions. See lines 20,22,27,28,30,40,42,47,48,50. Note that the adversary can freely refer to any session via a self-chosen identifier *sid*.[8] We further record the input-output history of sessions in arrays $\mathrm{H_E}$ and $\mathrm{H_D}$. More precisely, every completed VKEM encapsulation or decapsulation operation defines a history h of the form $h = L_1 \shortparallel hd_1 \shortparallel L_2 \shortparallel hd_2$ that records the public information (here: the involved labels and established handles) logically associated with the established session keys k_1, k_2. These histories are recorded in lines 23,31,43,51. Note how lines 37,56 implement *history matching/handle divergence* and lines 39,58 implement *key recovery*. While Def. 4, as is, specifies security against active adversaries, a strengthening to IND-CCA security can be achieved by activating the gray components including lines 24,32,38,44,52,57. (For the results of this article, this will not be necessary.)

4.1 Label binding

In Sect. 5 we specify a KEM combiner that transforms two ingredient VKEMs into a hybrid KEM such that the hybrid is secure if at least one of the VKEMs is. As we will see, proving the security of this construction will not be possible with just the properties guaranteed by Def. 4. Rather, the security proof will require an additional, relatively mild auxiliary security property that we dub *label binding*

[8] This notion of session id has nothing to do with the one used in the key exchange literature and mentioned in Sect. 1.1. In the context of Fig. 5, session ids are not visible by any protocol algorithm. Their function is exclusively to make sessions individually addressable by the adversary.

security. This notion places a restriction on the set of possible histories h. Concretely, it says that if an encapsulation history h and a decapsulation history h' match in the first three positions (we denote this condition with $h \doteq h'$), then they actually match fully. More precisely, if history $h = L_1 \shortparallel hd_1 \shortparallel L_2 \shortparallel hd_2$ emerges from a complete encapsulation invocation, and history $h' = L'_1 \shortparallel hd'_1 \shortparallel L'_2 \shortparallel hd'_2$ emerges from a complete decapsulation invocation, we define $h \doteq h' :\Longleftrightarrow L_1 \shortparallel hd_1 \shortparallel L_2 = L'_1 \shortparallel hd'_1 \shortparallel L'_2$ and let the label binding property demand that always $h \doteq h' \implies h = h'$. As a consequence, of course, we obtain $h \doteq h' \implies hd_2 = hd'_2$.

Definition 5. *A VKEM provides label binding if for every considerable adversary $\mathcal{A}$ the advantage function $\mathbf{Adv}^{\mathrm{lb}}(\mathcal{A}) := \Pr[\mathbf{LB}(\mathcal{A})]$ is negligible, where the game consists of the components specified in Fig. 5 and Fig. 6.*

```
INITIALIZATIONS: $\mathrm{A} \leftarrow \emptyset$; $\mathrm{K}[\cdot] \leftarrow \diamond$

Game $\mathbf{LB}(\mathcal{A})$
(sk, pk) ← gen
Invoke $\mathcal{A}(pk)$
For all $h \in \mathrm{H_E}[\cdot]$ with $|h| = 4$:
    For all $h' \in \mathrm{H_D}[\cdot]$ with $|h'| = 4$:
        Promise $h \doteq h' \implies h = h'$
Stop with 0
```

Fig. 6. Game required by Def. 5 to define label binding. The adversary can query the $\mathrm{Enc}_1, \mathrm{Enc}_2, \mathrm{Dec}_1, \mathrm{Dec}_2$ oracles specified in Fig. 5.

4.2 Constructions

VKEMs condense concepts explored in several lines of prior work into a single primitive. Many interesting constructions in the spirit of the same prior work will hence exist. As we are specifically interested in what one can do with VKEMs, we briefly exemplify the primitive in Appendix A and leave a more detailed study of VKEM instantiations for future research. Our construction combines a regular KEM with a PRF and is thus very efficient.

5 KEM Combiner from VKEMs

We present a combiner that constructs a KEM from two VKEMs. The combiner is generic, in the sense that it allows running any two (correct) VKEMs such that, as long as one of the VKEMs meets **KE** security definition (strong) and the other meets the **LB** definition (weak), then the overall combined scheme forms a **KE** secure (and correct) KEM. The label-binding property is crucial for proving the combiner secure. The reason is that the combined instances cross the first phase handles to exchange information about the other VKEM (see Fig. 1 for an illustration). Intuitively, this prevents attacks against the first phase part of weak VKEM. Adding label-binding prevents malleability attacks against the second phase of the weak VKEM. The full specification of the combiner is in Fig. 7.

Theorem 1. *Let* $\mathrm{VKEM}^1 := (\mathrm{gen}^1, \mathrm{enc}^1_1, \mathrm{enc}^1_2, \mathrm{dec}^1_1, \mathrm{dec}^1_2)$ *and* $\mathrm{VKEM}^2 := (\mathrm{gen}^2, \mathrm{enc}^2_1, \mathrm{enc}^2_2, \mathrm{dec}^2_1, \mathrm{dec}^2_2)$ *be two VKEMs. Let* $\mathrm{C} := (\mathrm{gen}, \mathrm{enc}, \mathrm{dec})$ *be the KEM constructed by combining them according to Fig. 7. For all adversaries $\mathcal{A}$ attacking the security of the KEM there exist adversaries $\mathcal{B}_1, \mathcal{B}_2$ attacking the security of* $\mathrm{VKEM}^1, \mathrm{VKEM}^2$*, respectively, and adversaries $\mathcal{C}_1, \mathcal{C}_2$ attacking the label binding of* $\mathrm{VKEM}^1, \mathrm{VKEM}^2$*, respectively, such that*

$$\mathbf{Adv}^{\mathrm{ke-kem}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ke-vkem}}(\mathcal{B}_1) + \mathbf{Adv}^{\mathrm{lb}}(\mathcal{C}_2)$$

and

$$\mathbf{Adv}^{\mathrm{ke-kem}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ke-vkem}}(\mathcal{B}_2) + \mathbf{Adv}^{\mathrm{lb}}(\mathcal{C}_1)$$

where the security definitions are those of Def. 2 and Def. 4 and Def. 5.

```
Proc gen
  $(sk^1, pk^1) \leftarrow \mathrm{gen}^1$
  $(sk^2, pk^2) \leftarrow \mathrm{gen}^2$
  $sk := (sk^1, sk^2)$
  $pk := (pk^1, pk^2)$
  Return $sk, pk$

Proc enc($pk$)
  $(c_1^1, hd_1^1, \_, st^1) \leftarrow \mathrm{enc}_1^1(pk^1, \diamond)$
  $(c_1^2, hd_1^2, \_, st^2) \leftarrow \mathrm{enc}_1^2(pk^2, \diamond)$
  $(c_2^1, hd_2^1, k_2^1) \leftarrow \mathrm{enc}_2^1(st^1, hd_1^2)$
  $(c_2^2, hd_2^2, k_2^2) \leftarrow \mathrm{enc}_2^2(st^2, hd_1^1)$
  $c := (c_1^1, c_2^1, c_1^2, c_2^2)$
  $hd \leftarrow (hd_2^1, hd_2^2)$
  $k \leftarrow k_2^1 + k_2^2$
  Return $c, hd, k$

Proc dec($sk, c$)
  $(hd_1^1, \_, st^1) \leftarrow \mathrm{dec}_1^1(sk^1, \diamond, c_1^1)$
  $(hd_1^2, \_, st^2) \leftarrow \mathrm{dec}_1^2(sk^2, \diamond, c_1^2)$
  $(hd_2^1, k_2^1) \leftarrow \mathrm{dec}_2^1(st^1, hd_1^2, c_2^1)$
  $(hd_2^2, k_2^2) \leftarrow \mathrm{dec}_2^2(st^2, hd_1^1, c_2^2)$
  $hd \leftarrow (hd_2^1, hd_2^2)$
  $k \leftarrow k_2^1 + k_2^2$
  Return $hd, k$
```

Fig. 7. KEM combiner from two VKEM schemes. The instantiated combiner runs a single time both encapsulation/decapsulation chains and crosses over the handles as depicted in Fig. 1. We assume $\{\diamond\} \subseteq \mathcal{L}_1^1 = \mathcal{L}_1^2$ and $\mathcal{HD}_1^2 \subseteq \mathcal{L}_2^1$ and $\mathcal{HD}_1^1 \subseteq \mathcal{L}_2^2$. We let $\mathcal{HD} = \mathcal{HD}_2^1 \times \mathcal{HD}_2^2$.

We give an overview of the proof; the details can be found in Appendix B. Starting with the KEM security game instantiated with the algorithms of Fig. 7, we add a Promise instruction that lets adversary $\mathcal{A}$ 'win' in case its actions break label binding, i.e., if $h \doteq h'$ yet $h \neq h'$ for an encapsulation history h and a decapsulation history h'. This game hop comes at the cost of $\mathbf{Adv}^{\mathrm{lb}}(\mathcal{C})$ for some adversary $\mathcal{C}$ derived from $\mathcal{A}$. Once the condition is taken care of, the history h' of every decapsulation query has either $h' = h$ for a prior h (and is then trivial to reply to), or the labels and first-stage handle of h' are sufficiently different from any prior h such that the access rules in Fig. 5 allow for straightforwardly replying to the query in a reduction by using the session keys released by the game's line 60. That is, the remaining advantage of $\mathcal{A}$ is $\mathbf{Adv}^{\mathrm{ke-vkem}}(\mathcal{B})$ for some adversary $\mathcal{B}$ derived from $\mathcal{A}$.

6 KDF Encapsulation Mechanisms: KDFEM

We formalize the second of our two KEM generalizations. As discussed in Sect. 1.1, KDFEMs don't output session keys directly, but instead establish keyed KDF instances. These KDF instances deterministically map a domain $\mathcal{L}$ to a range $\mathcal{K}$, and can be used to derive an arbitrary number of session keys.

Definition 6. *A* KDF encapsulation mechanism (KDFEM) *for label space $\mathcal{L}$ and (session) key space $\mathcal{K}$ consists of a secret key space $\mathcal{SK}$, a public key space $\mathcal{PK}$, a state space $\mathcal{ST}$, a ciphertext space $\mathcal{C}$, an encapsulation handle space $\mathcal{HD}$, a key generation algorithm* gen $\rightarrow \mathcal{SK} \times \mathcal{PK}$, *and algorithms* enc, dec, eval *as follows:*

$$\mathcal{PK} \rightarrow \mathrm{enc} \rightarrow \mathcal{C} \times \mathcal{HD} \times \mathcal{ST} \qquad \mathcal{SK} \times \mathcal{C} \rightarrow \mathrm{dec} \rightarrow \mathcal{HD} \times \mathcal{ST} \qquad \mathcal{ST} \times \mathcal{L} \rightarrow \mathrm{eval} \rightarrow \mathcal{K}$$

Intuitively, for correctness we demand that after $(sk, pk) \leftarrow \mathrm{gen}$ and $(c, hd, st) \leftarrow \mathrm{enc}(pk)$ and $k \leftarrow \mathrm{eval}(st, L)$ and $(hd', st') \leftarrow \mathrm{dec}(sk, c')$ and $k' \leftarrow \mathrm{eval}(st', L')$ we have (1) *handle freshness:* the handle hd output by enc is unique (doesn't collide with other handles output by enc); and (2) *key recovery:* $hd' = hd \wedge L' = L \implies k' = k$.[9] We formalize this in the following.

Definition 7. *A KDFEM is correct if for every considerable adversary $\mathcal{A}$ the advantage function $\mathbf{Adv}^{\mathrm{cor-kdfem}}(\mathcal{A}) := \Pr[(sk, pk) \leftarrow \mathrm{gen}; \mathrm{Invoke}\ \mathcal{A}(pk); \mathrm{Lose}]$ is negligible, where the adversary has access to the oracles of Fig. 8 and the game variables A, K are initialized as in Fig. 2. The KDFEM is secure (against active adversaries) if for every considerable adversary $\mathcal{A}$ the advantage function $\mathbf{Adv}^{\mathrm{ke-kdfem}}(\mathcal{A}) := |\Pr[\mathbf{KE}^0(\mathcal{A})] - \Pr[\mathbf{KE}^1(\mathcal{A})]|$ is negligible, where the $\mathbf{KE}^0, \mathbf{KE}^1$ games consist of the components specified in Fig. 2 and Fig. 8.*

In Fig. 8, the session management is organized in the same way as in Fig. 5. A novelty is the splitting of set A into two: Set A^- indicates the set of (pre-)accepted enc, dec operations (lines 23,24,51) and set A indicates the accepted KDF evaluations (lines 34,35,53,54). (The latter matches precisely the spirit of our KEM/VKEM formalizations in Sect. 3 and Sect. 4.) As in our KEM/VKEM formalizations, while Def. 7, as is, specifies security against active adversaries, a strengthening to IND-CCA security can be achieved by activating the gray components in Fig. 8. (As before, for the results of this article, this will not be necessary.)

[9] Analogously to Footnotes 6 and 7, it might be tempting to additionally require that $c' = c \implies hd' = hd$. However, as no part of our article logically depends on such a property, we once more abstain from *formally demanding* it.

```
INITIALIZATIONS: $\mathrm{ST_E}[\cdot],\mathrm{ST_D}[\cdot] \leftarrow \triangleright$; $\mathrm{A}^- \leftarrow \emptyset$; $\mathrm{H_E}[\cdot],\mathrm{H_D}[\cdot],\mathrm{C}[\cdot] \leftarrow \diamond$

Oracle Enc($sid$)
20 Require $\mathrm{ST_E}[sid] = \triangleright$
21 $(c, hd, st) \leftarrow \mathrm{enc}(pk)$
22 $\mathrm{ST_E}[sid] \leftarrow st$
23 Promise $hd \notin \mathrm{A}^-$
24 $\mathrm{A}^- \xleftarrow{\cup} \{hd\}$
25 $\mathrm{C}[hd] \leftarrow c$
26 $\mathrm{H_E}[sid] \leftarrow hd$
27 Share $c, hd$

Oracle Eval_E($sid, L$)
28 Require $\mathrm{ST_E}[sid] \neq \triangleright$
29 $st \leftarrow \mathrm{ST_E}[sid]$
30 $k \leftarrow \mathrm{eval}(st, L)$
31 $hd \leftarrow \mathrm{H_E}[sid]$
32 $\mathrm{Accept_E}(hd, L, k)$

Proc Accept_E($hd, L, k$)
33 $hd \leftarrow hd \mathbin{\|} L$
34 If $hd \notin \mathrm{A}$:
35     $\mathrm{A} \xleftarrow{\cup} \{hd\}$
36     $\mathrm{K}[hd] \leftarrow k$
37 Else:
38     Promise $\mathrm{K}[hd] = k$

Oracle Dec($sid, c$)
39 Require $\mathrm{ST_D}[sid] = \triangleright$
40 $(hd, st) \leftarrow \mathrm{dec}(sk, c)$
41 $\mathrm{ST_D}[sid] \leftarrow st$
42 If $hd \in \mathrm{A}^-$:
43     Promise $\mathrm{C}[hd] = c$
44 $\mathrm{H_D}[sid] \leftarrow hd$
45 Share $hd$

Oracle Eval_D($sid, L$)
46 Require $\mathrm{ST_D}[sid] \neq \triangleright$
47 $st \leftarrow \mathrm{ST_D}[sid]$
48 $k \leftarrow \mathrm{eval}(st, L)$
49 $hd \leftarrow \mathrm{H_D}[sid]$
50 $\mathrm{Accept_D}(hd, L, k)$

Proc Accept_D($hd, L, k$)
51 If $hd \in \mathrm{A}^-$:
52     $hd \leftarrow hd \mathbin{\|} L$
53     If $hd \notin \mathrm{A}$:
54         $\mathrm{A} \xleftarrow{\cup} \{hd\}$
55         $\mathrm{K}[hd] \leftarrow k$
56     Else:
57         Promise $\mathrm{K}[hd] = k$
58 Else:
59     Share $k$
```

Fig. 8. KDFEM-specific oracles required by Def. 7. (By default ignore the gray components.) In the $\mathbf{KE}^0, \mathbf{KE}^1$ games, the adversary can query the Reveal, Challenge oracles of Fig. 2 and the Enc, $\mathrm{Eval_E}$, Dec, $\mathrm{Eval_D}$ oracles specified here. The $\mathrm{Accept_E}$, $\mathrm{Accept_D}$ procedures are invoked (exclusively) from lines 32,50. See Sect. 2.1 for the meaning of instructions 'Share' and 'Promise' and 'Require'.

7 KEM combiner from KDFEMs

We present a combiner that generically constructs a KEM from two KDFEMs. We specify the details in Fig. 9. The idea is to derive session keys as per $k \leftarrow f_1(hd_2) + f_2(hd_1)$ where f_1, f_2 represent the keyed KDF instances of the two KDFEMs. Note that, similarly to Fig. 1, the handles of the two instances are crossed.

```
Proc gen
00 $(sk^1, pk^1) \leftarrow \mathrm{gen}^1$
01 $(sk^2, pk^2) \leftarrow \mathrm{gen}^2$
02 $sk := (sk^1, sk^2)$
03 $pk := (pk^1, pk^2)$
04 Return $sk, pk$

Proc enc($pk$)
05 $(c^1, hd^1, st^1) \leftarrow \mathrm{enc}^1(pk^1)$
06 $(c^2, hd^2, st^2) \leftarrow \mathrm{enc}^2(pk^2)$
07 $k^1 \leftarrow \mathrm{eval}^1(st^1, hd^2)$
08 $k^2 \leftarrow \mathrm{eval}^2(st^2, hd^1)$
09 $c := (c^1, c^2)$
10 $hd \leftarrow (hd^1, hd^2)$
11 $k \leftarrow k^1 + k^2$
12 Return $c, hd, k$

Proc dec($sk, c$)
13 $(hd^1, st^1) \leftarrow \mathrm{dec}^1(sk^1, c^1)$
14 $(hd^2, st^2) \leftarrow \mathrm{dec}^2(sk^2, c^2)$
15 $k^1 \leftarrow \mathrm{eval}^1(st^1, hd^2)$
16 $k^2 \leftarrow \mathrm{eval}^2(st^2, hd^1)$
17 $hd \leftarrow (hd^1, hd^2)$
18 $k \leftarrow k^1 + k^2$
19 Return $hd, k$
```

Fig. 9. A KEM combiner from two KDFEM schemes. The combiner crosses handles in lines 07 and 08 during encapsulation, and in lines 15 and 16 during decapsulation. We let $\mathcal{HD} = \mathcal{HD}^1 \times \mathcal{HD}^2$.

Our security theorem states that if one of the KDFEMs meets **KE** security, then the combined scheme is a **KE** secure KEM.

Theorem 2. *Let* $\mathrm{KDFEM}^1 := (\mathrm{gen}^1, \mathrm{enc}^1, \mathrm{dec}^1, \mathrm{eval}^1)$ *and* $\mathrm{KDFEM}^2 := (\mathrm{gen}^2, \mathrm{enc}^2, \mathrm{dec}^2, \mathrm{eval}^2)$ *be two KDFEMs. Let* $\mathrm{C} := (\mathrm{gen}, \mathrm{enc}, \mathrm{dec})$ *be the KEM constructed by combining them according to Fig. 9. For all adversaries* $\mathcal{A}$ *attacking the security of the KEM there exist adversaries* $\mathcal{B}_1, \mathcal{B}_2$ *attacking the security of* $\mathrm{KDFEM}^1, \mathrm{KDFEM}^2$*, respectively, and adversaries* $\mathcal{C}_1, \mathcal{C}_2$ *attacking the correctness of* $\mathrm{KDFEM}^1, \mathrm{KDFEM}^2$*, respectively, such that*

$$\mathbf{Adv}^{\mathrm{ke-kem}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ke-kdfem}}(\mathcal{B}_1) + \mathbf{Adv}^{\mathrm{cor-kdfem}}(\mathcal{C}_2)$$

and

$$\mathbf{Adv}^{\mathrm{ke-kem}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ke-kdfem}}(\mathcal{B}_2) + \mathbf{Adv}^{\mathrm{cor-kdfem}}(\mathcal{C}_1)$$

where the security definitions are those of Def. 2 and Def. 7.

The proof is of the same flavour as the one in Sect. 5. The details can be found in Appendix C.

8 Key Transport

A key transport scheme can be seen as a PKE scheme that is specialized on encrypting short constant-length symmetric keys from some key space $\mathcal{K}$. Typically we have $\mathcal{K} = \{0,1\}^\kappa$ for $\kappa = 128$ or $\kappa = 256$. In this section we specify its syntax and security. We provide a construction in Sect. 9.

Definition 8. *A* key transport (KT) *scheme for a payload key space $\mathcal{K}$ consists of a secret key space $\mathcal{SK}$, a public key space $\mathcal{PK}$, a ciphertext space $\mathcal{C}$, an encryption handle space $\mathcal{HD}$, a key generation algorithm* gen $\to \mathcal{SK} \times \mathcal{PK}$*, and algorithms* enc, dec *as follows:*

$$\mathcal{PK} \times \mathcal{K} \to \mathrm{enc} \to \mathcal{C} \times \mathcal{HD} \qquad\qquad \mathcal{SK} \times \mathcal{C} \to \mathrm{dec} \to \mathcal{HD} \times \mathcal{K}$$

Intuitively, for correctness we demand that after $(sk, pk) \leftarrow \mathrm{gen}$ and $(c, hd) \leftarrow \mathrm{enc}(pk, k)$ and $(hd', k') \leftarrow \mathrm{dec}(sk, c')$ we have (1) *handle freshness:* the handle hd output by enc is unique (doesn't collide with other handles output by enc); and (2) *payload key recovery:* $hd' = hd \implies k' = k$.[10]

A formal version of these demands is covered by Def. 9. Our security definition is simulation based. In a nutshell, we say that a KT scheme is secure if there exists a simulator that behaves precisely like (read: indistinguishably from) the real scheme, just that it never sees the payload keys that it is meant to transport. If no adversary can tell apart whether it interacts with the real scheme or such a simulator, it also cannot learn information about the transported keys.

We start with defining the syntax of a simulator that fits the specification of Def. 8: A simulator for a KT scheme consists of a state space $\mathcal{ST}$ and algorithms

$$\mathcal{PK} \to \mathrm{sim_E}\langle\mathcal{ST}\rangle \to \mathcal{C} \times \mathcal{HD} \qquad\qquad \mathcal{SK} \times \mathcal{C} \to \mathrm{sim_D}\langle\mathcal{ST}\rangle \to \mathcal{HD} \times \mathcal{K}\ ,$$

where the $\langle\mathcal{ST}\rangle$ notation suggests that the algorithms are stateful with the common state space $\mathcal{ST}$.

Definition 9. *A KT scheme is correct and secure (against active adversaries) if there exists a simulator such that for every considerable adversary $\mathcal{A}$ the advantage function $\mathbf{Adv}^{\mathrm{ind}}(\mathcal{A}) := |\Pr[\mathbf{IND}^0(\mathcal{A})] - \Pr[\mathbf{IND}^1(\mathcal{A})]|$ is negligible, where the games are in Fig. 10.*

Note how lines 04,05 formalize *handle freshness* while lines 07,12,14,15 formalize the *payload key recovery* demand. (In the $b = 1$ case there is no payload key, hence the conditioning in line 14.) Lines 03,11,17 model that ciphertexts and handles and the payload keys of dishonestly generated ciphertexts are not considered secret but public information. As in our KEM/VKEM/KDFEM formalizations, while Def. 9, as is, specifies security against active adversaries, a strengthening to IND-CCA security can be achieved by activating the gray components in Fig. 10. (As before, for the results of this article, this will not be necessary.)

9 Key transport from KDFEMs

We demonstrate how an efficient key transport (KT) scheme can be derived from a KDFEM. The details of our construction are in Fig. 11. We prove that if the KDFEM is secure then so is the KT scheme.

Intuitively, our transform follows an encrypt-then-mac approach. The KT encryption algorithm invokes the KDFEM encapsulation algorithm once and the KDF evaluation algorithm twice. The first KDF evaluation creates a mask with which the transported key is one-time pad encrypted, and the second KDF evaluation is used to create a MAC tag for the resulting ciphertext. The KT decryption algorithm reverses this, and rejects all ciphertexts that have a wrong MAC tag.

[10] Analogously to Footnotes 6 and 7, it might be tempting to additionally require that $c' = c \implies hd' = hd$. However, as no part of our article logically depends on such a property, we once more abstain from *formally demanding* it.

```
INITIALIZATIONS: $A \leftarrow \emptyset$; $C[\cdot], K[\cdot] \leftarrow \diamond$; $st \leftarrow \diamond$

Game $\mathbf{IND}^b(\mathcal{A})$: $(sk, pk) \leftarrow \text{gen}$; $b' \leftarrow \mathcal{A}(pk)$; Stop with $b'$

Oracle Enc($k$)
00 If $b = 0$: $(c, hd) \leftarrow \text{enc}(pk, k)$
01 If $b = 1$: $(c, hd) \leftarrow \text{sim}_\text{E}\langle st\rangle(pk)$
02 $\text{Accept}_\text{E}(c, hd, k)$
03 Share $c, hd$

Oracle Dec($c$)
08 If $b = 0$: $(hd, k) \leftarrow \text{dec}(sk, c)$
09 If $b = 1$: $(hd, k) \leftarrow \text{sim}_\text{D}\langle st\rangle(sk, c)$
10 $\text{Accept}_\text{D}(c, hd, k)$
11 Share $hd$

Proc $\text{Accept}_\text{E}(c, hd, k)$
04 Promise $hd \notin A$
05 $A \overset{\cup}{\leftarrow} \{hd\}$
06 $C[hd] \leftarrow c$
07 $K[hd] \leftarrow k$

Proc $\text{Accept}_\text{D}(c, hd, k)$
12 If $hd \in A$:
13     Promise $C[hd] = c$
14     If $b = 0$:
15         Promise $K[hd] = k$
16 Else:
17     Share $k$
```

Fig. 10. Games $\mathbf{IND}^0, \mathbf{IND}^1$ as required by Def. 9. (By default ignore the gray components.) The adversary can query the Enc, Dec oracles. The $\text{Accept}_\text{E}, \text{Accept}_\text{D}$ procedures are invoked (exclusively) from lines 02,10.

Theorem 3. *Let* KDFEM $:= (\overline{\text{gen}}, \overline{\text{enc}}, \overline{\text{dec}}, \overline{\text{eval}})$ *be a KDFEM. Let* KT $:= (\text{gen}, \text{enc}, \text{dec})$ *be the KT scheme constructed from it according to Fig. 11. Then there exists a simulator for* KT *such that for all adversaries $\mathcal{A}$ attacking the security of the KT scheme there exists an adversary $\mathcal{B}$ attacking the security of the KDFEM such that*

$$\mathbf{Adv}^{\text{ind}}(\mathcal{A}) \le \mathbf{Adv}^{\text{ke–kdfem}}(\mathcal{B}) + \frac{q}{|\mathcal{K}| - q}$$

where q denotes the number of decryption queries that $\mathcal{A}$ is allowed to pose, and the security games are those of Def. 7 and Def. 9.

```
Proc gen
00 $(sk, pk) \leftarrow \overline{\text{gen}}$
01 Return $sk, pk$

Proc enc($pk, k$)
02 $(\bar{c}, \overline{hd}, st) \leftarrow \overline{\text{enc}}(pk)$
03 $\mu \leftarrow \overline{\text{eval}}(st, \diamond)$
04 $\bar{k} \leftarrow k + \mu$
05 $\tau \leftarrow \overline{\text{eval}}(st, \bar{k})$
06 $c := (\bar{c}, \bar{k}, \tau)$
07 $hd \leftarrow \overline{hd} \parallel \bar{k}$
08 Return $c, hd$

Proc dec($sk, c$)
09 $(\overline{hd}, st) \leftarrow \overline{\text{dec}}(sk, \bar{c})$
10 $\tau' \leftarrow \overline{\text{eval}}(st, \bar{k})$
11 if $\tau = \tau'$:
12     $\mu \leftarrow \overline{\text{eval}}(st, \diamond)$
13     $k \leftarrow \bar{k} - \mu$
14     $hd \leftarrow \overline{hd} \parallel \bar{k}$
15     Return $hd, k$
16 else: Abort
```

Fig. 11. Key transport built from KDFEM algorithms. The input key is masked by μ. A tag τ is generated for the masked key $\bar{k}$ in line 05. Line 16 aborts when the tag in the ciphertext is deemed unauthentic.

The proof is in Appendix D. In the following we provide some intuition. We first fix the simulator such that sim_E runs $\overline{\text{enc}}$ to establish the KDFEM ciphertext and handle, and then picks values $\bar{k}, \tau$ uniformly at random, while sim_D decrypts ciphertexts using the secret key except for authentic ciphertexts which it can recognize based on their KDFEM handle and the tabulated values $\bar{k}, \tau$.

Given this simulator, the reduction from KT security to KDFEM security is straighforward, as all KDFEM algorithm invocations can be replaced by corresponding oracle calls. The term $q/(|\mathcal{K}| - q)$ of the theorem statement comes from the encrypt-then-MAC design and covers adversaries that try to find valid MAC tags by guessing them. (With one attempt per decryption query, hence the factor q; note that set $\mathcal{K}$ coincides with the universe of MAC tag.)

10 NIST KEM Candidates

We demonstrate that the four NIST post-quantum KEM finalists (Crystals-Kyber[11] [22], Classic McEliece [4], SABER [12] and NTRU [10]) are almost (post-quantum secure) KDFEMs. More precisely, only mild tweaks are required to turn them into KDFEMs. Two challenges have to be resolved for this:

1. The NIST KEMs don't natively support handles. Our KDFEM interpretations need to introduce them, such that each enc invocation outputs a fresh handle, and such that any corresponding dec invocation recovers it.
2. The two KEM algorithms (encapsulation and decapsulation) need to be broken into three KDFEM algorithms (encapsulation, decapsulation, evaluation).

We address the first challenge by exploiting that the NIST KEMs are CCA secure so that we can simply use the ciphertexts as handles. More compact solutions for the handle may exist, for instance inspired by the approach of [14] that hashes an unpredictible part of the ciphertext. The second point is addressed by observing a common structure of the NIST KEMs that is illustrated in Fig. 12.

```
Proc enc($pk$)
  $(c, k^*, T) \leftarrow \mathrm{enc}^*(pk)$
  $k \leftarrow \mathrm{KDF}(k^*, T)$
  Return $c, k$

Proc dec($sk, c$)
  $(k^*, T) \leftarrow \mathrm{dec}^*(sk, c)$
  $k \leftarrow \mathrm{KDF}(k^*, T)$
  Return $k$
```

```
Proc $\overline{\mathrm{enc}}(pk)$
  $(c, k^*, T) \leftarrow \mathrm{enc}^*(pk)$
  $hd := c$
  $st := (k^*, T)$
  Return $c, hd, st$

Proc $\overline{\mathrm{dec}}(sk, c)$
  $(k^*, T) \leftarrow \mathrm{dec}^*(sk, c)$
  $hd := c$
  $st := (k^*, T)$
  Return $hd, st$

Proc $\overline{\mathrm{eval}}(st, L)$
  $k \leftarrow \mathrm{KDF}(k^*, T \parallel L)$
  Return $k$
```

Fig. 12. The left-hand side represents a high level abstraction of the encapsulation and decapsulation algorithms of all four NIST post-quantum candidates. Each of these algorithms can be seen as a succession of core steps (denoted with enc^* or dec^*) that output a pre-key k^*, some additional terms T, and a ciphertext in the case of enc^*. Both algorithms end with a key derivation step denoted with KDF. The right-hand side shows how we transform the KEMs into the KDFEM setting. Note that the KDF step is outsourced into a separate procedure, which adds the label input to the information in T. Note also that the KEM ciphertexts are used as handles.

In the remaining part of this section we provide the details of how the four NIST KEMs can be turned into KDFEMs. For concreteness we use the symbols from the documents provided by the KEMs' authors. While their notation differs from ours in many cases, the overall concepts remain sufficiently visible.

Crystals-Kyber. Considering page 10 of the specification document [22], we build the generation algorithm exactly as in Algorithm 7. The encapsulation algorithm $\overline{\mathrm{enc}}$ is constructed from lines 1–4 and returns $(c, hd, (\bar{K}, H(c)))$ where hd is actually c. The decapsulation $\overline{\mathrm{dec}}$ is the same except for line 8 that should now return $(hd, (\bar{K}', H(c)))$ and line 10 that should return $(hd, (z, H(c)))$. The $\overline{\mathrm{eval}}$ function is simply the KDF where the label is appended to the state.

Classic McEliece. We build $\overline{\mathrm{gen}}$ similarly as in page 9 of the specification document [4]. $\overline{\mathrm{enc}}$ is described as in lines 1 and 2 from the encapsulation section on page 10 and $\overline{\mathrm{dec}}$ as in lines 1–3 from the decapsulation section. Recall that in $\overline{\mathrm{enc}}$ and $\overline{\mathrm{dec}}$, the ciphertext is assigned to the handle. $\overline{\mathrm{eval}}$ computes the hash H of the state appended to the label.

Saber. $\overline{\mathrm{gen}}$ should be the same as the generation algorithm described in page 10 of the specification document [12]. $\overline{\mathrm{enc}}$ represents lines 1–3 of the encapsulation figure with the returned value being

[11] Crystals-Kyber has been selected as a winner by the NIST on July 5, 2022.

$(c, hd, (\mathcal{H}(c), \hat{K}))$. $\overline{\mathrm{dec}}$ is similar to the one presented in lines 1–7 of Algorithm 6 but with the exception that line 5 returns $(hd, (\mathcal{H}(c), \hat{K}'))$ and line 7 returns $(hd, (\mathcal{H}(c), z))$. Finally, $\overline{\mathrm{eval}}$ computes the hash $\mathcal{H}$ of the state appended to the label.

NTRU. This case is very similar to the previous ones: $\overline{\mathrm{gen}}$ is described similarly as in section 1.12.1 [10], $\overline{\mathrm{enc}}$ is set to execute lines 1, 2, 3 and 5 of section 1.12.2 with the handle being the ciphertext. $\overline{\mathrm{enc}}$ returns the tuple $(\text{packed\_ciphertext}, hd, st)$ where $st := \mathsf{bytes\_to\_bits}(packed\_rm, 8 \cdot \mathsf{dpke\_plaintext\_bytes})$. $\overline{\mathrm{dec}}$ shall execute lines 1, 2, 4 and 5 but with the output being (hd, st) if $fail = 0$, and (hd, st') otherwise where $st' := \mathsf{bytes\_to\_bits}(prf\_key, \mathsf{prf\_key\_bits}) \parallel \mathsf{bytes\_to\_bits}(packed\_ciphertext, 8 \cdot \mathsf{kem\_ciphertext\_bytes})$. $\overline{\mathrm{eval}}$ is now simply executing the function Hash over the state concatenated with the label.

11 Conclusion

The current efforts by NIST and other bodies to standardize quantum-resilient KEMs have a huge impact on the next decades of practical cryptography. This is not only because the new schemes have the potential to protect us from possible future threats, but also because of the conceptual change of considering KEMs instead of PKE schemes as the more fundamental building block. (Prior confidentiality standards like OAEP and IES and ECIES tended to formalize PKE, not KEM; this is now reversed.) It is of utmost importance to get this PKE → KEM transition right: History has shown that any detail that can be misunderstood by practitioners might be gotten wrong eventually, with severe security issues as a consequence.

While cryptographic theory has found the classic KEM concept to be the most versatile abstraction, practical needs suggest that KEMs should be a little stronger than theory assumes. Our research explores two avenues to provide such a strengthening. We test our newly proposed primitives, VKEM and KDFEM, with benchmarks in the important domains of KEM combiners and key transport. We found in particular the KDFEM approach promising, as (1) the concept is simple and the constructions of combiners and key transports are immediate; and (2) all four NIST finalist KEMs require only minimal modifications to meet our KDFEM syntax and security. We hope that our work helps informing future standardization efforts.

Acknowledgments

Many valuable comments of anonymous SSR'23 reviewers helped improving this article. We also received constructive feedback from Christian Stohrer. This research was partially funded by armasuisse Science and Technology (Project Nr. CYD-C-2020010).

References

1. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. Tech. rep., NIST (November 2016), https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf
2. CYBER; Quantum-safe Hybrid Key Exchanges. Technical Specification TS 103 744, ETSI (December 2020), https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf
3. Abe, M., Gennaro, R., Kurosawa, K., Shoup, V.: Tag-KEM/DEM: A new framework for hybrid encryption and a new analysis of Kurosawa-Desmedt KEM. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 128–146. Springer, Heidelberg (May 2005). https://doi.org/10.1007/11426639_8
4. Albrecht, M.R., Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., Wang, W.: Classic McEliece. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
5. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (Dec 2000). https://doi.org/10.1007/3-540-44448-3_41
6. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_11

7. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_21
8. Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., Stebila, D.: Hybrid key encapsulation mechanisms and authenticated key exchange. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019. pp. 206–226. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-25510-7_12
9. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R.: OpenPGP Message Format. RFC 4880, RFC Editor (November 2007). https://doi.org/10.17487/RFC4880, https://www.rfc-editor.org/info/rfc4880
10. Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z., Saito, T., Yamakawa, T., Xagawa, K.: NTRU. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
11. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM Journal on Computing **33**(1), 167–226 (2003)
12. D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F., Mera, J.M.B., Beirendonck, M.V., Basso, A.: SABER. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
13. Dodis, Y., Katz, J.: Chosen-ciphertext security of multiple encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 188–209. Springer, Heidelberg (Feb 2005). https://doi.org/10.1007/978-3-540-30576-7_11
14. Duman, J., Hövelmanns, K., Kiltz, E., Lyubashevsky, V., Seiler, G.: Faster lattice-based KEMs via a generic fujisaki-okamoto transform using prefix hashing. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2722–2737. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484819
15. Giacon, F., Heuer, F., Poettering, B.: KEM combiners. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 190–218. Springer, Heidelberg (Mar 2018). https://doi.org/10.1007/978-3-319-76578-5_7
16. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7_34
17. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. Journal of Cryptology **24**(3), 588–613 (Jul 2011). https://doi.org/10.1007/s00145-010-9073-y
18. McGrew, D.: An Interface and Algorithms for Authenticated Encryption. RFC 5116, RFC Editor (January 2008). https://doi.org/10.17487/RFC5116, https://www.rfc-editor.org/info/rfc5116
19. Pinto, A., Poettering, B., Schuldt, J.C.N.: Multi-recipient encryption, revisited. In: Moriai, S., Jaeger, T., Sakurai, K. (eds.) ASIACCS 14. pp. 229–238. ACM Press (Jun 2014)
20. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 2002. pp. 98–107. ACM Press (Nov 2002). https://doi.org/10.1145/586110.586125
21. Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer, Heidelberg (Feb 2004). https://doi.org/10.1007/978-3-540-25937-4_22
22. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
23. Shoup, V.: A Proposal for an ISO Standard for Public Key Encryption. Tech. Rep. Version 2.1, IBM Zurich Research Lab (December 2001), https://www.shoup.net/papers/iso-2_1.pdf
24. Zhang, R., Hanaoka, G., Shikata, J., Imai, H.: On the security of multiple encryption or CCA-security+CCA-security=CCA-security? In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 360–374. Springer, Heidelberg (Mar 2004). https://doi.org/10.1007/978-3-540-24632-9_26

A Constructing a VKEM

We show that VKEMs can be efficiently constructed from KEMs in combination with PRFs. In Fig. 13 we propose a construction that provides both key encapsulation security Def. 4 and label binding Def. 5. Notice that, in this construction, the ciphertext c_2 is empty. The label binding is straightforward as hd_2 is generated by computing a PRF on $L_1 \shortparallel hd \shortparallel L_2$. The KE security of the VKEM is implied by the KE security of the underlying KEM and the pseudo-randomness of the PRF. All four algorithms make use of the same PRF instance that generates the session key.

```
Proc enc_1(pk, L_1)
    $(c_1, hd, k) \leftarrow \mathrm{encap}(pk)$
    $(k_1, \_) \leftarrow \mathrm{PRF}(k, L_1)$
    $hd_1 := L_1 \shortparallel hd$
    $st := (k, hd_1)$
    Return $c_1, hd_1, k_1, st$

Proc enc_2(st, L_2)
    $(k_2, hd_2) \leftarrow \mathrm{PRF}(k, hd_1 \shortparallel L_2)$
    Return $\diamond, hd_2, k_2$

Proc dec_1(sk, L_1, c_1)
    $(hd, k) \leftarrow \mathrm{decap}(sk, c_1)$
    $k_1 \leftarrow \mathrm{PRF}(k, L_1)$
    $hd_1 := L_1 \shortparallel hd$
    $st := (k, hd_1)$
    Return $hd_1, k_1, st$

Proc dec_2(st, L_2, c_2)
    $(k_2, hd_2) \leftarrow \mathrm{PRF}(k, hd_1 \shortparallel L_2)$
    Return $hd_2, k_2$
```

Fig. 13. A label binding VKEM construction from a KEM and a PRF. We omit gen as it simply consists of running the KEM generation algorithm.

B Proof of Theorem 1

```
INITIALIZATIONS: $\mathrm{K}[\cdot], \mathrm{C}[\cdot] \leftarrow \diamond$; $\mathrm{A}, \mathrm{X}, \mathrm{F} \leftarrow \emptyset$; $\mathrm{R}[\cdot] \leftarrow \$(\mathcal{K})$
                 $\mathrm{H}[\cdot] \leftarrow \diamond$; $\mathrm{ST_E}[\cdot], \mathrm{ST_D}[\cdot] \leftarrow \triangleright$; $\mathrm{H_E}[\cdot], \mathrm{C_E}[\cdot], \mathrm{H_D}[\cdot], \mathrm{C_D}[\cdot] \leftarrow \epsilon$;

Game $\mathbf{KE}_0^b(\mathcal{A})$
00  $(sk^1, pk^1) \leftarrow \mathrm{gen}^1$
01  $(sk^2, pk^2) \leftarrow \mathrm{gen}^2$
02  $b' \leftarrow \mathcal{A}(pk^1, pk^2)$
03  Require $\mathrm{X} \cap \mathrm{F} = \emptyset$
04  Stop with $b'$

Oracle Reveal(hd)
05  Require $hd \in \mathrm{A}$
06  $\mathrm{X} \xleftarrow{\cup} \{hd\}$
07  $k \leftarrow \mathrm{K}[hd]$
08  Return $k$

Oracle Challenge(hd)
09  Require $hd \in \mathrm{A}$
10  $\mathrm{F} \xleftarrow{\cup} \{hd\}$
11  $k \leftarrow b \,?\, \mathrm{K}[hd] : \mathrm{R}[hd]$
12  Return $k$

Oracle Enc()
13  $(c_1^1, hd_1^1, \_, st^1) \leftarrow \mathrm{enc}_1^1(pk^1, \diamond)$
14  $(c_1^2, hd_1^2, \_, st^2) \leftarrow \mathrm{enc}_1^2(pk^2, \diamond)$
15  $(c_2^1, hd_2^1, k_2^1) \leftarrow \mathrm{enc}_2^1(st^1, hd_1^2)$
16  $(c_2^2, hd_2^2, k_2^2) \leftarrow \mathrm{enc}_2^2(st^2, hd_1^1)$
17  $c := (c_1^1, c_2^1, c_1^2, c_2^2)$
18  $hd \leftarrow (hd_2^1, hd_2^2)$
19  $k \leftarrow k_2^1 + k_2^2$
20  $\mathrm{Accept_E}(c, hd, k)$
21  Share $c, hd$

Oracle Dec(c)
26  $(hd_1^1, \_, st^1) \leftarrow \mathrm{dec}_1^1(sk^1, \diamond, c_1^1)$
27  $(hd_1^2, \_, st^2) \leftarrow \mathrm{dec}_1^2(sk^2, \diamond, c_1^2)$
28  $(hd_2^1, k_2^1) \leftarrow \mathrm{dec}_2^1(st^1, hd_1^2, c_2^1)$
29  $(hd_2^2, k_2^2) \leftarrow \mathrm{dec}_2^2(st^2, hd_1^1, c_2^2)$
31  $hd \leftarrow (hd_2^1, hd_2^2)$
32  $k \leftarrow k_2^1 + k_2^2$
33  $\mathrm{Accept_D}(c, hd, k)$
34  Share $hd$

Proc Accept_E(c, hd, k)
22  Promise $hd \notin \mathrm{A}$
23  $\mathrm{A} \xleftarrow{\cup} \{hd\}$
24  $\mathrm{C}[hd] \leftarrow c$
25  $\mathrm{K}[hd] \leftarrow k$

Proc Accept_D(c, hd, k)
35  If $hd \in \mathrm{A}$:
36      Promise $\mathrm{C}[hd] = c$
37      Promise $\mathrm{K}[hd] = k$
38  Else:
39      Share $k$
```

Fig. 14. Games $\mathbf{KE}_0$ and $\mathbf{KE}_1$ in a single figure. Adding the statement ‘Promise label-binding of VKEM^2’ in line 30 switches from $\mathbf{KE}_0$ to $\mathbf{KE}_1$.

Proof. We prove Theorem 1 when **KE** is instantiated with VKEM^1 and **LB** is instantiated with VKEM^2. The alternative case can be proven using symmetric arguments.
We write $\mathbf{KE}_0$ in Fig. 14 as the instantiation of **KE** with the inlined C. The following equality is straightforward:

$$\mathbf{Adv}^{\text{ke-kem}}(\mathcal{A}) = \mathbf{Adv}^{\mathbf{KE}_0}(\mathcal{A})$$

We modify $\mathbf{KE}_0$ into $\mathbf{KE}_1$ by adding a Promise statement in line 30 reflecting Fig. 6:

$$\text{Promise label-binding of VKEM}^2$$

Inspection shows the following inequality:

$$\mathbf{Adv}^{\mathbf{KE}_0}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{KE}_1}(\mathcal{A}) \ .$$

Letting E be the event that $\mathbf{KE}_1$ aborts on line 30, the next relation holds:

$$\begin{aligned}\mathbf{Adv}^{\mathbf{KE}_1}(\mathcal{A}) &\coloneqq |\Pr[\mathbf{KE}_1^0(\mathcal{A})] - \Pr[\mathbf{KE}_1^1(\mathcal{A})]| \\ &\leq |\Pr[\mathbf{KE}_1^0(\mathcal{A}) \wedge \neg E] - \Pr[\mathbf{KE}_1^1(\mathcal{A}) \wedge \neg E]| + |\Pr[\mathbf{KE}_1^0(\mathcal{A}) \wedge E] - \Pr[\mathbf{KE}_1^1(\mathcal{A}) \wedge E]|\end{aligned}$$

Proving the inequality of Theorem 1 boils down to showing that there exist $\mathcal{B}$ and $\mathcal{C}$ such that:

$$\begin{aligned}&\mathbf{(1)}\ |\Pr[\mathbf{KE}^0(\mathcal{A}) \wedge \neg E] - \Pr[\mathbf{KE}^1(\mathcal{A}) \wedge \neg E]| = \mathbf{Adv}^{\text{ke-vkem}}(\mathcal{B}) \quad \text{and} \\ &\mathbf{(2)}\ |\Pr[\mathbf{KE}_1^0(\mathcal{A}) \wedge E] - \Pr[\mathbf{KE}_1^1(\mathcal{A}) \wedge E]| \leq \mathbf{Adv}^{\text{lb}}(\mathcal{C}) \ .\end{aligned}$$

We now prove relation **(1)**. Let $\mathbf{KE}_2$ (Fig. 14) be the key establishement game **KE** of VKEM^1. Let $\mathcal{B}$ be the adversary constructed from $\mathcal{A}$ and described in Fig. 15. We prove that $\mathcal{B}$ is a valid reduction from $\mathbf{KE}_1$ to $\mathbf{KE}_2$. More formally, we show that the Promise, Require and Share statements of $\mathbf{KE}_1$ are properly simulated in $\mathbf{KE}_2(\mathcal{B})$.

```
Proc $\mathcal{B}(pk^1)$
00 $\mathrm{A} \leftarrow \emptyset$
01 $\bar{\mathrm{K}}[\cdot] \leftarrow \diamond$
02 $(sk^2, pk^2) \leftarrow \mathrm{gen}^2$
03 $b' \leftarrow \mathcal{A}(pk^1, pk^2)$
04 Return $b'$

Oracle Enc()
05 Pick fresh $sid$
06 $(c_1^1, hd_1^1) \leftarrow \mathrm{Enc}_1(sid, \diamond)$
07 $(c_1^2, hd_1^2, \_, st^2) \leftarrow \mathrm{enc}_1^2(pk^2, \diamond)$
08 $(c_2^1, hd_2^1) \leftarrow \mathrm{Enc}_2(sid, hd_1^2)$
09 $(c_2^2, hd_2^2, k_2^2) \leftarrow \mathrm{enc}_2^2(st^2, hd_1^1)$
10 $\bar{\mathrm{K}}[hd_2^2] \leftarrow k_2^2$
11 $c \leftarrow (c_1^1, c_2^1, c_1^2, c_2^2)$
12 $hd \leftarrow (hd_2^1, hd_2^2)$
13 $\mathrm{A} \overset{\cup}{\leftarrow} \{hd\}$
14 Share $c, hd$

Oracle Reveal($hd$)
15 Require $hd \in \mathrm{A}$
16 $(hd_2^1, hd_2^2) \leftarrow hd$
17 $k_2^1 \leftarrow \mathrm{Reveal}(hd_2^1)$
18 $k_2^2 \leftarrow \bar{\mathrm{K}}[hd_2^2]$
19 Share $k_2^1 + k_2^2$

Oracle Dec($c$)
20 Pick fresh $sid$
21 $(c_1^1, c_2^1, c_1^2, c_2^2) \leftarrow c$
22 $hd_1^1 \leftarrow \mathrm{Dec}_1(sid, \diamond, c_1^1)$
23 $(hd_1^2, \_, st^2) \leftarrow \mathrm{dec}_1^2(sk^2, \diamond, c_1^2)$
24 $(hd_2^1, k_2^1) \leftarrow \mathrm{Dec}_2(sid, hd_1^2, c_2^1)$
25 $(hd_2^2, k_2^2) \leftarrow \mathrm{dec}_2^2(st^2, hd_1^1, c_2^2)$
26 Require label-binding VKEM$^2$
27 $hd \leftarrow (hd_2^1, hd_2^2)$
28 Share $hd$
29 if $hd \notin \mathrm{A}$:
30     Share $k_2^1 + k_2^2$

Oracle Challenge($hd$)
31 Require $hd \in \mathrm{A}$
32 $(hd_2^1, hd_2^2) \leftarrow hd$
33 $k_2^1 \leftarrow \mathrm{Challenge}(hd_2^1)$
34 $k_2^2 \leftarrow \bar{\mathrm{K}}[hd_2^2]$
35 Share $k_2^1 + k_2^2$
```

Fig. 15. Reduction from game $\mathbf{KE}_1$ to $\mathbf{KE}_2$. $\mathcal{B}$ runs $\mathcal{A}$ and provides a compatible interface to access the simulated combiner. The statements in violet correspond to $\mathcal{B}$ calling the VKEM^2 oracles from Fig. 5. Notice that the color matches Fig. 1. The Require statements in lines 15, 26 and 31 abort the whole game (including $\mathcal{B}$) when the respective conditions are not met. Promising label-binding is now Require LB.

First, notice that picking fresh sid in Fig. 15 (lines 05 and 20) prevents the simulation from aborting on lines 20, 27, 40 and 47 in Fig. 5. For simplicity, we denote all the variables, arrays and sets of Fig. 5 with a superindex (1 or 2), depending on which VKEM we are discussing (VKEM^1 or VKEM^2). For example, A^1 now corresponds to the set A when intantiating Fig. 5 with VKEM^1.
The following arguments match the Promise statements in the simulation to those in $\mathbf{KE}_1$:

- If hd_2^1 is fresh (line 35 Fig. 5), then so is hd (line 22 Fig. 14). Formally,

$$hd_2^1 \notin \mathrm{A}^1 \Rightarrow (hd_2^1, hd_2^2) \notin \mathrm{A} \Rightarrow hd \notin \mathrm{A}$$

- Assuming the handles are honest (line 35 Fig. 14), if both VKEMs decrypt to the correct key (line 58 Fig. 5), then so does the overall simulation (line 37 Fig. 14). Formally speaking, if $hd \in \mathrm{A}$, then

$$hd_2^1 \in \mathrm{A}^1 \Rightarrow k_2^1 = \mathrm{K}^1[hd_2^1] \quad \text{and} \quad hd_2^2 \in \mathrm{A}^2 \Rightarrow k_2^2 = \mathrm{K}^2[hd_2^2]$$

  thus:

$$k = k_2^1 + k_2^2 = \mathrm{K}[hd] \ .$$

- A similar proof applies to line 57 of Fig. 5 and line 36 of Fig. 3.
- Promising label-binding of VKEM^2 in line 30 of Fig. 14 is assumed to never abort (event E does not occur) and thus line 30 can be simulated in a require line 26.

We prove that the Require statements are also simulated properly:

- If for a certain query, no abort occurs on lines 05 and 09 of Fig. 14 (hd is honest during the challenge and reveal calls), then for the same query, no abort occurs on lines 15 and 31 of Fig. 15 (and vice versa).
- If $\mathbf{KE}_2$ does not abort when testing if the adversary queried the challenge and reveal oracle on the same handle (02 in Fig. 2), then line 03 in Fig. 14 does not abort.
  In order to prove this assertion, we first argue that the set A can only contain pairs of which both elements are fresh. In fact, applying the handle freshness property (line 35 Fig. 5) on both successive lines 08 and 09 of Fig. 15 imply that:

$$\nexists \{(hd_2^1, hd_2^2), (\overline{hd}_2^1, hd_2^2)\} \subseteq \mathrm{A} \quad \text{s.t.} \quad hd_2^1 \neq \overline{hd}_2^1 \ .$$

  Second, we notice that lines 15 and 31 from Fig. 15 require that $(hd_2^1, hd_2^2) \in \mathrm{A}$.
  Third, having $\mathcal{A}$ querying Challenge and Reveal on two pairs (hd_2^1, hd_2^2) and $(hd_2^1, \overline{hd}_2^2)$, the game aborts on line 02 of Fig. 2.
  Putting the pieces together, if line 03 from Fig. 14 aborts in $\mathbf{KE}_1$, then one of the require lines 02 (Fig. 2), 15 or 31 (Fig. 15) must abort in the simulation.

The Share statements proofs follow.

- Sharing c, hd in line 21 Fig. 14 is reproduced in the simulation on line 14 Fig. 15.
- Similarly, sharing hd in line 34 Fig. 14 is reproduced in 28 of Fig. 15.
- k_2^1 and k_2^2 are computed analogously in both Fig. 15 and Fig. 14. The key shared in line 19 (resp. 35) Fig. 15 has the same probability distribution as the one shared in lines 08 (resp. 12) Fig. 14. Notice that when $b = 0$, line 33 Fig. 15 outputs a uniformly random key k_2^2 making $k_2^1 + k_2^2$ uniformly random.
- Assuming $hd \notin \mathrm{A}$, then key k shared in line 39 Fig. 14 is of the same distribution as that in line 28 Fig. 15. Assuming the schemes do not abort on correctness conditions, we can present the following arguments:
  If $hd_2^1 \notin \mathrm{A}^1$ then Dec_2 of line 24 Fig. 15 will output a key k_2^1 thus sharing $k_2^1 + k_2^2$ is possible.
  If $hd_2^1 \in \mathrm{A}^1$ for a certain session sid, then we show that $hd_2^2 \in \mathrm{A}^2$ contradicting the initial assumption $hd \notin \mathrm{A}$. First, notice that if $hd_2^1 \in \mathrm{A}^1$, this means that Enc was called and several handles were generated honestly. More Formally:

$$\begin{aligned} \exists \overline{sid}, \overline{hd}_1^1, \overline{hd}_1^2, \overline{hd}_2^2 \ \text{ s.t. } (hd_2^1, \overline{hd}_2^2) \in \mathrm{A} \text{ (line 13 Fig. 15)} \\ \diamond \| \overline{hd}_1^1 \| \overline{hd}_1^2 \| hd_2^1 = \mathrm{H}^1[\overline{sid}] \text{ (VKEM}^1 \text{ line 37 Fig. 5)} \\ \diamond \| \overline{hd}_1^2 \| \overline{hd}_1^1 \| \overline{hd}_2^2 = \mathrm{H}^2[\overline{sid}] \text{ (VKEM}^2 \text{ line 37 Fig. 5)} \end{aligned}$$

  Let $\diamond \| hd_1^1 \| hd_1^2 \| hd_2^1 := \mathrm{H}_\mathrm{D}^1[sid]$ and $\diamond \| hd_1^2 \| hd_1^1 \| hd_2^2 := \mathrm{H}_\mathrm{D}^2[sid]$ be the decryption histories in session sid. Using the perfect correctness definition of line 56 of Fig. 5, we know that

$$\begin{aligned} hd_2^1 \in \mathrm{A}^1 &\Rightarrow \mathrm{H}_\mathrm{D}^1[sid] = \mathrm{H}^1[\overline{sid}] \\ &\Rightarrow \diamond \| hd_1^1 \| hd_1^2 = \diamond \| \overline{hd}_1^1 \| \overline{hd}_1^2 \\ &\Rightarrow hd_1^1 = \overline{hd}_1^1 \wedge hd_1^2 = \overline{hd}_1^2 \\ &\Rightarrow \diamond \| \overline{hd}_1^2 \| \overline{hd}_1^1 \| hd_2^2 = \mathrm{H}_\mathrm{D}^2[sid] \end{aligned}$$

Requiring label-binding in line 26 Fig. 15 means that

$$\diamond \mathbin{\|} \overline{hd}_1^2 \mathbin{\|} \overline{hd}_1^1 \mathbin{\|} hd_2^2 = \mathrm{H}_{\mathrm{D}}^2[sid] \quad \text{and} \quad \diamond \mathbin{\|} \overline{hd}_1^2 \mathbin{\|} \overline{hd}_1^1 \mathbin{\|} \overline{hd}_2^2 = \mathrm{H}^2[sid]$$

imply that $\overline{hd}_2^2 = hd_2^2$. Finally, we have:

$$hd = (hd_2^1, \overline{hd}_2^2) = (hd_2^1, \overline{hd}_2^2) \in \mathrm{A}$$

which contradicts the assumption $hd \notin \mathrm{A}$.

The above shows that the simulation is valid concluding the first part of the proof.

We now suppose that the event E occurs and prove relation **(2)**. We show that there exists an adversary $\mathcal{C}$ constructed from $\mathcal{A}$ in Fig. 16, such that for any $b \in \{0,1\}$, $\Pr[\mathbf{KE}_1^b(\mathcal{A}) \wedge E] \leq \mathbf{Adv}^{\mathrm{lb}}(\mathcal{C})$. In fact, $\mathcal{C}$ runs $\mathcal{A}$ and deals with the queries from $\mathcal{A}$ similarly as the previous adversary $\mathcal{B}$ with only two main exceptions:

1. The oracle Dec can reveal the key k_2^2 if VKEM^2 decryption oracle did not share a key.
2. The challenge oracle always reveals the combined key k.

The first exception does not affect the validity of the simulation (aborting on $\mathrm{X} \cap \mathrm{F} \neq \emptyset$) because we assume that the game already aborts on E (that occurs before promising $\mathrm{X} \cap \mathrm{F} = \emptyset$). The second exception only strengthens the adversary $\mathcal{A}$ as the latter might have more information about the generated keys. Let q_d, q_c and q_r be the number of queries that $\mathcal{A}$ is allowed to make respectively to Dec, Challenge and Reveal oracles. In order to simulate properly, $\mathcal{C}$ should be allowed to make at most $q_r + q_d + q_c$ queries to the reveal oracle. With this we conclude that:

$$\forall b \in \{0,1\},\ \Pr[\mathbf{KE}_1^b(\mathcal{A}) \wedge E] \leq \Pr[\mathbf{LB}(\mathcal{C})] = \mathbf{Adv}^{\mathrm{lb}}(\mathcal{C})$$

and thus

$$|\Pr[\mathbf{KE}_1^0(\mathcal{A}) \wedge E] - \Pr[\mathbf{KE}_1^1(\mathcal{A}) \wedge E]| \leq \mathbf{Adv}^{\mathrm{lb}}(\mathcal{C})\ .$$

```
Proc $\mathcal{C}(pk^2)$
  $/\!\!/ \mathrm{A} \leftarrow \emptyset$
  $\bar{\mathrm{K}}[\cdot] \leftarrow \diamond$
  $(sk^1, pk^1) \leftarrow \mathrm{gen}^1$
  $\mathcal{A}(pk^1, pk^2)$

Oracle Enc()
  Pick fresh $sid$
  $(c_1^1, hd_1^1, \_, st^1) \leftarrow \mathrm{enc}_1^1(pk^1, \diamond)$
  $(c_1^2, hd_1^2) \leftarrow \mathrm{Enc}_1(sid, \diamond)$
  $(c_2^1, hd_2^1, , k_2^1) \leftarrow \mathrm{enc}_2^1(st^1, hd_1^2)$
  $(c_2^2, hd_2^2) \leftarrow \mathrm{Enc}_2(sid, hd_1^1)$
  $\bar{\mathrm{K}}[hd_2^1] \leftarrow k_2^1$
  $c \leftarrow (c_1^1, c_2^1, c_1^2, c_2^2)$
  $hd \leftarrow (hd_2^1, hd_2^2)$
  $/\!\!/ \mathrm{A} \xleftarrow{\cup} \{hd\}$
  Share $c, hd$

Oracle Reveal($hd$)
  $/\!\!/$ Require $hd \in \mathrm{A}$
  $(hd_2^1, hd_2^2) \leftarrow hd$
  $k_2^1 \leftarrow \bar{\mathrm{K}}[hd_2^1]$
  $k_2^2 \leftarrow \mathrm{Reveal}(hd_2^2)$
  Share $k_2^1 + k_2^2$

Oracle Dec($c$)
  Pick fresh $sid$
  $(c_1^1, c_2^1, c_1^2, c_2^2) \leftarrow c$
  $(hd_1^1, \_, st^1) \leftarrow \mathrm{dec}_1^2(sk^1, \diamond, c_1^1)$
  $hd_1^2 \leftarrow \mathrm{Dec}_1(sid, \diamond, c_1^2)$
  $(hd_2^1, k_2^1) \leftarrow \mathrm{dec}_2^2(st^1, hd_1^2, c_2^1)$
  $(hd_2^2, k_2^2) \leftarrow \mathrm{Dec}_2(sid, hd_1^1, c_2^2)$
  $hd \leftarrow (hd_2^1, hd_2^2)$
  Share $hd$
  if $hd \notin \mathrm{A}$:
    if $k_2^2 = \diamond$: $k_2^2 \leftarrow \mathrm{Reveal}(hd_2^2)$
    Share $k_2^1 + k_2^2$

Oracle Challenge($hd$)
  Same as Reveal
```

Fig. 16. The commented statements are the ones related to the set A. These statements can be omited since we know that the Promise label binding event is the only event that can be triggered. The statements in cyan correspond to $\mathcal{C}$ calling the VKEM^1 oracles from Fig. 5. Notice the color matching the one in Fig. 1.

C Proof of Theorem 2

Proof. Without loss of generality, we prove Theorem 2 when **KE** is instantiated with KDFEM^1. We denote $\mathbf{KE}_0$ the key establishment game of the inlined C and $\mathbf{KE}_1$ the key establishment game of

```
Proc $\mathcal{B}(pk^1)$
00 $\mathrm{A}^{-} \leftarrow \emptyset$
01 $\bar{\mathrm{K}}[\cdot] \leftarrow \diamond$
02 $(sk^2, pk^2) \leftarrow \mathrm{gen}^2$
03 $b' \leftarrow \mathcal{A}(pk^1, pk^2)$
04 Return $b'$
```

```
Oracle Enc()
05 Pick fresh $sid$
06 $(c^1, hd^1) \leftarrow \mathrm{Enc}(sid)$
07 $(c^2, hd^2, st^2) \leftarrow \mathrm{enc}^2(pk^2)$
08 $\mathrm{Eval}_{\mathrm{E}}(sid, hd^2)$
09 $k^2 \leftarrow \mathrm{eval}^2(st^2, hd^1)$
10 $c \leftarrow (c^1, c^2)$
11 $hd \leftarrow (hd^1, hd^2)$
12 $\bar{\mathrm{K}}[hd] \leftarrow k^2$
13 $\mathrm{A} \overset{\cup}{\leftarrow} \{hd\}$
14 Share $c, hd$
```

```
Oracle Reveal($hd$)
15 Require $hd \in \mathrm{A}^{-}$
16 $(hd^1, hd^2) \leftarrow hd$
17 $k^1 \leftarrow \mathrm{Reveal}(hd^1 \mathbin{\|} hd^2)$
18 $k^2 \leftarrow \bar{\mathrm{K}}[hd]$
19 Share $k^1 + k^2$
```

```
Oracle Dec($c$)
20 Pick fresh $sid$
21 $(c^1, c^2) \leftarrow c$
22 $hd^1 \leftarrow \mathrm{Dec}(sid, c^1)$
23 $(hd^2, st^2) \leftarrow \mathrm{dec}^2(sk^2, c^2)$
24 $k^1 \leftarrow \mathrm{Eval}_{\mathrm{D}}(sid, hd^2)$
25 $k^2 \leftarrow \mathrm{eval}^2(st^2, hd^1)$
26 $hd \leftarrow (hd^1, hd^2)$
27 Share $hd$
28 if $hd \notin \mathrm{A}^{-}$:
29     if $k^1 = \diamond$:
30         $k^1 \leftarrow \mathrm{Reveal}(hd^1 \mathbin{\|} hd^2)$
31     Share $k^1 + k^2$
```

```
Oracle Challenge($hd$)
32 Require $hd \in \mathrm{A}^{-}$
33 $(hd^1, hd^2) \leftarrow hd$
34 $k^1 \leftarrow \mathrm{Challenge}(hd^1 \mathbin{\|} hd^2)$
35 $k^2 \leftarrow \bar{\mathrm{K}}[hd]$
36 Share $k^1 + k^2$
```

Fig. 17. The KDFEM adversary $\mathcal{B}$ is very similar to the same adversary from the previous section. The colored lines correspond to $\mathcal{B}$ calling the KDFEM oracles. Notice that a Reveal query is happening in the decryption in order to recover the key.

KDFEM^1. We associate $\mathcal{A}$ to $\mathbf{KE}_0$ and $\mathcal{B}$ to $\mathbf{KE}_1$. We construct $\mathcal{B}$ from $\mathcal{A}$ in Fig. 17 and prove that $\mathcal{B}$ simulates properly the combined KDFEM instances C. First, picking fresh sid in Fig. 17 (lines 05 and 20) prevents the simulation from aborting on lines 20, 28, 39 and 46 in Fig. 8. For simplicity, we denote all the variables, arrays and sets of Fig. 8 with a superindex 1 or 2, to refer respectively to either KDFEM^1 or KDFEM^2. Similarly to the proof in Appendix A, we prove that the Promise statements in $\mathbf{KE}_1$ match those in $\mathbf{KE}_0$. In fact, promising handle freshness of KDFEM^1 (line 34 Fig. 8) implies freshness of the pair of handles of the combined instances (line 23 Fig. 3). Similarly, promising correctness of line 31 (resp. 32) Fig. 3 is ensured by the correctness of both KDFEM schemes in line 43 (resp. 57) Fig. 8. Requiring honesty of the handles pair in line 04 (resp. 08) Fig. 2 is properly simulated on line 15 (resp. 32) of Fig. 17. Additionally, analogous arguments used in Sect. 5 can be applied in this case to show that line 22 (resp. 29) of Fig. 3 matches line 14 Fig. 17 (resp. 27). Left to prove that, if $hd \notin \mathrm{A}$, then the simulation is able to share a (valid) key, and if $\mathcal{A}$ does not make challenge and reveal queries on the same handle pair, then the simulation does not abort.

For the first statement, we assume $hd \notin \mathrm{A}$ as in line 30 of Fig. 8 which implies $hd \notin \mathrm{A}^{-}$ line 23 Fig. 17. Supposing $hd^1 \notin \mathrm{A}^{-1}$, then by line 59 of Fig. 8 k^1 is shared which makes it possible to share $k^1 + k^2$ in line 31 of Fig. 17. Supposing $hd^1 \in \mathrm{A}^{-1}$, we have

$$\begin{aligned} hd^1 \in \mathrm{A}^{-1} &\Rightarrow L^1 \coloneqq hd^2 \notin \mathrm{A}^{-2} \\ &\Rightarrow hd^1 \mathbin{\|} L^1 \notin \mathrm{A}^1 \\ &\Rightarrow \mathrm{A}^1 \overset{\cup}{\leftarrow} \{hd^1 \mathbin{\|} L^1\} \quad \text{line 56 Fig. 8} \\ &\quad \wedge \mathrm{K}^1[hd^1 \mathbin{\|} L^1] \leftarrow k^1 \quad \text{line 57 Fig. 8} \end{aligned}$$

thus line 30 of Fig. 17 does not abort and outputs a key k^1.

For the second statement, notice that Reveal$(hd^1 \mathbin{\|} hd^2)$ (line 30 Fig. 17) is called only if $(hd^1, hd^2) \notin \mathrm{A}^{-}$, whereas in lines 17 and 34 Fig. 17 both Reveal$(hd^1 \mathbin{\|} hd^2)$ and Challenge$(hd^1 \mathbin{\|} hd^2)$ can only be called if $(hd^1, hd^2) \in \mathrm{A}^{-}$. Using similar arguments to those used in the previous proof, we have:

$$\nexists\{(hd^1, hd^2), (\overline{hd}^1, hd^2)\} \subseteq \mathrm{A}^{-} \quad \text{s.t.} \quad hd^1 \neq \overline{hd}^1 \quad .$$

We can thus conclude that if line 02 Fig. 2 passes in $\mathbf{KE}_1$ then the same line passes in $\mathbf{KE}_0$.

D Proof of Theorem 3

Proof. Consider the **IND** game defined in Fig. 10 and instantiated with the key transport from Fig. 11. Let Fig. 18 describe the KT simulator algorithms that get plugged into lines 01 and 09 of Fig. 10.

```
INITIALIZATIONS: $\mathrm{A} \leftarrow \emptyset$; $\mathrm{T}[\cdot] \leftarrow \diamond$; $st := (\mathrm{A}, \mathrm{T})$

Proc $\mathrm{sim}_{\mathrm{E}}\langle st \rangle(pk)$
00 $(\bar{c}, \overline{hd}, \_) \leftarrow \overline{\mathrm{enc}}(pk)$
01 $\bar{k} \leftarrow \$(\mathcal{K})$
02 $\tau \leftarrow \$(\mathcal{K})$
03 $hd \leftarrow \overline{hd} \mathbin{\|} \bar{k}$
04 $\mathrm{A} \overset{\cup}{\leftarrow} \{hd\}$
05 $\mathrm{T}[hd] \leftarrow \tau$
06 $c := (\bar{c}, \bar{k}, \tau)$
07 Return $c, hd$

Proc $\mathrm{sim}_{\mathrm{D}}\langle st \rangle(sk, c)$
08 $(\overline{hd}, st) \leftarrow \overline{\mathrm{dec}}(sk, \bar{c})$
09 $hd \leftarrow \overline{hd} \mathbin{\|} \bar{k}$
10 if $hd \in \mathrm{A}$:
11     $\tau' \leftarrow \mathrm{T}[hd]$
12     if $\tau' = \tau$: $\mu \leftarrow \$(\mathcal{K})$
13     else: Abort
14 else:
15     $\tau' \leftarrow \overline{\mathrm{eval}}(st, \bar{k})$
16     if $\tau = \tau'$: $\mu \leftarrow \overline{\mathrm{eval}}(st, \diamond)$
17     else: Abort
18 $k \leftarrow \bar{k} - \mu$
19 Return $hd, k$
```

Fig. 18. The simulator sim used in proving Theorem 3. Recall that $\overline{\mathrm{enc}}$ and $\overline{\mathrm{dec}}$ (lines 00 and 08) are the KDFEM encapsulation and decapsulation algorithms. Notice that $\mathrm{sim}_{\mathrm{E}}$ picks uniformly at random both, the masked key $\bar{k}$ and the tag τ. If the handle is authentic in $\mathrm{sim}_{\mathrm{D}}$ (line 10), then any key k could be output (lines 12 and 18). Otherwise k must be computed properly as in lines 16 and 18.

Consider now the **KE** game defined in Fig. 2 and instantiated with a KDFEM. Let $\mathcal{A}$ be the adversary playing **IND** and $\mathcal{B}$ playing **KE**. We build $\mathcal{B}$ from $\mathcal{A}$ in Fig. 19. We prove that the Promise, Share and Abort statements in Fig. 10 are properly simulated by the reduction both when $b = 0$ and when $b = 1$.

First, it is clear that the handle freshness of KT in Fig. 10 on line 04 is properly simulated by the handle freshness of the KDFEM construction (Fig. 8 line 34)) and thus in line 04 Fig. 19 for both $b = 0$ and $b = 1$. Second, assuming that line 12 Fig. 10 is executed, then line 13 would match lines 43 and 57 of Fig. 8 that are respectively called by lines 15, 18 and 21 from Fig. 19 when $b = 0$ and by 15, 25 and 26 when $b = 1$. Assuming that $b = 0$, dec does not abort and $hd \in \mathrm{A}$, if line 57 Fig. 8 passes on both eval calls of lines 18 and 21 Fig. 17, then line 15 Fig. 10 passes.

We now study the simulation of the abort lines. When $b = 0$, the KT decryption algorithm aborts if the tags do not match on line 11. This condition is reproduced precisely in lines 23 and 27 and under the same conditions (recall that Challenge outputs real masks and tags when $b = 0$). Things become trickier when $b = 1$. It is clear that if lines 23 and 27 Fig. 19 abort then so do lines 13 and 17 Fig. 18. We now prove the other way around. Suppose, that the else statement in line 14 is executed. We prove that line 27 Fig. 19 is executed with overwhealming probability. Having $\bar{k}' \neq \bar{k}$, we show that the probability of $\tau' = \tau$ is upper bounded by $\frac{q}{|\mathcal{K}|-q}$ where q is the number of queries $\mathcal{A}$ is allowed to make to the decryption oracle. In fact, for a fresh key $\bar{k}$, the probability that Challenge(hd) outputs exactly τ' is precisely $\frac{1}{|\mathcal{K}|}$. Having that the adversary can query multiple times the decryption oracle and on fresh inputs, then $\frac{1}{|\mathcal{K}|} + \frac{1}{|\mathcal{K}|-1} + \ldots + \frac{1}{|\mathcal{K}|-q+1} \leq \frac{q}{|\mathcal{K}|-q}$. We have shown that the simulator aborts if and only if $\mathcal{B}$ aborts.

As for the share statement in line 03, if $b = 1$ then $\mathrm{sim}_{\mathrm{E}}$ outputs elements of exactly the same distribution as Enc oracle from Fig. 19. In fact, when $b = 1$, the challenge oracle outputs uniformly random elements μ and τ that lead to $\bar{k} \leftarrow k + \mu$ being uniformly random. Additionally, the outputs of lines 04 and 15 Fig. 19 are respectively of the same distributions as those in lines 00 and 08 Fig. 19. Finally, line 17 Fig. 10 is only executed in the case of $b = 0$. Thus the challenge oracle on lines 19 and 22 of Fig. 19 outputs the real values τ' and μ which leads in line 28 Fig. 19 to the exact same key as in dec of Fig. 11. This concludes the proof of Theorem 3.

```
Proc $\mathcal{B}(pk)$
$\mathrm{A} \leftarrow \emptyset$
$b' \leftarrow \mathcal{A}(pk)$
Return $b'$

Oracle $\mathrm{Enc}(k)$
Pick fresh $sid$
$(\bar{c}, \overline{hd}) \leftarrow \overline{\mathrm{Enc}}(sid)$
$\overline{\mathrm{Eval}}_{\mathrm{E}}(sid, \diamond)$
$\mu \leftarrow \overline{\mathrm{Challenge}}(\overline{hd} \mathbin{\|} \diamond)$
$\overline{k} \leftarrow k + \mu$
$hd \leftarrow \overline{hd} \mathbin{\|} \overline{k}$
$\overline{\mathrm{Eval}}_{\mathrm{E}}(sid, \overline{k})$
$\tau \leftarrow \overline{\mathrm{Challenge}}(hd)$
$\mathrm{A} \xleftarrow{\cup} \{hd\}$
$c \coloneqq (\bar{c}, \overline{k}, \tau)$
Share $c, hd$

Oracle $\mathrm{Dec}(c)$
Pick fresh $sid$
$\overline{hd} \leftarrow \overline{\mathrm{Dec}}(sid, \bar{c})$
$hd \leftarrow \overline{hd} \mathbin{\|} \overline{k}$
if $hd \in \mathrm{A}$:
  $\overline{\mathrm{Eval}}_{\mathrm{D}}(sid, \overline{k})$
  $\tau' \leftarrow \overline{\mathrm{Challenge}}(hd)$
  if $\tau = \tau'$:
    $\overline{\mathrm{Eval}}_{\mathrm{D}}(sid, \diamond)$
    $\mu \leftarrow \overline{\mathrm{Challenge}}(\overline{hd} \mathbin{\|} \diamond)$
  else: Abort
else:
  $\tau' \leftarrow \overline{\mathrm{Eval}}_{\mathrm{D}}(sid, \overline{k})$
  if $\tau = \tau'$: $\mu \leftarrow \overline{\mathrm{Eval}}_{\mathrm{D}}(sid, \diamond)$
  else: Abort
$k \leftarrow \overline{k} - \mu$
Share $hd, k$
```

Fig. 19. A reduction from **IND** of KT to **KE** of KDFEM. In the real world, the challenge oracle outputs the real tag and mask. In the ideal world, it outputs uniformly random strings from the key space $\mathcal{K}$.