A short version of this paper appears in the proceedings of ACM CCS 2024. This is the full version.

Keeping Up with the KEMs: Stronger Security Notions for KEMs and Automated Analysis of KEM-based Protocols

Version 1.1.0, May 29, 2024*

Cas Cremers, Alexander Dax, and Niklas Medinger

CISPA Helmholtz Center for Information Security
{cremers,alexander.dax,niklas.medinger}@cispa.de

Abstract

Key Encapsulation Mechanisms (KEMs) are a critical building block for hybrid encryption and modern security protocols, notably in the post-quantum setting. Given the asymmetric public key of a recipient, the primitive establishes a shared secret key between sender and recipient. In recent years, a large number of abstract designs and concrete implementations of KEMs have been proposed, e.g., in the context of the NIST process for post-quantum primitives.

In this work, we (i) establish stronger security notions for KEMs, and (ii) develop a symbolic analysis method to analyze security protocols that use KEMs. First, we generalize existing security notions for KEMs in the computational setting, introduce several stronger security notions and prove their relations. Our new properties formalize in which sense outputs of the KEM uniquely determine, i.e., *bind*, other values. Our new binding properties can be used, e.g., to prove the absence of attacks that were not captured by prior security notions. Among these, we identify a new class of attacks that we coin re-encapsulation attacks.

Second, we develop a family of fine-grained symbolic models that correspond to our hierarchy of computational security notions, and are suitable for the automated analysis of KEM-based security protocols. We encode our models as a library in the framework of the Tamarin prover. Given a KEM-based protocol, our approach can automatically derive the minimal binding properties required from the KEM; or, if also given a concrete KEM, can analyze if the protocol meets its security goals. In case studies, Tamarin automatically discovers, e.g., that the key exchange protocol proposed in the original Kyber paper [12] requires stronger properties from the KEM than were proven in [12].

1 Introduction

A Key Encapsulation Mechanism (KEM) [18] is a common building block in security protocols and cryptographic primitives such as hybrid encryption. Intuitively, a KEM can be seen as a specialized version of Public Key Encryption (PKE) that, instead of encrypting a payload message, specifically serves to generate and share a symmetric key between sender and recipient. During the last decade, many post-quantum secure KEMs have been proposed, see e.g., [2, 3, 6, 7, 11, 12, 25, 31, 37, 39]. This has made KEMs a prime candidate to replace Diffie-Hellman constructions, for which no practical post-quantum secure scheme is currently available.

The traditional security notion for a KEM is a version of IND-CCA that is directly inherited from its related Public Key Encryption (PKE) notion. Intuitively, a KEM is (IND-CCA) secure if, given a ciphertext, an adversary that does not have the corresponding private key cannot tell the difference between a random key and the encapsulated key. Additionally, robustness-like properties have been proposed for KEMs in [29], which similarly inherit from their PKE counterparts. Initially, "robustness" [1] was defined in the PKE setting as the difficulty of finding a ciphertext valid under two different encryption keys. Phrased differently, a PKE is robust if a ciphertext "binds to" (only decrypts under) one key.

In this work, we set out to enable automated analysis of KEM-based security protocols that can take the differences between concrete KEMs into account. We first systematically explore the possible

*We provide a list of main changes in Appendix F.

binding properties of KEMs. Our work is similar in spirit to explorations in the space of digital signatures [13, 21, 33, 42] and authenticated encryption [19, 24, 28, 36], where recent works have identified many desirable binding properties for these primitives that could have prevented real-world attacks.

Our systematic analysis leads to the formulation of several core binding properties for KEMs, with multiple variants. Whereas traditional KEM robustness properties only considered binding values to a specific ciphertext, we propose variants that bind values to a specific output key. We argue this is much more in line with viewing a KEM as a one-pass key exchange. Similarly, implicitly rejecting KEMs resemble implicitly authenticated key exchanges, where correct binding properties of the established key prevent classes of unknown key share attacks [9]. We relate our properties to properties previously reported in the literature, as well as related notions such as contributory behavior. We provide a full hierarchy for our properties with implications and separating examples.

We use our new hierarchy to develop novel symbolic analysis models that reflect the binding differences between concrete KEMs. We implement our models in the framework of the Tamarin prover and apply the methodology to several case studies.

Notably, our automated analysis uncovers an attack on an example key exchange protocol in the Kyber documentation when instantiated with another KEM, which proves that the protocol design in fact relies on properties of the used KEM beyond just IND-CCA. We coin this type of attack a "re-encapsulation attack", as it relies on the adversary encapsulating keying material that it previously obtained from decapsulation, causing two ciphertexts to decapsulate to the same key. We also show how our novel properties can prove the absence of such attacks.

Our main contributions are the following:

1. We introduce a novel hierarchy of computational binding properties for KEMs. We position existing notions within our hierarchy and introduce several new properties. KEMs that satisfy our key-binding properties will leave fewer pitfalls for protocol designers.

2. We develop a symbolic analysis methodology to automatically analyze the security of KEM-based protocols, using fine-grained models of their KEMs, and implement them in the Tamarin prover. Our methodology can also be used to automatically establish the KEM binding properties that are needed for a protocol to be secure. In case studies, our automated analysis finds new attacks and missed proof obligations.

3. Our findings include a new type of protocol attack we coin a "re-encapsulation attack".[1] Notably, such attacks can occur even with robust IND-CCA secure KEMs but not with KEMs that satisfy our stronger key-binding properties.

Reproducibility and artifacts Our symbolic KEM model library, case studies, and execution instructions are available for inspection and reproducibility at [20].

Outline We provide background and further related work in the computational setting in Section 2. We then motivate our binding properties by the example of the re-encapsulation attack in Section 3, before developing our family of new security notions in Section 4. We then turn to developing our automated symbolic analysis in Section 5 and report on case studies in Section 6. We conclude in Section 8.

Appendices We present proofs establishing the relationship between our KEM properties in Appendices A and B. Appendix C provides details on our symbolic KEM implementation. In Appendix E we give an overview of the binding properties of some prominent KEM schemes.

2 Background

We now give the necessary background knowledge on KEMs and their main security notions.

KEMs and IND-CCA security

A key encapsulation scheme [18] KEM consists of three algorithms (KeyGen, Encaps, Decaps). It is associated with a key space $\mathcal{K}$ and a ciphertext space $\mathcal{C}$. The probabilistic key-generation algorithm KeyGen creates a key pair (pk, sk) where pk is the public key and sk is the secret key. Given a public key

[1] Tamarin discovered this attack and we coined the term already in September 2022. We communicated this to one of the authors of [8], who later found an attack from this class in their work on PQ-X3DH and also used our terminology in [8].

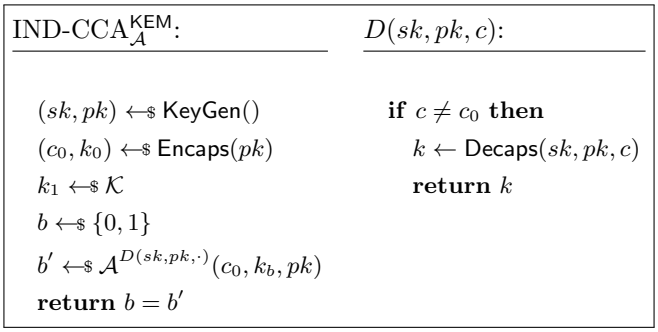

Figure 1: IND-CCA experiment for KEMs. Originally introduced in [18], we re-use syntax from [12].

pk as input, the probabilistic encapsulation algorithm Encaps returns a ciphertext $c \in \mathcal{C}$ and a key $k \in \mathcal{K}$. In this paper, we sometimes want to view Encaps as a deterministic algorithm with explicit randomness r, in which case we write $\mathsf{Encaps}(pk; r)$. To avoid ambiguity, we refer to k as the *output key* or the *shared secret*. The deterministic decapsulation algorithm Decaps uses a public key pk, a secret key sk, and a ciphertext $c \in \mathcal{C}$ to compute an output key $k \in \mathcal{K}$ or the error symbol $\perp$ that represents rejection. If decapsulation never returns $\perp$, we call KEM an *implicitly rejecting* KEM. Otherwise, we call it an *explicitly rejecting* KEM. We say that a KEM is ϵ-correct if for all $(sk, pk) \leftarrow\$ \mathsf{KeyGen}()$ and $(c, k) \leftarrow\$ \mathsf{Encaps}(pk)$, it holds that $\mathbf{Pr}[\mathsf{Decaps}(sk, c) \neq k] \leq \epsilon$.

The security of a KEM is defined through indistinguishability of the output key $k \in \mathcal{K}$ computed by Encaps against different adversaries. The standard security notion is resistance against a chosen-ciphertext attack (IND-CCA) [12, 47]. We recall the formal definition of the IND-CCA experiment shown in Figure 1.

First, the experiment creates a key pair (sk, pk) and encapsulates against the public key, returning (c_0, k_0). Next, it samples a random key k_1 from the key space and a random bit b. Then, the adversary $\mathcal{A}$ is given c_0, the key corresponding to the bit b, and pk, and outputs its guess b'. The adversary wins if they correctly guessed b, i.e., $b = b'$. During the experiment, the adversary has access to the decapsulation oracle $\mathsf{Decaps}(sk, \cdot)$, which returns the decapsulation of any ciphertext c except for the challenge ciphertext c_0.

Fujisaki-Okamoto (FO) transform

A common construction for KEMs is the FO transform [27]. The FO transform can be used to turn any weakly secure (i.e., IND-CPA) public-key encryption scheme into a strongly (i.e., IND-CCA) secure KEM scheme by hashing a random message (and optionally other values) into an output key. Since the FO transform gives cryptographers a straightforward way to create a post-quantum secure KEM from a post-quantum secure PKE, these KEMs have surged in popularity and are now the de-facto standard post-quantum secure KEMs. All the finalists of the KEM NIST PQC [39] process are FO-KEMs.

3 Re-encapsulation attacks

Our initial motivation for this work was to uncover the subtle difference in guarantees offered by different KEM designs and to analyze their impact on protocols. As we will see later, this leads to a hierarchy of new binding properties, which we used to build an automated analysis that discovered new attacks. Notably, Tamarin found instances of a class of attacks that we call *re-encapsulation attacks*. While re-encapsulation attacks were not our original motivation, they clearly illustrate important binding properties that were not captured by previous security notions.

Intuitively, re-encapsulation attacks exploit the fact that for some KEMs, it is possible to decapsulate a ciphertext to an output key k and then produce a second ciphertext for a different public key that decapsulates to the same k. This can be possible even for robust IND-CCA KEMs since neither IND-CCA nor robustness prescribe that the output key binds a unique public key. At the protocol level, a re-encapsulation attack can typically manifest as an unknown-key-share attack, where two parties compute the same key despite disagreeing on their respective partners.

We illustrate this on a concrete example, which was automatically found by Tamarin, on an authenticated key exchange protocol from the Kyber paper [12] shown in Figure 2.

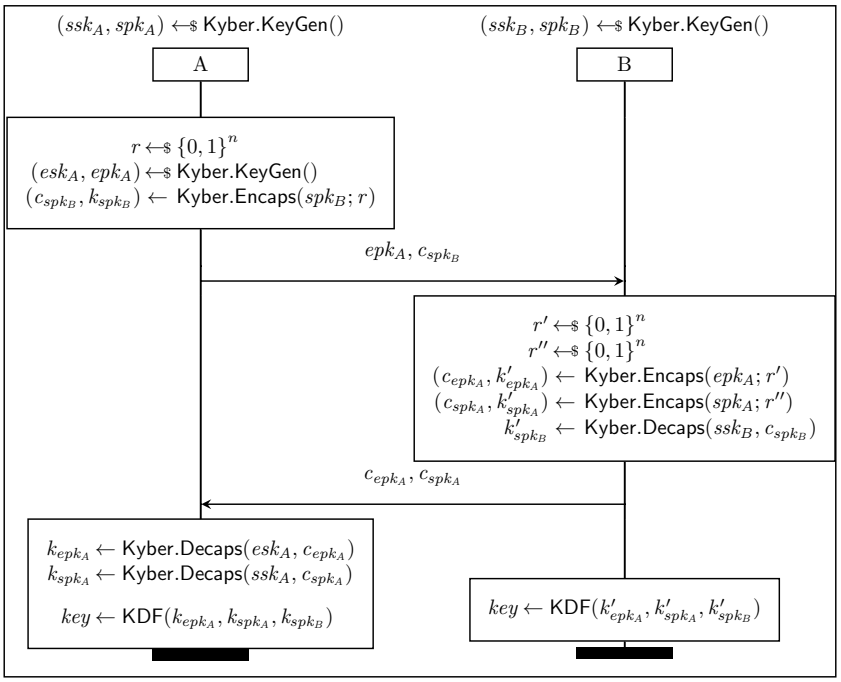

Figure 2: The authenticated key exchange described in the original Kyber paper [12].

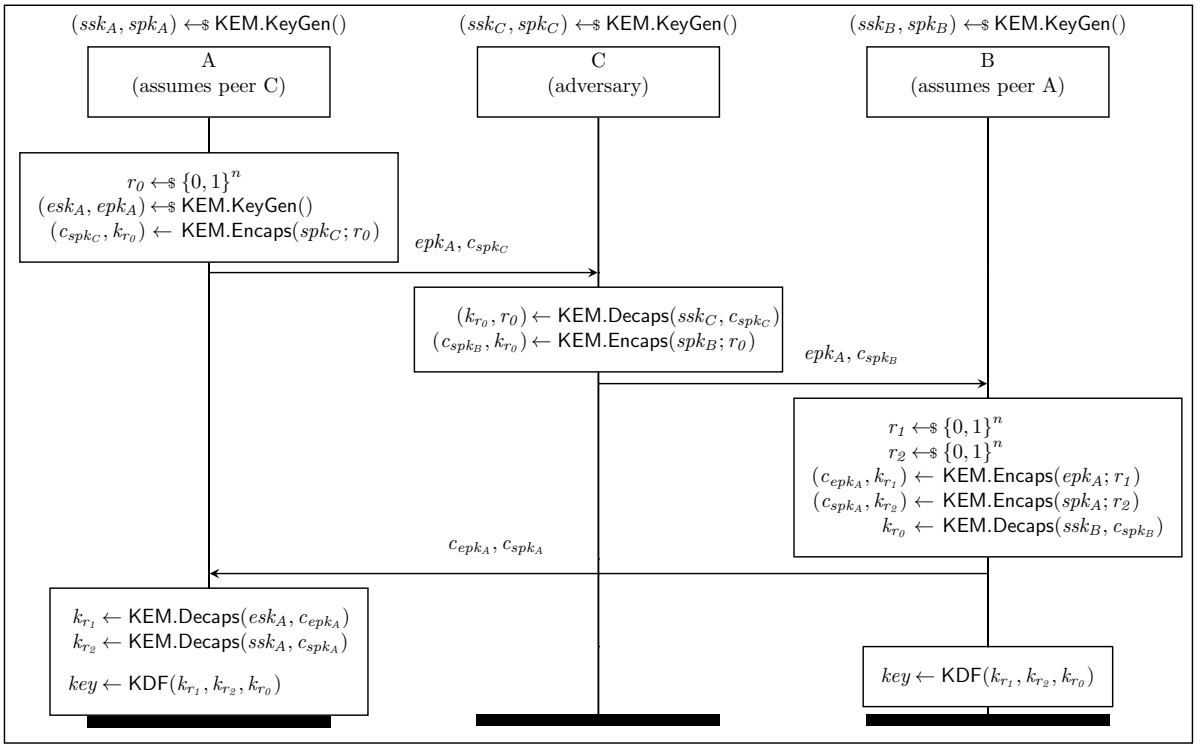

Figure 3: Re-encapsulation attack against the Authenticated Key Exchange (AKE) suggested for the Kyber KEM [12] where the adversary C coerces honest A into unknowingly sharing the *key* with honest B, who correctly thinks they are being contacted by honest A. This violates the implicit key agreement guarantee for B, who expects to share a key with someone that assumes B is the peer. Note that this attack is only possible when the AKE is instantiated with a KEM that does not bind the output key to the public key, and is not possible when instantiated with Kyber.

We stress that when the key exchange protocol is instantiated with Kyber as intended by the paper, the protocol seems secure. However, can Kyber be replaced by any other KEM? In the paper, Kyber is

only proven to be IND-CCA secure. Is IND-CCA sufficient for the protocol's security? It turns out this is not the case.

To show this, we consider the same key exchange protocol, but instantiated with $\mathsf{KEM}_m^{\perp}$ from [30]. $\mathsf{KEM}_m^{\perp}$ is an FO-KEM (cf. Section 2). In the context of our work, FO-KEMs are interesting because they have a property that is not captured by the current syntax of KEMs: when a party A decapsulates a ciphertext to learn k, they can also learn the message m that was encrypted by the underlying PKE. This is unavoidable for any PKE-based KEM because the ciphertext that contains m needs to be decrypted before deriving k from m.

To simplify notation in this example, we assume that we can infer the randomness r from the message m. This allows for a slightly more abstract description of the attack, but we can instantiate the attack for any concrete, vulnerable FO-KEM without this assumption. We capture this by writing $(k, r) \leftarrow \mathsf{KEM.Decaps}(sk, c)$ in this example.

We now explain the re-encapsulation attack in Figure 3. In the attack, A and B are honest. The adversary C wants to coerce B into establishing a key shared with A, where B mistakenly assumes that A thinks they share the key with B; instead, A will think they share it with C. This is a so-called unknown-key-share attack [9], which violates B's implicit key agreement.

The attack proceeds as follows: A initiates communication with C, after which C decapsulates the ciphertext c_{spk_C} to obtain k_{r_0} and, more importantly, r_0, which was used by A to create c_{spk_C}. Now, C impersonates A towards B by encapsulating against B's static public key with r_0 and forwarding the resulting ciphertext and epk_A. B responds with the expected values to A, as B thinks A is communicating with them. Finally, A decapsulates the ciphertexts received from B, and both A and B derive the final key. Since we instantiated the protocol with $\mathsf{KEM}_m^{\perp}$, the keys obtained via Decaps only depend on the randomness supplied by the encapsulating party. As a result, A and B derive the same *key*; this is a violation of implicit authentication since A thinks they now share a key with C, which does not match B's expectations.

One might wonder whether a KEM with strong robustness properties would prevent this attack. Unfortunately, this is not the case: robustness properties reason about *a single ciphertext* c that should not decapsulate to the same key under different key pairs. However, our re-encapsulation attack revolves around two *different* ciphertexts: based on A's ciphertext, the malicious C creates a different ciphertext c_{spk_B} that decapsulates to the same key as c_{spk_C} by reusing the randomness r_0. For more details, see Cons. B.2 .

4 New security notions for KEMs

ID	P	Q	Property	Explanation	Relation to existing notions
1	{k}	{ct}	*X-BIND-K-CT*	Output key binds the ciphertext.	
2	{k}	{pk}	*X-BIND-K-PK*	Output key binds the public key.	
3	{ct}	{k}	*X-BIND-CT-K*	Ciphertext binds the output key.	
4	{ct}	{pk}	*X-BIND-CT-PK*	Ciphertext binds the public key.	*HON-BIND-CT-PK* is equivalent to SROB [29].
5	{k, ct}	{pk}	*X-BIND-K, CT-PK*	Together, the output key and ciphertext bind the public key.	*HON-BIND-K, CT-PK* is equivalent to SCFR [29].
6	{k, pk}	{ct}	*X-BIND-K, PK-CT*	Together, the public key and the output key bind the ciphertext.	*LEAK-BIND-K, PK-CT* is equivalent to CCR [5].

Table 1: The six core instantiations of our generic binding property *X-BIND-P-Q* before choosing $X \in \{HON, LEAK, MAL\}$.

We now turn to our first main objective: to establish a generic family of binding properties of KEMs. We first identify the elements that may be candidates for binding. The syntax of a KEM includes a long-term key pair, a ciphertext, and an (output) key. In some formalizations, the randomness of the KEM is made explicit, but we are looking for universal black-box notions that do not require us to know the internals of a KEM. With respect to the long-term key pair, we note that we want the guarantees to be relevant for both sender and recipient, which means we only consider the public key as the identifying aspect of the key pair. This leaves us with pk, ct, and k: we expect that for each invocation of the KEM's encapsulation with the same pk, the outputs ct and k would be unique.

We can thus wonder: if we have a specific instance of one of these, does that mean the others are uniquely determined? If we have a ciphertext, can it only be decapsulated by one key?

4.1 Design choices

To define our notions, we make the following design decisions:

1. We consider the set of potential binding elements $BE = \{\mathsf{pk}, \mathsf{ct}, \mathsf{k}\}$.
2. We will consider if an instance of a set $P \subset BE$ "binds" some instance of another set of elements $Q \subset BE$ with respect to decapsulation with the KEM. Thus, "P binds Q" if for fixed instances of P there are no collisions in the instances of Q.
3. When using a KEM, pk is re-used in multiple encapsulations by design. Thus, pk does not bind any values on its own, and we hence exclude it from occurring in P alone. However, ciphertexts or keys might bind a public key pk, so it may occur in Q alone.
4. Adding multiple elements in the set Q corresponds to a logical "and" of the singleton versions, i.e., we have that P binds $\{q_1, \ldots, q_n\}$ iff for all $i \in [n]$. P binds $\{q_i\}$. We therefore choose to focus on the core properties, i.e., with $|Q| = 1$.
5. We require P and Q to be disjoint: elements that would occur on both sides are trivially bound. Additionally, we require both P and Q to be non-empty.
6. For all of our properties, we will consider honest variants (i.e., the involved key pairs are output by the key generation algorithm of the KEM), leakage variants (i.e., the involved key pairs are output by the key generation algorithm of the KEM and then leaked to the adversary), and malicious variants (i.e., the adversary can create the key pairs any way they like in addition to the key generation).

Based on the above choices, we have five choices for P. We refer to this set of choices as $\mathbb{P} = \big\{\{\mathsf{k}\}, \{\mathsf{ct}\}, \{\mathsf{k}, \mathsf{ct}\}, \{\mathsf{k}, \mathsf{pk}\}, \{\mathsf{ct}, \mathsf{pk}\}\big\}$. For Q, we can choose from the set $\mathbb{Q} = \big\{\{\mathsf{pk}\}, \{\mathsf{k}\}, \{\mathsf{ct}\}\big\}$. Without disjointness this would yield 5×3 options, but since we require the sets to be disjoint, this yields seven combinations.

One of these seven cases is the case where $P = \{\mathsf{pk}, \mathsf{ct}\}$ and $Q = \{\mathsf{k}\}$. This property holds when the public key pair and the ciphertext, which are the inputs to Decaps, bind the output key. If Decaps is deterministic, this is trivially true. We will therefore not consider this case in the remainder of the paper, leaving us with six combinations that we will investigate further, which we show in Table 1.

Defining properties over pk

In contrast to the *standard* KEM API, our API has the decapsulation algorithm explicity take a public key as input (see Section 2). We opted for this choice because it allows us to easily refer to the public key in both encapsulation and decapsulation operations, which allows us to easily state our binding properties. This decision comes with an assumption that we want to make explicity now: We assume that our KEM schemes do not ignore the pk in Decaps. Whenever the Decaps algorithm uses the public key, the supplied pk is used; it is not derived or extracted from the sk.

This assumption is necessary to avoid trivial attacks on the *MAL* versions of our security properties: A KEM could simply ignore the pk supplied to Decaps and, instead, use a public key stored inside of the sk. This is often done in practice, and Schmieg describes possible attacks in [49].

We want to stress that while our KEM API is syntactically different from the standard API, it is semantically equivalent. To move from one API to the other, one simply needs to extract the public key stored inside of the secret key and explicitly supply it, or put the explicitly supplied public key into the secret key.

4.2 Naming conventions

Naming security notions is hard; once names are fixed, they tend to stick around for (too) long. We opt here for clarity and being descriptive at the cost of some verbosity. In the literature, it is more common to collapse all of these properties into "robustness" or "collision-freeness", but this becomes very ambiguous because one can imagine many subtle variants, depending on the exact robust/collision-free element in the construction. This has lead to a long list of non-descriptive names in the literature, including: Robustness, Fuller Robustness (FROB), even Fuller Robustness (eFROB), CROB, KROB, SROB, USROB, WROB, XROB, SCFR, WCFR, CCR, etc. In contrast, we illustrate our descriptive naming scheme for our binding properties in Figure 4.

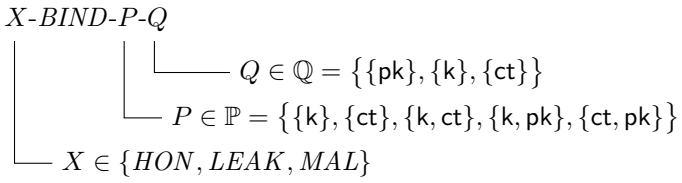

Figure 4: Design space and naming conventions for our security properties: For a KEM scheme that is *X-BIND-P-Q* secure, we say that "P [honestly|leak|maliciously] binds Q", using "honestly" when $X = HON$, "leak" when $X = LEAK$, and "maliciously" when $X = MAL$. We commonly omit set brackets in the notation when clear from the context, and we use uppercase for all characters. For example, *HON-BIND-CT-PK* corresponds to "ct honestly binds pk."

$X\text{-}BIND\text{-}P\text{-}Q^{\mathsf{KEM}}_{\mathcal{A}}$:

$\mathsf{sk}_0, \mathsf{pk}_0 \leftarrow \mathsf{KeyGen}()$
$\mathsf{sk}_1, \mathsf{pk}_1 \leftarrow \mathsf{KeyGen}()$
if $\mathsf{pk} \in Q$: $b \leftarrow 1$
else if $\mathsf{pk} \in P$: $b \leftarrow 0$
else : $b \in \{0,1\}, st \leftarrow \mathcal{A}()$
$\mathsf{sk}_1, \mathsf{pk}_1 \leftarrow \mathsf{sk}_b, \mathsf{pk}_b$
if $X = HON$: $\mathsf{ct}_0, \mathsf{ct}_1 \leftarrow \mathcal{A}^{D_{b'}(\mathsf{sk}_{b'}, \cdot)}(\mathsf{pk}_0, \mathsf{pk}_1, st)$
if $X = LEAK$: $\mathsf{ct}_0, \mathsf{ct}_1 \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{sk}_0, \mathsf{pk}_1, \mathsf{sk}_1, st)$
$\mathsf{k}_0 \leftarrow \mathsf{KEM.Decaps}(\mathsf{sk}_0, \mathsf{pk}_0, \mathsf{ct}_0)$
$\mathsf{k}_1 \leftarrow \mathsf{KEM.Decaps}(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{ct}_1)$
if $k_0 = \bot \vee k_1 = \bot$: **return** 0
// $\mathcal{A}$ wins if $\neg\big((\forall x \in P \,.\, x_0 = x_1) \implies (\forall y \in Q \,.\, y_0 = y_1)\big)$
return $\forall x \in P \,.\, x_0 = x_1 \wedge \exists y \in Q \,.\, y_0 \neq y_1$

Figure 5: Generic game for our new binding notions *X-BIND-P-Q* for $X \in \{HON, LEAK\}$.

4.3 Generic binding notions of KEMs

We now introduce the generic security notion for our class of binding properties. In Figure 5 we show the generic game for $X \in \{HON, LEAK\}$, and in Figure 6 we show the game when $X = MAL$.

Definition 4.1. *Let* KEM *be a key encapsulation mechanism. Let* $X \in \{HON, LEAK, MAL\}$*, let* $P \in \mathbb{P}$ *and* $Q \in \mathbb{Q}$ *such that* $P \cap Q = \emptyset$*. We say that* KEM *is X-BIND-P-Q-secure iff for any PPT adversary* $\mathcal{A}$*, the probability that X-BIND-P-*$Q^{\mathsf{KEM}}_{\mathcal{A}}$ *returns 1 (true) is negligible.*

In our definitions, $X \in \{HON, LEAK, MAL\}$ indicates the adversary's control over the considered key pairs. In the honest case $X = HON$, two honestly generated key pairs are considered, and we give the adversary access to a decapsulation oracle $D_{b'}(\mathsf{sk}_{b'}, \cdot)$, where $b' \in \{0,1\}$, that they can use to decapsulate ciphertexts with either secret key. For the leak case $X = LEAK$, we also give the adversary access to both secret keys. In the malicious case $X = MAL$, the adversary can choose or construct the key pairs in any way they want. For $X \neq HON$ we do not need a decapsulation oracle since the adversary already has the secret keys.

If $X \in \{HON, LEAK\}$, we check whether $\mathsf{pk} \in P$ or $\mathsf{pk} \in Q$ and choose the key pairs for the second call to Decaps accordingly. For $X = MAL$, the adversary chooses the key pairs.

The difference between $X = LEAK$ and $X = HON$ is whether the adversary only has access to a decapsulation oracle or has access to the secret keys. Given the secret keys, the adversary can decapsulate ciphertexts and learn intermediate values of the decapsulation. If they only have the oracle, they only learn the output of decapsulation but no intermediate values.

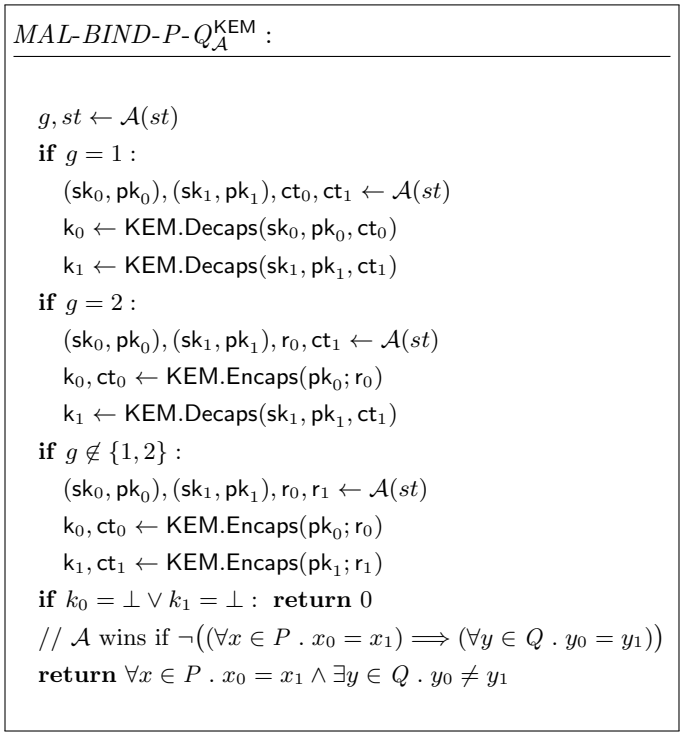

Figure 6: Generic game for our new binding notions *MAL-BIND-P-Q*. The adversary can use g to choose whether they want to find a collision between two calls to Encaps, Decaps or a single call to both, in line with [26].

4.4 Relating binding to contributive behavior

In the context of other cryptographic primitives, the notion of contributive (or contributory) behavior exists. Intuitively, in a two-party protocol that yields some randomized output, a protocol is contributive if the output is not only determined by one of the parties, but both contribute to the results.

For example, in a standard FO-KEM such as $\mathsf{KEM}^{\bot}_{m}$ from [30], the randomness sampled for encapsulation is the direct (and only) input for the key derivation function (KDF). Thus, for $\mathsf{KEM}^{\bot}_{m}$, the only party that contributes to the output key is the sender. We say that such KEMs are non-contributory and can enable re-encapsulation attacks, as described in Section 3.

In contrast, if the KEM's key binds the public key (e.g., by including the public key in the KDF), then the KEM satisfies *MAL-BIND-K-PK*, and we say that the KEM is contributory because the recipients' key contributes to the output key.

If the KEM's key binds the ciphertext (*MAL-BIND-K-CT*), it is not immediately clear whether this is enough to make the KEM contributory, and it depends on the collision freeness [29] (SCFR) of the underlying PKE. If the underlying PKE is not SCFR, i.e., it is possible to decrypt a single ciphertext to the same message with different secret keys, then the KEM is not contributory. The reason for that is that a single ciphertext is valid for multiple public keys, and thus the identity of the receiver is not bound by including the ciphertext in the output key of the KEM. On the other hand, if the PKE is strongly collision free (or even robust) then including the ciphertext makes the KEM contributory. See Corollary 4.9 and Section 4.7 for more details.

4.5 Relationship to other Properties

Our generic security notions cover a wide array of different properties and generalize existing security properties in the literature. In this section, we give a short overview of other properties that can be expressed using our generic notions.

When $P = \{\mathsf{ct}\}$ and $Q = \{\mathsf{pk}\}$, our generic games resemble different robustness notions. For example, *HON-BIND-CT-PK* corresponds to strong robustness (SROB) from [29] and *HON-BIND-K, CT-PK* corresponds to strong collision freeness from [29]. Interestingly, the strong robustness notion from [1], which coined the term in the context of PKEs, is weaker than both our *HON-BIND-CT-PK* notion and

the strong robustness notion from [29], since they both allow the adversary to query an oracle; this was not possible in the original definition. As an example, we compare our notions to the SROB and SCFR notions from [29] in Figure 7.

The properties introduced for PKEs in [26] are formulated analogously to ours: complete robustness (CROB) resembles *MAL-BIND-CT-PK*, and their intermediate notion unrestricted strong robustness (USROB) resembles our *HON-BIND-CT-PK* notion.

Our *HON-BIND-K, CT-PK* is equivalent to the strong collision freeness property from [29]; it is a weaker version of SROB, where an adversary has to decapsulate a single ciphertext to the same output key for distinct public keys.

LEAK-BIND-K, PK-CT matches the ciphertext collision resistance (CCR) property for KEMs from [5].

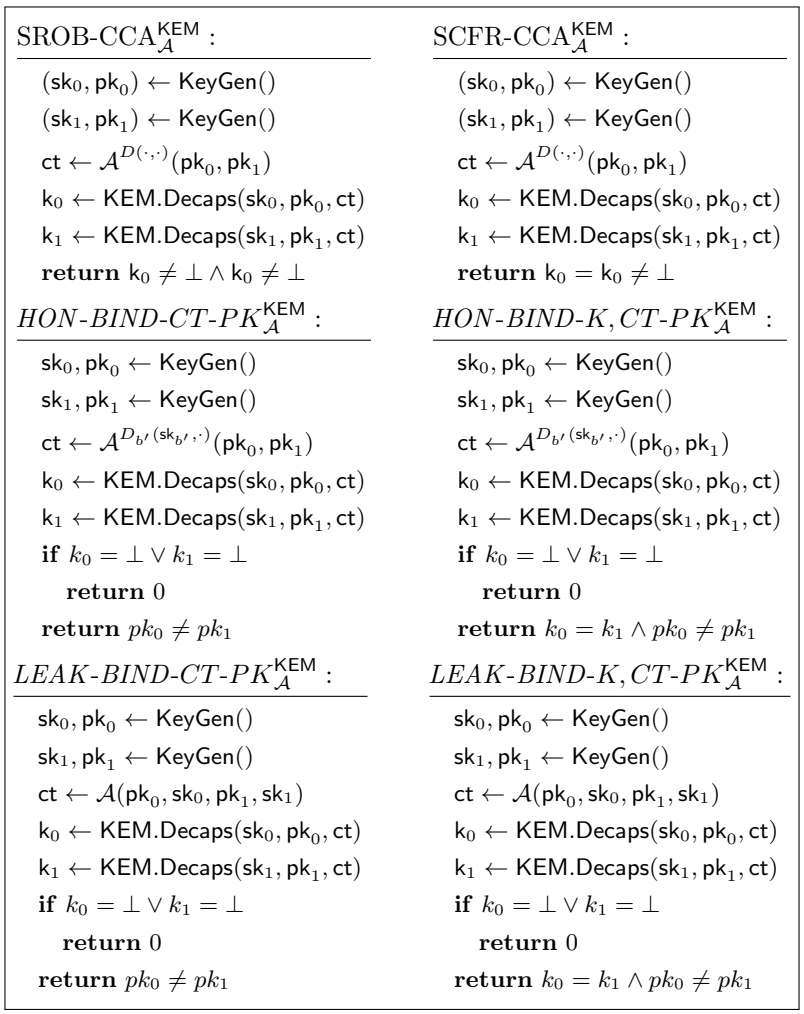

Figure 7: At the top, the strong robustness and strong collision freeness definitions from [29].In the middle, our *HON-BIND-CT-PK* and *HON-BIND-K, CT-PK* definitions which correspond to SROB and SCFR respectively. At the bottom, our *LEAK-BIND-CT-PK* and *LEAK-BIND-K, CT-PK* definitions which give the adversary access to the secret keys.

4.6 Relations and implications

In this section, we show the relations between our various binding notions.

We provide the proofs and separating examples in Appendix B and Appendix A and only show the main results here, resulting in the hierarchy in Figure 8. We show that our properties are largely orthogonal for different choices of P and Q: there exist KEMs that have a certain property but do not meet the other properties in our hierarchy. We summarize these results in Table 4.

We first formalize the ordering of our threat models.

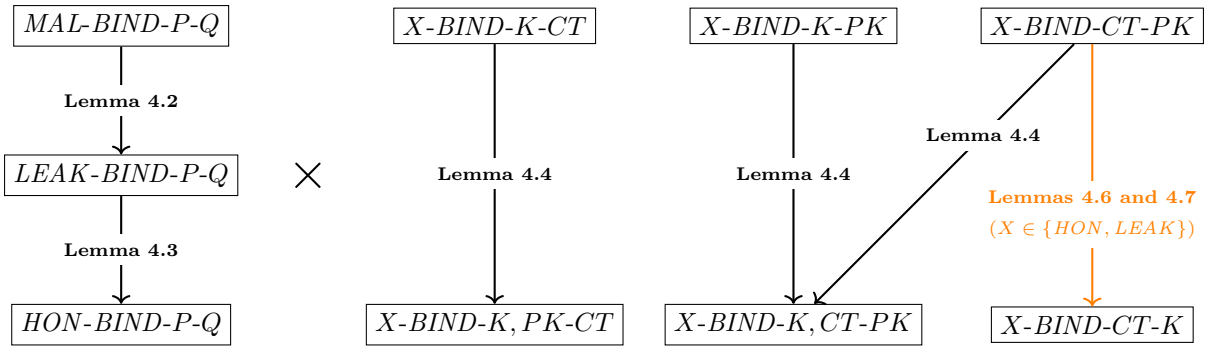

Figure 8: General hierarchy of binding properties for KEMs. An edge from A to B indicates that any KEM that is A-binding is also B-binding. Missing edges represent the existence of separating examples, which we show in Table 4. The hierarchy left of the $\times$ denotes the implications between the different attacker capabilities {MAL, LEAK, HON}. The hierarchy on the right of the $\times$ represents the implications between our binding properties, independent of the attacker capabilities. We can combine both hierarchies by choosing a node from the left and instantiating P and Q according to a node from the right, resulting in, for instance, an implication between *MAL-BIND-CT-PK* and *HON-BIND-K, CT-PK*. For $X =$ MAL, *X-BIND-CT-PK* and *X-BIND-CT-K* are incomparable. The orange edge indicates that for $X \in$ {HON, LEAK}, *X-BIND-CT-PK* implies *X-BIND-CT-K*.

Lemma 4.2. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. If* KEM *is MAL-BIND-P-Q-secure, then* KEM *is also LEAK-BIND-P-Q-secure.*

Lemma 4.3. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. If* KEM *is LEAK-BIND-P-Q-secure, then* KEM *is also HON-BIND-P-Q-secure.*

The next lemma states that adding elements to P or removing elements from Q weakens a property. Intuitively, if, e.g., k binds pk, then k *and* ct also bind pk (since it is already bound by k).

Lemma 4.4. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. For* $X \in \{MAL, LEAK, HON\}$*, if* KEM *is X-BIND-P-Q′-secure and* $P \subseteq P' \wedge Q \subseteq Q'$*, then* KEM *is also X-BIND-P′-Q-secure.*

Theorem 4.5. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. For* $X \in \{MAL, LEAK, HON\}$*, if* KEM *is X-BIND-P-Q′-secure, X-BIND-Q-R′-secure and* $P \subseteq P'$*,* $Q \subseteq Q' \cup P'$*, and* $R \subseteq R'$*, then* KEM *is also X-BIND-P′-R-secure.*

In the following, we will investigate the relation between *X-BIND-CT-PK* and *X-BIND-CT-K* more closely. First, we will prove that the former implies the latter when $X = HON$ or $X = LEAK$. Then, we will show that this is not the case when $X = MAL$.

Lemma 4.6. *Let* KEM *be a KEM that is HON-BIND-CT-PK secure. Then* KEM *is also HON-BIND-CT-K secure.*

Lemma 4.7. *Let* KEM *be a KEM that is LEAK-BIND-CT-PK secure. Then* KEM *is also LEAK-BIND-CT-K secure.*

Proposition 4.8. *There exists a KEM scheme* KEM *that is MAL-BIND-CT-PK but not MAL-BIND-CT-K.*

These results give rise to a hierarchy for our properties, which we visualize in Figure 8. We want to highlight that for so-called *implicitly* rejecting KEMs, i.e., KEMs whose Decaps algorithm never returns $\perp$, we establish a reduced hierarchy in Section 4.8 by showing that for these KEMs the ciphertext alone cannot bind other values. As a result, implicitly rejecting KEMs cannot meet properties like *HON-BIND-CT-PK*, i.e., be robust.

4.7 Ensuring strong binding properties

In nearly all KEM designs, the last step of encapsulation and decapsulation is to produce the output key by using a KDF (Key Derivation Function); if not, such a step can be added. In order to ensure that the

key binds another element, we can simply add this element to the KDF inputs. Thus, to achieve *MAL-BIND-K-CT* and *MAL-BIND-K-PK*, we can simply add CT and PK to the input of the key derivation function. Of course, this is not the only way to achieve such binding properties: Leaving out either CT or PK does not mean that the corresponding property does not hold; it simply means there is a proof obligation to show that a KEM meets such a binding property without this construction.

In practice, and in particular for post-quantum KEMs, the public key can be substantially larger than the ciphertext. It may therefore be desirable to avoid directly including the public key in the key derivation. For this case, we give Corollary 4.9 below. It implies that KEM designers who want to achieve *X-BIND-K-PK* but want to avoid using the public key in the KDF can instead use a robust PKE and include the ciphertext in the KDF, which will yield the desired binding to the public key.

Corollary 4.9. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. For all* $X \in \{MAL, LEAK, HON\}$ *we have that if* KEM *is* X*-BIND-K-CT-secure and* X*-BIND-K*, CT*-PK-secure, then* KEM *is also* X*-BIND-K-PK-secure.*

We provide the proof in the appendix of A.9.

4.8 Implicitly rejecting KEMs

Many real-world KEMs are so-called *implicitly rejecting*: Their decapsulation algorithm never returns $\perp$ for a valid ciphertext and any valid private key. However, only when decapsulating with the correct private key (corresponding to the public key used for encapsulation), the correct key is output.

Intuitively, an implicitly rejecting KEM is similar to an implicitly authenticated key exchange: Successfully completing the protocol does not imply that someone else has the same key or sent any message; instead, the guarantee is that only the correct party can possibly compute the same secret key.

Rejection Keys

When decrypting a ciphertext using implicitly rejecting KEMs, one of two outcomes can occur: if the ciphertext is valid for the KEM key pair, decryption proceeds successfully. However, if the ciphertext is invalid, the algorithm still yields an output key, referred to as the *rejection key*. In Lemma 4.10, we state a useful, informal lemma that establishes how an implicitly rejecting KEM has to compute its rejection keys.

Lemma 4.10. *The rejection key computation of an implicitly rejecting KEM has to at least contain a secret random value and the rejected ciphertext.*

We do not give a formal proof of this statement. Instead, we argue informally why for any KEM that does not compute its rejection keys in this manner, an adversary can actually distinguish the rejection keys from a random key, turning them into an error flag. Note that this does not indicate a problem with IND-CCA, as IND-CCA only requires "accepting" keys to be indistinguishable from a random key.

Recall that Decaps(sk, pk, ct) is a deterministic algorithm. Therefore, sk, pk, and ct are the only possible inputs to the rejection key computation, as Decaps cannot sample random values. Notice that if the computation only contains pk or ct, the adversary can easily compute the same rejection key since only public values were used for the computation. Thus, a secret random value z has to be part of the computation.

Due to Decaps's deterministic nature, KEM designers are now left with two choices: Modify the decapsulation API to include z directly or let the secret key sk contain z. Depending on the choice made here and the origin of z (is it randomly sampled independent or dependent of sk?), it has implications on the achievable binding properties which we will discuss in Section 7.

Lastly, we point out that the ciphertext needs to be part of the rejection key computation, as otherwise there will be collisions: The rejection key would be the same for every ciphertext since sk and pk are static. One might wonder whether the statement is no longer true if we allow for a probabilistic Decaps. We argue that the statement still holds. If Decaps were to sample random, instead of using *static* randomness contained in the secret key, different queries with the same secret key, public key, and ciphertext would result in different rejection keys.

Modifying the hierarchy

A trivial side effect of implicitly rejecting KEMs is that the ciphertext alone cannot bind any other value: Any ciphertext will be accepted, and these will (with overwhelming probability) decapsulate to different

keys. A special case of this is the observation in [29] that an implicitly-rejecting KEM cannot satisfy SROB, i.e., *HON-BIND-CT-PK*.

Theorem 4.11. *An implicitly rejecting key encapsulation scheme* KEM *cannot satisfy X-BIND-CT-PK or X-BIND-CT-K for* $X \in \{HON, LEAK, MAL\}$.

Proof. We now show that:

1. KEM cannot be *HON-BIND-CT-K*-secure.
2. KEM cannot be *HON-BIND-CT-PK*-secure.

The analogous statements for the malicious case then follow by the contraposition of Lemma 4.3 and Lemma 4.2.

1. We construct an adversary $\mathcal{A}$ against *HON-BIND-CT-K*. On input $(\mathsf{pk}_0, \mathsf{pk}_1)$, $\mathcal{A}$ creates a valid ciphertext by encapsulating against one public key and returns this ciphertext. Thus, one decapsulation call in the *HON-BIND-CT-K* game yields a key that is indistinguishable from true randomness since KEM is IND-CCA-secure. With overwhelming probability, the other decapsulation call will return a rejection key that is computed from a secret random value and the ciphertext. Thus, if H is a random oracle, we can use the generic birthday bound to bound the probability that the valid key and the rejection key of the two decapsulation operations collide is negligible. Thus, $k_0 \neq k_1$ with overwhelming probability, and $\mathcal{A}$ wins the *HON-BIND-CT-K* game; KEM is not *HON-BIND-CT-K*-secure.
2. We construct an adversary $\mathcal{A}$ against *HON-BIND-CT-PK*. As KEM is implicitly rejecting, Decaps will always return a key $k \neq \perp$. Hence $\mathcal{A}$ chooses two different public keys and an arbitrary value for the ciphertext. Since all $\mathcal{A}$ needs to achieve is Decaps returning a key that is not equal to $\perp$, $\mathcal{A}$ trivially wins the game *HON-BIND-CT-PK*.

□

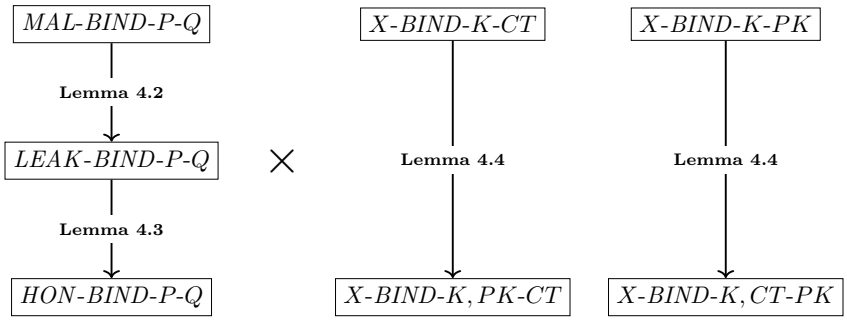

Figure 9: Restricted hierarchy of binding properties for implicitly rejecting KEMs, as *X-BIND-CT-K* and *X-BIND-CT-PK* cannot be met by any implicitly rejecting KEM

Thus, for implicitly rejecting KEMs, we have a reduced hierarchy of relevant properties. The separation between honest and malicious variants persists, but only four core properties are relevant and distinct, which are the key-binding properties. This leaves us with a simple hierarchy with only eight relevant binding properties overall for implicitly rejecting KEMs, which we visualize in Figure 9.

5 Symbolic Analysis of KEMs

We now turn to our second main objective: to develop a formal analysis framework for automatically analyzing security protocols that use KEMs. Our framework is rooted in the *symbolic model* of cryptography. In this model, cryptographic messages are represented as abstract terms from a fixed algebra, such as $\mathsf{sign}(m, sk)$ for message m signed with secret key sk. Equational theories over terms are used to encode properties of cryptographic primitives, like the signature verification equation $\mathsf{verify}(\mathsf{sign}(m, sk), m, \mathsf{pk}(sk)) = \mathsf{true}$.

The symbolic model gained prominence, especially with the advent of tools like the Tamarin Prover [44] and ProVerif [10] enabling automated security analysis of complex security protocols.

We start by examining existing symbolic models of KEMs, used in the analysis of security protocols with tools such as Tamarin Prover and ProVerif, exploring the extent to which they satisfy our specified properties. We then introduce Tamarin in more detail, as we will use it to implement our framework. Tamarin is one of the state-of-the-art analysis tools, which has demonstrated its efficacy with similar approaches [16, 33]. Lastly, we create new fine-grained symbolic models for KEMs that allow us to configure which cryptographic properties they have, allowing us to precisely model real KEM schemes in the symbolic model.

5.1 Previous Symbolic KEM Models

We are not the first to model KEMs in the symbolic model. Some KEM-based protocols, e.g., KEMTLS [45, 46, 48], post-quantum Wireguard [32], and PQXDH [8], were recently analyzed in the symbolic model. In this section, we investigate how these case studies model KEMs, which of our binding properties these models achieve, and why a class of symbolic models for KEMs implicitly assumes certain binding properties, highlighting the need for a new symbolic model that can model any combination of our binding properties.

In [48], the authors create and analyze two Tamarin models of KEMTLS, a variant of TLS that uses KEMs to achieve post-quantum security. Similarly, [32] uses Tamarin to analyze a KEM-based variant of Wireguard. The authors of [8] use another tool, ProVerif, to analyze the PQXDH protocol. In all of these case studies, function symbols and equational theories are used to model the behavior of KEMs. In fact, all but one model from [48] use the built-in equational theories for public key encryption to model KEMs. Concretely, they use the function symbols $\mathsf{aenc}/2$, denoting encryption, and $\mathsf{adec}/2$, denoting decryption. The function symbols are related through the equation $\mathsf{adec}(sk, \mathsf{aenc}(pk(sk), msg)) = msg$ and mapped to the standard KEM API in the following way:

$$\mathsf{Encaps}(pk(sk), r) = \mathsf{aenc}(pk(sk), r)$$
$$\mathsf{Decaps}(sk, ct) = \mathsf{adec}(sk, ct)$$

Note that Encaps does not return a tuple of ciphertext and key but only the ciphertext. Instead, r directly serves as the output key. One model from [48] uses a slightly different approach: instead of using r directly as the output key, they incoporate the receiver's public key and model the output key as a function $\mathsf{kdf}(r, pk)$. These models for KEMs are not surprising, since PKEs and KEMs are also strongly related in the computational model.

However, unlike the computational model, the above symbolic models encode much stronger assumptions on KEMs than just IND-CCA. Because the output key and the ciphertext are deterministic functions of pk and r, they bind these values by construction. That is, given an output key (or a ciphertext), the corresponding public key and randomness are uniquely determined. In fact, the reverse is also true. Thus, any symbolic model that computes, for instance, the ciphertext as $ct = \mathsf{Encaps}(pk, r)$ is *MAL-BIND-CT-PK*, *MAL-BIND-CT-K* (assuming k is also a function of pk and r), and *MAL-BIND-K*, *PK-CT* by construction.

As a result, the second KEM model from [48] implicitly assumes that the KEM satisfies all of our *MAL* binding properties. This means they cannot detect, e.g., re-encapsulation attacks. On the other hand, the first model from [48] and the models from [8, 32] do not assume *HON-BIND-K-PK* or *HON-BIND-K-CT*, because the output key is independent of the public key. Consequently, this model might detect some of our re-encapsulation attacks, which the findings of [8] confirm.

5.2 The Tamarin prover

Tamarin [44] is a tool for the automated analysis of protocols in the symbolic setting. It takes a protocol description in a custom modeling language and security properties specified in a fragment of first-order logic as input. The modeling language allows a user to specify the protocol rules and the adversary's capabilities via *multiset rewriting rules*. These rules induce a labeled transition system. Tamarin then tries to verify whether the given security properties hold for all traces of the transition system. We will now give more background on some features of the Tamarin prover that are necessary to understand the remainder of the paper.

A multiset of *facts* serves as the state of the labeled transition system. The rewriting rules manipulate this state by adding and removing facts. Facts are special user-defined symbols that contain terms and represent the state of the protocol. The state of the adversary (i.e., their knowledge) is modeled by a

distinct set of facts. An example of a fact would be $\mathsf{Alice}(\mathsf{pk}, \mathsf{sk})$, which models Alice who is in possession of some key pair $(\mathsf{pk}, \mathsf{sk})$.

A labeled multiset rewriting rule looks as follows:

$$[\mathsf{Alice}(\mathsf{pk}, \mathsf{sk}, \mathsf{ct}), !\mathsf{KeyValues}(\mathsf{k})] \quad (1)$$
$$-[\mathsf{Decaps}(\mathsf{k}, \mathsf{ct}, \mathsf{pk}, \mathsf{sk})]\rightarrow \quad (2)$$
$$[\mathsf{Out}(\mathsf{k})] \quad (3)$$

The lines (1), (2), and (3) contain multisets of facts called *premises*, *actions*, and *conclusions*, respectively. Facts annotated with a *!* are called *persistent* and are not removed from the multiset when a rule is executed. A rule can be executed in a given state if the premises are a subset of the current state. To execute the rule, Tamarin removes the premises from the state and adds the conclusions to it.

The execution of the protocol starts with the empty multiset as state and uses the rules to transition from one state to another. Rules can be used any number of times. The resulting sequence of actions is called a *trace*.

The user can specify formulas in a fragment of first-order logic that features quantification over terms and timepoints. In these formulas, the user can refer to the actions of the protocol and specify security properties. We write an action F at a timepoint $\#t$ as $\mathsf{F}(terms)@\#t$. The first-order logic fragment features the usual boolean connectives, ordering and equality of timepoints.

For our work, we also make use of Tamarin's *restrictions*. Restrictions are formulas like the security properties, but they are used to constrain the execution of the protocol: if a trace violates any restriction, Tamarin immediately discards it. Commonly, restrictions are used to model the conditional execution of rules based on the equality of terms or to ensure that certain rules are executed only once. However, recent work [16,33] has used restrictions to create event-based models of cryptography that directly encode the properties of cryptographic primitives as defined by their security definitions, only disallowing behavior that is explicitly forbidden by the security notion. We will follow a similar approach to create our own symbolic models for KEMs.

5.3 Improved Symbolic Model for KEMs

In Section 5.1, we investigated which of our binding properties are met by existing symbolic models for KEMs, which properties are not met, and which properties are always implicitly assumed by function-symbol based models. While the approaches used in [8,32,45,46] can capture some specific KEM properties, they can not model most combinations of our binding properties.

In this section, we rectify this by developing a new symbolic model for KEMs that allows the user to specify exactly which binding properties the model gives. Like the *Symbolic Verification of Signatures* model from [33], our model achieves this by only relying on the implications that the standard computational security definitions, e.g, IND-CCA, and our binding properties give, which makes it perfectly suited for verification.

Specification We observe that the definitions of correctness and IND-CCA only hold when the key pair that is used is honestly generated. When this is not the case, no guarantees are given. Correctness requires that the decapsulation of a ciphertext created by encapsulating against an honest public key returns the same output key for both algorithms. IND-CCA requires that the output key, created by encapsulating against an honest public key, is indistinguishable from true randomness (even when the adversary has access to a decryption oracle). Additionally, we note that Encaps can be a probabilistic algorithm, while Decaps is deterministic.

We now build a symbolic model that follows these constraints but allows for any other behavior. To do so, we model the key- and ciphertext space of a KEM and allow the adversary to choose arbitrary values from them as the result of Encaps and Decaps, as long as they respect the following constraints:

1. If the public key was honestly generated, an Encaps computation must return a fresh key different from any other Encaps computation.
2. If the public key was honestly generated, Encaps and Decaps computations using the same ciphertext and public key pair must return the same output key.
3. Given X-*BIND-P-Q*, any pair of calls to Decaps (and/or Encaps if $X = MAL$) must agree on Q if the parameters in P are equal.

4. Multiple computations of Decaps with the same inputs give the same result.
5. Any Encaps computation by the adversary only results in fresh keys or known values from the key space.

Constraints 1) and 2) model IND-CCA and correctness respectively, 3) ensures that the relevant binding properties are met, 4) makes Decaps deterministic, and 5) models the adversary computing a *derandomized* Encaps as seen in Section 3. We overapproximate this by allowing the adversary to let Encaps result in any element of the key space if they were already aware of this value, as letting the adversary choose any value would break IND-CCA.

Additionally, we add an option that, when enabled, specifies that Encaps and Decaps only work on honestly generated key pairs, rejecting any other values. This allows us to prevent the adversary from breaking IND-CCA and correctness by feeding *bogus* values into Encaps and Decaps. To achieve this, we require that a user of our library annotates rules in which the protocol honestly generates a public key pk with an action $\mathsf{GoodKey}(pk)$.

Due to space limitations, we only give a brief example of how our KEM model works. Recall the previous multiset rewriting rule. It shows how Alice, who possesses a public key pk, a secret key sk, and a ciphertext ct, decapsulates with these values to obtain key k. As long as the semantic constraints of our KEM model are fulfilled, the key can be an arbitrary value from the key space, represented by !KeyValues. Note that the key space only contains atomic values, which allows our KEM model to avoid achieving certain binding properties, e.g., *MAL-BIND-CT-PK*, by construction, which was not possible for previous symbolic models (see Section 5.1).

Definition 5.1. *HON-BIND-K-PK restriction*

$$\begin{aligned}&\forall\ k\ ct1\ ct2\ pk1\ pk2\ sk1\ sk2\ \#i\ \#j\ \#l\ \#m\ .\\&\mathsf{Decaps}(k, ct1, pk1, sk1)@\#i \wedge\ \mathsf{Decaps}(k, ct2, pk2, sk2)@\#j\\&\wedge\ \mathsf{GoodKey}(pk1)@\#l \wedge \mathsf{GoodKey}(pk2)@\#m\\&\Rightarrow\ (pk1 = pk2)\end{aligned}$$

In Definition 5.1, we show an example of how we formulate our binding properties as restrictions in Tamarin. For a more detailed description of our implementation, we refer the reader to Appendix C.

6 Case Studies

Model	#Lemmas Secrecy	Auth.	Auxiliary	#Tamarin calls	KEM-Binding Dependent Protocol Properties	Runtime for Initial Configurations
Onepass AKE	5	4	3	48	Implicit Key Authentication (Init.)	∼ 1m
Σ_0'-protocol	3	6	0	36	Implicit Key Authentication (Init.), SK-security	∼ 6m
PQ-SPDM	6	12	8	104		∼ 38m
Kyber-AKE	4	2	7	52	Implicit Key Authentication (Init., Resp.)	∼ 4h10m

Table 2: Summary of the analysis for our initial configurations. For the listed protocol properties, Tamarin returns different verification results in our initial configurations. For the minimal binding properties required to prove them, we refer to Table 3.

This section showcases the practicality of our approach through case studies. We begin with a brief overview of the evaluation methodology applied to evaluate the various Authenticated Key Exchange (AKE) protocols we modelled as case studies. In Section 6.2, we summarize the outcome of the chosen case studies. As case studies, we cover diverse post-quantum AKE protocols from the literature like the Kyber-AKE [12] or PQ-SPDM [50, 51], detailed in Sections 6.3 to 6.6.

6.1 Methodology

Our novel KEM model allows us to reason about both (i) an adversary that is restricted to use honestly generated, valid public key pairs as required by the KEM scheme and (ii) arbitrary, potentially malicious keys. Together with the option to use any combination of our specified binding properties from Section 4, this leads to a high number of configurations. Thus, it is infeasible to analyze each security property of every case study with every configuration of our KEM model. To analyze the influence of our binding notions on a protocol's security properties, we develop a methodology that allows us to discover the minimal requirements on a KEM that are needed to prove the property, while pruning the search space.

Model	Protocol Property	Minimal Binding Properties	
One-pass AKE	Implicit Key Authentication (Init.)	*X-BIND-K-PK*	{ *X-BIND-K-CT*, *X-BIND-K*, *CT-PK* }
Σ_0'-perfect	Implicit Key Authentication (Init.)	*X-BIND-K-PK*	{ *X-BIND-K-CT*, *X-BIND-K*, *CT-PK* }
Σ_0'-protocol	Implicit Key Authentication (Init.)	*X-BIND-K-PK*	{ *X-BIND-K-CT*, *X-BIND-K*, *CT-PK* }
	SK-Security	*HON-BIND-CT-K*	
		MAL-BIND-CT-K	*MAL-BIND-CT-PK*
Kyber-AKE	Implicit Key Authentication (Init.)	*X-BIND-K-PK*	{ *X-BIND-K-CT*, *X-BIND-K*, *CT-PK* }
	Implicit Key Authentication (Resp.)	*X-BIND-K-PK*	{ *X-BIND-K-CT*, *X-BIND-K*, *CT-PK* }

Table 3: **Minimal binding properties required by Tamarin to prove the property of an AKE model. We omit the set brackets for singleton sets like *HON-BIND-CT-K*. We write *X-BIND-P-Q* without specifying *X* to indicate that this set of properties is a solution for $X = MAL$ when all keys are allowed and for $X = HON$ when only honestly generated keys are allowed.**

Initial Configuration Testing First, we analyze each statement of a protocol with the following initial configurations:

1. Only keys from KeyGen and no binding properties
2. Only keys from KeyGen and **all** leak binding properties
3. Any keys and no binding properties
4. Any keys and **all** malicious binding properties

If, for a specific property of a protocol, Tamarin terminates with the same result in each of these configurations, we conclude that the protocol gives this property independent of any binding properties of the KEM or maliciously generated key pairs.

Key-Based Inference If Tamarin falsifies a property when we allow malicious keys, we infer that the protocol indeed relies on honestly generated key pairs to give the property.

Binding Property Analysis In the event that Tamarin gives different results when we change the binding property, we proceed with a more in-depth analysis:

1. For both the *LEAK* and *MAL* setting, we construct a directed acyclic graph whose nodes correspond to the possible combinations of our binding properties. We add an edge from node u to node v iff the properties that correspond to u imply the properties that correspond to v.
2. To efficiently compute for which combinations of properties a given statement is valid, we explore the graph as follows: We pick an unexplored node randomly and try to verify the statement using the corresponding binding properties. If Tamarin proves the statement, we mark the node as proven and recursively mark all of its parent nodes as proven too. We know that Tamarin will also prove the statement for the parent nodes since their corresponding properties imply the properties that were sufficient for a proof. If Tamarin falsifies the statement we mark the node as falsified, and recursively mark all of its child nodes as falsified too. This result is also valid for the child nodes because the corresponding binding properties of these nodes are weaker, removing fewer traces from the model and thus allowing for the same counterexample. In the event of a timeout (30 minutes), we mark the node as timed out and continue.
3. Once all nodes are marked, we extract the nodes for which Tamarin could still verify the statement but for whose direct child nodes Tamarin cannot verify the statement. The properties corresponding to these nodes are the minimal binding properties Tamarin needs to verify the statement.

6.2 Discussion of Results

We ran our models on an Intel(R) Xeon(R) CPU E5-4650L 2.60GHz machine with 1TB of RAM and 4 threads per Tamarin call. The execution time of our full methodology was approximately ∼16h30m.

We summarize the results of our initial configuration (see Section 6.1) in Table 2. We can observe that the specified security properties of the one-pass AKE, Kyber-AKE, and Σ_0'-protocol do not solely rely on

IND-CCA but also on other binding properties of the KEM. Details regarding this can be found in the corresponding sections 6.3, 6.4, and 6.6.

In Table 3 we show the concrete binding properties the aforementioned protocols require of their KEMs to achieve the desired security properties.

Challenges The newly introduced symbolic definitions of Encaps and Decaps can now result in arbitrary values from the key- and ciphertext space instead of only compound terms built from their inputs–vastly increasing the size of the search space. As the default heuristics of Tamarin do not prioritize solving, e.g., Encaps goals, when exploring the search space, we additionally develop *proof tactics*, tailored towards our KEM model. These tactics are prioritizing goals related to the output key, the KEM secret keys, and key-derivation functions, as well as deprioritizing goals related to ciphertexts.

6.3 One-Pass AKE

As a starting point, we model a one-pass AKE based on the SIKE protocol from [30], which we show in Figure 10. Note that in this protocol the Recipient does not receive standard mutual authentication guarantees since it cannot verify any information from the Initiator. Thus, we focus on the properties the Initiator can achieve.

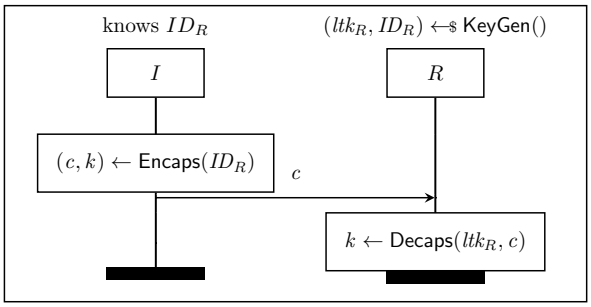

Figure 10: Simplified one-pass AKE.

In particular, we are interested in authentication properties like *Implicit Key Authentication* (as defined in Definition 6.1). However, in this protocol, neither the Initiator nor the Recipient can achieve it, since only one party contributes to the final key. Thus, we focus on a weaker, unilateral version, where only the identity of the Recipient has to match in both sessions, i.e., both Initiator and Recipient agree on the Recipient's identity when they derive the same shared key.

We find that the Initiator can achieve this weaker property when the protocol uses a KEM that is at least *X-BIND-K-PK*-secure. When the KEM does not have this binding property, the adversary can mount a re-encapsulation attack against the Initiator by leaking ltk_R of the Recipient and then re-encapsulating towards another Recipient, resulting in two Recipient sessions with the same key.

Definition 6.1. *Implicit Key Authentication Initiator*

$$\begin{aligned}&\forall\ id1\ id2\ pkI1\ pkI2\ pkR1\ pkR2\ k\ ct1\ ct2\ \#i\ \#j\ .\\ &\mathsf{FinishInitiator}(id1, pkI1, pkR1, k, ct1\ )@\#i\\ &\wedge\ \mathsf{FinishResponder}(id2, pkI2, pkR2, k, ct2\ )@\#j\\ &\Rightarrow\ (pkI1 = pkI2 \wedge pkR1 = pkR2)\end{aligned}$$

6.4 Σ_0'-protocol

The Σ_0'-protocol is introduced by [41] as a KEM-based variation of the Σ_0-protocol [15]. The original Σ_0-protocol is a component of the Internet Key Exchange (IKE) protocols [35], and Σ_0' was suggested as a post-quantum replacement. We provide a description of the Σ_0'-protocol in Figure 11. [41] claims that the Σ_0'-protocol is SK-secure in the post-specified peer model [15] for any IND-CCA KEM. We analyze whether Σ_0' achieves SK-security (Definition 6.3) and Implicit Key Authentication and Final Key Secrecy (Definition 6.2).

Definition 6.2. *Final Key Secrecy Initiator*

$$\forall\ id\ pkI\ pkR\ k\ ct\ \#i\ \#j\ .$$
$$\mathsf{FinishInitiator}(id, pkI, pkR, k, ct\ )@\#i \wedge \mathsf{GoodKey}(pkR)@\#j$$
$$\Rightarrow\ (\exists\ \#x\ .\mathsf{K}(k)@\#x) \vee (\exists\ \#x\ .\mathsf{RevealLTK}(pkR)@\#x)$$

Definition 6.3. *SK-Security*

$$\forall\ sid\ pkI\ pkR\ k\ k2\ \#i\ \#j\ .$$
$$\mathsf{FinishInit}(sid, pkI, pkR, k)@\#i \wedge \mathsf{FinishResp}(sid, pkI, pkR, k2)@\#j$$
$$\wedge\ not(\exists\ \#y\ .\mathsf{RevealLTK}(pkR)@\#y) \wedge not(\exists\ \#x\ .\mathsf{RevealLTK}(pkI)@\#x)$$
$$\Rightarrow\ (k\ = k2) \wedge not(\exists\ \#z\ .\mathsf{K}(k)@\#z)$$

When modeling Σ_0', we noticed two issues with the protocol description in [41]. First, the authors assume that the Responder can store an unlimited number of session identifiers it receives from the Initiator and that it only accepts sessions that use a new, unused identifier, which is notoriously hard to achieve. Second, after replying to the Initiator, the Responder should erase pk_I from its state. However, when the Responder receives the final message, it has to verify σ_I–which contains pk_I. It is unclear how the Responder can verify this signature after erasing pk_I.

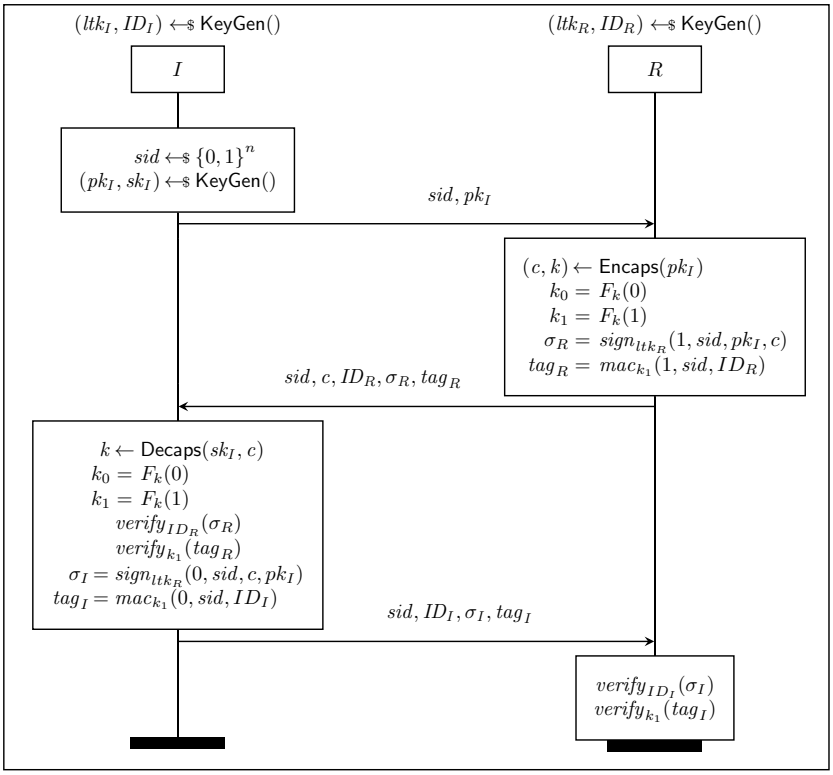

Figure 11: The Σ_0'-protocol introduced by [41].

To address these issues, we create two Σ_0' models. In the first model, Σ_0'-perfect, the Responder keeps pk_I in its state and verifies σ_I correctly, as well as only accepting the first message when it sees a new, fresh session identifier. The second model, Σ_0', does *not* keep pk_I in the Responder's state, and the verification of $\sigma_I = (0, sid, c, pk)$ succeeds for any KEM public key pk as long as 0, sid, and c are correctly signed. In this model, the Responder always replies to the first message, even if session identifiers repeat.

We find that Σ_0' and Σ_0'-perfect achieve Implicit Key Authentication and Final Key Secrecy for the Responder in all of our initial configurations. Additionally, we prove *Full Key Confirmation* for the Responder, which we define in Definition 6.4.

Definition 6.4. *Full Key Confirmation Responder*

$$\forall\ sid\ pkI\ pkR\ k\ epkI\ \#i\ \#j\ .$$
$$\mathsf{FinishResponder}(sid, pkI, pkR, k, epkI\ )@\#i \wedge \mathsf{GoodKey}(epkI)@\#j$$
$$\Rightarrow\ (\exists\ pkI2\ pkR2\ epkI2\ \#x\ .\mathsf{FinishInitiator}(sid, pkI2, pkR2, k, epkI2\ )@\#x)$$

As is the case for Kyber-AKE (Section 6.6), both Σ_0' and Σ_0'-perfect do not achieve Implicit Key Authentication for the Initiator for any IND-CCA-secure KEM. This is because the adversary can switch pk_I for their own ephemeral key and reveal ltk_R of the Responder. Let A and B be honest agents. The attack then proceeds as follows: the adversary waits until A initiates a session as initiator with peer B and starts another session impersonating as C towards B in the Responder role. After replacing pk_A with their own ephemeral KEM key pk_C and revealing ltk_B, the adversary forwards sid, pk_C to B, who acts according to the protocol. Then, the adversary decapsulates c to learn k, k_0, and k_1. Next, the adversary mounts a re-encapsulation attack against A's actual ephemeral key pk_A, resulting in a ciphertext c' that also decapsulates to key k. Since the adversary knows both lkt_B and k_1, they can forge B's signature on c' and create a valid tag_B, which they both forward to A, completing A's run. Finally, the adversary creates a valid signature and tag for B, who thinks they are communicating with the adversary. At the end of their respective runs, A and B agree on key k but not on their peers' identities.

We find that the protocol gives Implicit Key Authentication for the Initiator when the KEM satisfies at least *HON-BIND-K-PK* or both *HON-BIND-K-CT* and *HON-BIND-K, CT-PK*. Note that the later two, together, imply *HON-BIND-K-PK* by Corollary 4.9. Thus, *HON-BIND-K-PK* really is the property that prevents the attack: I and R deriving the same key k implies that they agree on pk_I. This stops the adversary from switching out pk_I for their own ephemeral key, which prevents them from learning the key k. However, knowledge of k is necessary to create a valid tag_I for R in the last message. Thus, the above attack is prevented.

We observe that Σ_0' does not achieve SK-security for any IND-CCA-secure KEM when the Responder erases pk_I from its state and accepts duplicate session identifiers. To understand this attack, we want to highlight that the signatures σ_I and σ_R only include c; they *do not* include the actual output key k. Thus, the adversary can choose c such that it decapsulates to different keys for the Initiator and the Responder. This behavior is not excluded by correctness of the KEM, since SK security does not enforce that Initiator and Responder agree on the ephemeral key pk_I; they only need to agree on their respective identities. Thus, the adversary can force a session between A and B where they agree on their identities but disagree on the ephemeral key pk_B. Together with the deficiencies of Σ_0', this allows for an attack on SK security, which can be prevented by using, for instance, a *MAL-BIND-CT-PK*-secure KEM.

The adversary starts a session between Initiator A and Responder B where they agree on their identities and pk_A. After receiving the first message, B computes σ_B and tag_B, which the adversary intercepts. Then, impersonating A, the adversary starts another session with B, where they switch out pk_A for another KEM public key but reuse the session identifier sid. Recall that the Responder only accepts this session in the Σ_0' protocol where we do not assume infinite, persistent storage at the Responder side. The Responder B again acts according to the protocol and computes $(c, k) \leftarrow \mathsf{Encaps}(pk_A)$. The adversary then forwards σ_B and tag_B to A, who computes σ_A, tag_A, and finishes their session with B decapsulating k' from c because of the lack of binding properties of KEM. The adversary then forwards these values to B in the session where they disagree on pk_A. Since B accepts any σ_B as long as c and sid match, they also finish their session with A computing k. As these sessions agree on the identities but disagree on the final shared key, SK security does not hold.

When honestly generated keys are used, the KEM must at least satisfy *HON-BIND-CT-K*, and, in the presence of maliciously generated keys, the KEM must at least satisfy *MAL-BIND-CT-PK* or *MAL-BIND-CT-K* to prevent this attack. These properties prevent Initiator and Responder from opening the same ciphertext to different output keys.

To summarize, we find that Σ_0' does not achieve Implicit Key Authentication for the Initiator when used with any IND-CCA-secure KEM due to a re-encapsulation attack, and that SK security does not hold when the Responder *misbehaves* as described previously. A KEM with additional binding properties, e.g., *HON-BIND-K-PK* and *HON-BIND-CT-K*, could have prevented these attacks.

6.5 PQ-SPDM

The Security Protocol and Data Model (SPDM) [23] is an emerging industry standard aimed at ensuring end-to-end trust in infrastructure, focusing on hardware and chip-to-chip communication. Although the standard is being developed by major industry players, there has been limited cryptographic analysis. Recent efforts using the Tamarin tool [22] have initiated formal analysis, but no formal cryptographic proof of its session establishment protocol has been conducted yet. Additionally, to address post-quantum security concerns, post-quantum versions of SPDM's session establishment have been proposed by [50, 51]. We model this proposed post-quantum variant and analyze for both Initiator and Responder whether the protocol achieves Final Key Secrecy, Implicit Key Authentication, and Full Key Confirmation. A

detailed description of the (post-quantum) SPDM protocol is out of scope for this paper; we refer the reader to [22, 50, 51].

We find that, for the Initiator, Final Key Secrecy holds as long as neither the ephemeral KEM key pair nor the long-term key of the Responder is revealed. For the Responder, we find that Final Key Secrecy holds even when the ephemeral KEM key pair is revealed. The Initiator obtains Implicit Key Authentication only if the ephemeral KEM key pair is not revealed, and the Responder as long as either the long-term key of the Initiator or the ephemeral KEM key pair is not revealed.

Full Key Confirmation does not hold for the Initiator if either key pair is revealed. For the Responder, we find that the property holds as long as at least one key pair is not revealed.

We find these results across all initial configurations and conclude that PQ-SPDM provides these guarantees independently of any KEM binding properties or maliciously generated keys.

6.6 Kyber-AKE

We model the Kyber-AKE (see Figure 2) in Tamarin and analyze the protocol, both in terms of secrecy and authentication properties.

The secrecy of the final key of the Kyber-AKE is guaranteed for both Initiator and Responder as long as the long-term secrets are not revealed. The property is defined analogously to Definition 6.2.

We also analyze Implicit Key Authentication for both the Initiator and Responder analogously to Definition 6.1. Without any additional binding properties, even when we restrict the adversary to only use honest keys out of KeyGen, Tamarin is able to produce a counterexample violating the defined property. We call this attack *re-encapsulation* attack, as described in Section 3.

We show that, in the setting where we do not allow keys outside of KeyGen, the used KEM in the Kyber-AKE additionally needs to provide *HON-BIND-K-PK* or both *HON-BIND-K-CT* and *HON-BIND-K, CT-PK* to guarantee implicit authentication for both parties. Analogously, in the stronger adversary model, the used KEM needs to provide *MAL-BIND-K-PK* or both *MAL-BIND-K-CT* and *MAL-BIND-K, CT-PK* to guarantee implicit authentication. Note that we could show with Corollary 4.9 that *X-BIND-K-CT* and *X-BIND-K, CT-PK* imply *X-BIND-K-PK*, confirming that one can prevent that attack by binding the public key to the final key.

This observation also confirms why we do not see this kind of attack in the original Kyber-AKE, as the Kyber KEM is conjectured to fulfill these properties (see Cons. B.1).

Note that [12] claims the Kyber-AKE is secure in the Canetti-Krawczyk (CK) model with weak forward secrecy [14]. However, they state no explicit proof and claim that this "*follows directly from the generic security bounds of [4, 17]*". Our results show that this statement is incorrect if the KEM is only IND-CCA secure. In particular, in the CK model with weak PFS, the re-encapsulation attack would imply that the sessions are not matching, and this would allow the adversary to reveal the session key at A's session.

7 Related Work

In this Section, we give a brief overview of related work. First, we investigate security notions for KEMs that go beyond IND-CCA like robustness, which are similar in spirit to our novel binding properties. Then, we introduce recent related work by Schmieg [49] and discuss how it relates to this paper.

7.1 Further security notions

While IND-CCA is still the main security notion for KEMs, additional security notions have been proposed.

The term "robustness" was initially coined by Abdalla, Bellare, and Neven in [1] in the context of PKE schemes. In a nutshell, robustness means it is hard to produce a ciphertext that is valid for two different key pairs (or users). They introduce both *weak* (WROB) and *strong* (SROB) robustness. In the weak robustness game, an adversary has to find a message m and two distinct public keys pk_0 and pk_1 such that encrypting m with pk_0 results in a valid ciphertext when decrypted with sk_1, pk_1's secret key. In the strong robustness game, the adversary has to find a ciphertext c and two distinct public keys such that c decrypts under both corresponding secret keys. This strengthens the adversary since c does not have to be the result of an *honest* encryption but could have been specifically created by the adversary.

In [26], the authors make a case for new, stronger robustness notions by showing how the notions of [1] fail to prevent attacks in certain applications such as fair auction protocols. First, they observe that the original strong robustness definition does not allow the adversary to query their oracle with the secret keys of the public keys they are challenged with; removing this restriction leads to an intermediate notion

that they call *unrestricted* strong robustness (USROB). Then, they go on to remove the restriction that the adversary is challenged with honestly generated public keys. Instead, the adversary is given complete control over the key generation, and it is up to the decryption algorithm to reject invalid key pairs, which leads to their *full* robustness (FROB) notion. The USROB and FROB notions define robustness via the decryption routine of a PKE, implicitly assuming that robustness "carries" over to the encryption algorithm since encryption and decryption are related through correctness. However, in a setting where the adversary can freely choose key pairs and ciphertexts, correctness may no longer hold, since the adversary can feed values from outside the key- and ciphertext-space into the PKE algorithms. Thus, it is necessary that the whole cryptosystem satisfies a robustness notion. To capture this, [26] defines *complete* robustness (CROB), which challenges the adversary to find a ciphertext that decrypts under different key pairs for any combination of encryption and decryption calls.

In [29], Grubbs, Maram, and Paterson define anonymity, robustness, and so-called *collision freeness* for KEMs, building upon Mohassel's work [38] that only defined these properties for PKEs. They investigate whether a PKE constructed via the KEM-DEM paradigm inherits anonymity and robustness from the underlying KEM. They show that this is true for explicitly rejecting KEMs. However, for implicitly rejecting KEMs, this is not the case in general. Since all NIST PQC finalist KEMs are implicitly rejecting KEMs constructed via variants of the FO transform [27], they then go on to analyze how the FO transform lifts robustness and anonymity properties from a PKE scheme, first to the KEM built via the FO transform and then to the hybrid PKE scheme obtained via the KEM-DEM paradigm. They apply their generic analysis of the FO transform to the NIST PQC finalists Saber [25], Kyber [12], and Classic McEliece [7] as well as the NIST alternate candidate FrodoKEM [11]. Another finding of [29] regarding the IND-CCA secure Classic McEliece scheme will be relevant for our work: for any plaintext m, they find that it is possible to construct a single ciphertext c that always decrypts to m under *any* Classic McEliece private key.

7.2 *MAL-BIND-P-Q* in the Wild

Building on an earlier pre-release of this paper, Schmieg reports in [49] attacks on ML-KEM's *MAL-BIND-K-PK* and *MAL-BIND-K-CT* security, which we conjectured in an earlier version (Version 1.0.5) of our paper. We have updated Table 5 accordingly.

The attack in [49] on *MAL-BIND-K-CT* exploits the fact that implementations of ML-KEM store a hash of the public key, which is needed to compute the shared secret, inside of the secret key. This is done to avoid recomputing this hash across decapsulation operations. While this increases the performance of the KEM, it allows strong adversaries that can control the secret key for decapsulation–like the adversary in our *MAL* properties–to replace the correct hash with a faulty one. In a nutshell, replacing this hash does not trigger the FO rejection flow, and the different stored hashes of the public key lead to different shared secret. For further details of the attack, we refer the reader to [49]. We want to highlight that this attack is neither specific to ML-KEM nor to implicitly rejecting KEMs (since it does not trigger the rejection flow) but caused by the serialization format of the secret key.

To mitigate the attack, [49] suggests to not cache the hash of the public key inside of the secret key but to recompute it for every decapsulation operation. Another mitigation they suggest is to check whether the stored hash is actually the hash of the public key. In summary, both of these mitigation strategies assert that the secret key is wellformed. Therefore, we believe that KEM implementations can achieve the *MAL-BIND-K-CT* property in practice, albeit at the cost of a minor performance loss, and the attack does not indicate a problem with our property.

The attack on ML-KEM's *MAL-BIND-K-PK* is very similar to the previously described attack on *MAL-BIND-K-CT*. The difference is that the adversary now replaces the rejection value z, which is also stored inside the secret key. The attack then proceeds as follows: The adversary creates two secret keys which share a rejection value z, produces a random ciphertext c, and tries to decapsulate c with both secret keys. With overwhelming probability, both decapsulation calls will reject the ciphertext resulting in the same shared secret because the rejection flow computes the shared secret as a hash of c and z.

Again, the mitigation that Schmieg suggests tries to verify that the secret key is indeed wellformed. The idea is to not store the secret key directly, but to store the seed that the key was derived from and to recompute the key in every decapsulation call. Unfortunately, this mitigation is impossible to implement for current implicitly rejecting KEM proposals (e.g., ML-KEM) without technical changes, since they currently derive the rejection value z from a different seed that is independent of the key generation seed. Because of this independence, it is impossible to check whether a secret key is wellformed: the rejection value z stored inside of the key is random and independent of the key; any value is possible.

As a consequence, there seems to be no mitigation for Schmieg's attack without changing the current proposals, and our *MAL-BIND-K-PK* and *MAL-BIND-K, CT-PK* properties seem to be unachievable for implicitly rejecting KEMs with such independent values. If, instead, the key pair and rejection value are derived from the same seed, a KEM can verify whether a secret is wellformed by recomputing it from the seed. However, this change might have security implications and will at least partially invalidate existing analysis of implicitly rejecting KEMs which assume independence of these values.

To conclude, we find that the attack on *MAL-BIND-K-CT* is tightly linked to the additional information that is stored inside of the secret key, but KEM implementations can achieve the property if they check the secret key for wellformedness. However, we are not aware of any current KEM implementations that do this due to its negative performance implications. Regarding the attack on *MAL-BIND-K-PK*, we find that it is tightly linked to the relationship of the key pair's seed and the rejection value's seed. If they are independent, there appears to be no way of mitigating this attack since any combination of key pair and rejection value is wellformed. If they are dependent, checking for wellformedness becomes possible, however, this would require changes to current proposals as well as their implementations, and might invalidate existing analysis which assume that these values are independent.

8 Conclusion

We introduced a new family of security notions for KEMs, that capture relevant binding properties, and we establish a hierachy within them. We show how our notions capture existing binding properties like Robustness. KEM schemes that meet our binding properties are harder to misuse, as they leave fewer pitfalls for protocol designers.

We develop a novel symbolic KEM model for the Tamarin prover and an analysis methodology that allows us to automatically find the minimum binding properties a protocol requires of a KEM to meet its security properties. We evaluate our KEM model in case studies and find several new attacks and missed proof obligations. Notably, we find a new type of attack, which we call "re-encapsulation attack".

Our work is in line with a wider trend of constructing cryptographic primitives that are harder to misuse and offer cleaner behavior with fewer side-cases. In particular, the guarantees offered by our properties perform a similar role as exclusive ownership and message-binding properties of digital signatures and the various robustness notions defined for authenticated encryption schemes and public key encryption schemes.

As migration to post-quantum secure protocols progresses in the next years, we expect many existing protocol designs to be adapted to use KEMs. Our symbolic protocol analysis approach supports this transition by attack finding, verification, and requirement discovery.

Acknowledgements The authors would like to thank Deirdre Connolly and Sophie Schmieg for input and discussion.

References

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust Encryption. Cryptology ePrint Archive, Paper 2008/440, 2008. https://eprint.iacr.org/2008/440.

[2] Martin Albrecht, Carlos Cid, Kenneth G Paterson, Cen Jung Tjhai, and Martin Tomlinson. NTS-KEM. *NIST PQC Second Round*, 2, 2019. https://nts-kem.io/ (Accessed December 2023).

[3] Nicolas Aragon, Paulo SLM Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. BIKE: bit flipping key encapsulation. 2017.

[4] Atsushi Fujioka and Koutarou Suzuki and Keita Xagawa and Kazuki Yoneyama. Strongly Secure Authenticated Key Exchange from Factoring, Codes, and Lattices. Cryptology ePrint Archive, Paper 2012/211, 2012. https://eprint.iacr.org/2012/211.

[5] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karoline Varner, and Bas Westerbaan. X-Wing: The Hybrid KEM You've Been Looking For. Cryptology ePrint Archive, Paper 2024/039, 2024. https://eprint.iacr.org/2024/039.

[6] Mihir Bellare, Hannah Davis, and Felix Günther. Separate Your Domains: NIST PQC KEMs, Oracle Cloning and Read-Only Indifferentiability. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 3–32, Cham, 2020. Springer International Publishing.

[7] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic McEliece: conservative code-based cryptography. *NIST submissions*, 2017.

[8] Karthikeyan Bhargavan, Charlie Jacomme, Franziskus Kiefer, and Rolfe Schmidt. An Analysis of Signal's PQXDH, Oct 2023. `https://cryspen.com/post/pqxdh/` (Accessed Jan 2024).

[9] Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (STS) protocol. In *Public Key Cryptography: Second International Workshop on Practice and Theory in Public Key Cryptography, PKC'99 Kamakura, Japan, March 1–3, 1999 Proceedings 2*, pages 154–170. Springer, 1999.

[10] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[11] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016.

[12] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[13] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of ed25519: theory and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1659–1676. IEEE, 2021.

[14] Ran Canetti and Hugo Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. Cryptology ePrint Archive, Paper 2001/040, 2001. `https://eprint.iacr.org/2001/040`.

[15] Ran Canetti and Hugo Krawczyk. Security Analysis of IKE's Signature-based Key-Exchange Protocol. Cryptology ePrint Archive, Paper 2002/120, 2002. `https://eprint.iacr.org/2002/120`.

[16] Vincent Cheval, Cas Cremers, Alexander Dax, Lucca Hirschi, Charlie Jacomme, and Steve Kremer. Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses. In *USENIX 2023*, 2023.

[17] Colin Boyd and Yvonne Cliff and Juan M. Gonzalez Nieto and Kenneth G. Paterson. Efficient One-round Key Exchange in the Standard Model. Cryptology ePrint Archive, Paper 2008/007, 2008. `https://eprint.iacr.org/2008/007`.

[18] Ronald Cramer and Victor Shoup. Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. Cryptology ePrint Archive, Paper 2001/108, 2001. `https://eprint.iacr.org/2001/108`.

[19] Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. Automated Analysis of Protocols that use Authenticated Encryption: Analysing the Impact of the Subtle Differences between AEADs on Protocol Security. In *USENIX 2023*, 2023.

[20] Cas Cremers, Alexander Dax, and Niklas Medinger. KEM library and Case Studies. `https://github.com/FormalKEM/Symbolic_KEM_Models`, January 2024.

[21] Cas Cremers, Samed Düzlü, Rune Fiedler, Marc Fischlin, and Christian Janson. BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.

[22] Cremers, Cas and Dax, Alexander and Naska, Aurora. Formal Analysis of SPDM: Security Protocol and Data Model version 1.2. In *USENIX 2023*, 2023.

[23] DMTF. DSP0274: Security Protocol and Data Model (SPDM) Specification, Version 1.3.0. https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.3.0.pdf, May 2023. accessed: 2024-01-26.

[24] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference*, 2018.

[25] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In *Progress in Cryptology–AFRICACRYPT 2018: 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7–9, 2018, Proceedings 10*, pages 282–305. Springer, 2018.

[26] Pooya Farshim, Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Robust Encryption, Revisited. Cryptology ePrint Archive, Paper 2012/673, 2012. https://eprint.iacr.org/2012/673.

[27] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*, 1999.

[28] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message Franking via Committing Authenticated Encryption. Cryptology ePrint Archive, Paper 2017/664, 2017. https://eprint.iacr.org/2017/664.

[29] Paul Grubbs, Varun Maram, and Kenneth G. Paterson. Anonymous, Robust Post-Quantum Public Key Encryption. Cryptology ePrint Archive, Paper 2021/708, 2021. https://eprint.iacr.org/2021/708.

[30] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A Modular Analysis of the Fujisaki-Okamoto Transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 341–371, Cham, 2017. Springer International Publishing.

[31] Andreas Hülsing, Joost Rijneveld, John Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 232–252. Springer, 2017.

[32] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Philip R. Zimmermann. Post-quantum WireGuard. Cryptology ePrint Archive, Paper 2020/379, 2020. https://eprint.iacr.org/2020/379.

[33] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. Seems legit: Automated analysis of subtle attacks on protocols that use signatures. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2165–2180, 2019.

[34] Joël Alwen and Bruno Blanchet and Eduard Hauck and Eike Kiltz and Benjamin Lipp and Doreen Riepel. Analysing the HPKE Standard. Cryptology ePrint Archive, Paper 2020/1499, 2020. https://eprint.iacr.org/2020/1499.

[35] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference*, Lecture Notes in Computer Science, 2003. Invited paper.

[36] Julia Len, Paul Grubbs, and Thomas Ristenpart. Authenticated Encryption with Key Identification. Cryptology ePrint Archive, Paper 2022/1680, 2022. https://eprint.iacr.org/2022/1680.

[37] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. Hamming quasi-cyclic (HQC). *NIST PQC Round*, 2018.

[38] Payman Mohassel. A Closer Look at Anonymity and Robustness in Encryption Schemes. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *Lecture Notes in Computer Science*, pages 501–518. Springer, 2010.

[39] NIST. NIST Post-Quantum Cryptography. https://csrc.nist.gov/projects/post-quantum-cryptography. Accessed: 2024-01-16.

[40] NIST. Module-Lattice-Based Key-Encapsulation Mechanism Standard (Initial Public Draft). https://csrc.nist.gov/pubs/fips/203/ipd, 2023. Accessed: 2024-02-23. Date Published: August 24, 2023.

[41] Chris Peikert. Lattice Cryptography for the Internet. Cryptology ePrint Archive, Paper 2014/070, 2014. https://eprint.iacr.org/2014/070.

[42] Thomas Pornin and Julien P Stern. Digital signatures do not guarantee exclusive ownership. In *Applied Cryptography and Network Security: Third International Conference, ACNS 2005*, 2005.

[43] Richard Barnes and Karthikeyan Bhargavan and Benjamin Lipp and Christopher A. Wood. Hybrid Public Key Encryption. RFC 9180, February 2022.

[44] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 78–94. IEEE, 2012.

[45] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. Cryptology ePrint Archive, Paper 2020/534, 2020. https://eprint.iacr.org/2020/534.

[46] Peter Schwabe, Douglas Stebila, and Thom Wiggers. More efficient post-quantum KEMTLS with pre-distributed public keys. Cryptology ePrint Archive, Paper 2021/779, 2021. https://eprint.iacr.org/2021/779.

[47] Victor Shoup. A Proposal for an ISO Standard for Public Key Encryption. *IACR Cryptology ePrint Archive*, 2001:112, 2001.

[48] Sofía Celi and Jonathan Hoyland and Douglas Stebila and Thom Wiggers. A tale of two models: formal verification of KEMTLS via Tamarin. Cryptology ePrint Archive, Paper 2022/1111, 2022. https://eprint.iacr.org/2022/1111.

[49] Sophie Schmieg. Unbindable Kemmy Schmidt: ML-KEM is neither MAL-BIND-K-CT nor MAL-BIND-K-PK. Cryptology ePrint Archive, Paper 2024/523, 2024. https://eprint.iacr.org/2024/523.

[50] Jiewen Yao, Anas Hlayhel, and Krystian Matusiewicz. Post Quantum KEM authentication in SPDM for secure session establishment. *Design & Test*, 2023.

[51] Jiewen Yao, Krystian Matusiewicz, and Vincent Zimmer. Post Quantum Design in SPDM for Device Authentication and Key Establishment. *Cryptography*, 6(4):48, 2022.

A Proofs for the Properties of Our Generic Binding Notions

Lemma A.1. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. If* KEM *is MAL-BIND-P-Q-secure, then* KEM *is also LEAK-BIND-P-Q-secure.*

Proof. We prove the contraposition. Let $\mathcal{A}$ be an adversary against $\mathit{LEAK\text{-}BIND\text{-}P\text{-}Q}_{\mathcal{A}}^{\mathsf{KEM}}$. We construct an adversary $\mathcal{B}$ against $\mathit{MAL\text{-}BIND\text{-}P\text{-}Q}_{\mathcal{B}}^{\mathsf{KEM}}$. $\mathcal{B}$ generates two key pairs honestly, chooses $g = 1$, and then calls $\mathcal{A}$ with these key pairs. $\mathcal{B}$ returns both key pairs and the ciphertext that $\mathcal{A}$ returned to win $\mathit{MAL\text{-}BIND\text{-}P\text{-}Q}_{\mathcal{B}}^{\mathsf{KEM}}$ with the same non-negligible probability that $\mathcal{A}$ wins $\mathit{LEAK\text{-}BIND\text{-}P\text{-}Q}_{\mathcal{A}}^{\mathsf{KEM}}$ with. □

Lemma A.2. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. If* KEM *is LEAK-BIND-P-Q-secure, then* KEM *is also HON-BIND-P-Q-secure.*

Proof. The proof is a straightforward reduction that is analogous to Lemma A.1. □

Lemma A.3. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. For* $X \in \{\mathit{MAL}, \mathit{LEAK}, \mathit{HON}\}$*, if* KEM *is X-BIND-P-Q′-secure and* $P \subseteq P' \wedge Q \subseteq Q'$*, then* KEM *is also X-BIND-P′-Q-secure.*

Proof. Assume KEM is X-BIND-P-Q'-secure and $P \subseteq P' \wedge Q \subseteq Q'$. Now assume towards contradiction that KEM is not X-BIND-P'-Q-secure. Thus, there exists an adversary $\mathcal{A}$ that wins the X-BIND-P'-$Q_{\mathcal{A}}^{\mathsf{KEM}}$ game with non-negligible probability. From $\mathcal{A}$, we now build an adversary $\mathcal{B}$ that wins the X-BIND-P-$Q'_{\mathcal{B}}{}^{\mathsf{KEM}}$ game with non-negligible probability. $\mathcal{B}$ simply calls $\mathcal{A}$ to win X-BIND-P-$Q'_{\mathcal{B}}{}^{\mathsf{KEM}}$ with the same probability that $\mathcal{A}$ wins X-BIND-P'-$Q_{\mathcal{A}}^{\mathsf{KEM}}$. We now argue why the *winning condition*

$$\forall x \in P' \,.\, x_0 = x_1 \wedge \exists y \in Q \,.\, y_0 \neq y_1$$

of $\mathcal{A}$ in X-BIND-P'-$Q_{\mathcal{A}}^{\mathsf{KEM}}$ implies the winning condition

$$\forall x \in P \,.\, x_0 = x_1 \wedge \exists y' \in Q' \,.\, y'_0 \neq y'_1$$

of $\mathcal{B}$ in X-BIND-P-$Q'_{\mathcal{B}}{}^{\mathsf{KEM}}$. Since $P \subseteq P'$, each equality constraint in P is also present in P' and thus satisfied. Since $Q \subseteq Q'$, we can use the witness y from Q as witness y' for Q' to satisfy the existential constraint.

□

Theorem A.4. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. For* $X \in \{\mathit{MAL}, \mathit{LEAK}, \mathit{HON}\}$*, if* KEM *is X-BIND-P-Q′-secure, X-BIND-Q-R′-secure and* $P \subseteq P'$*,* $Q \subseteq Q' \cup P'$*, and* $R \subseteq R'$*, then* KEM *is also X-BIND-P′-R-secure.*

Proof. Assume KEM is X-BIND-P-Q'-secure, X-BIND-Q-R'-secure as well as $P \subseteq P'$, $Q \subseteq Q' \cup P'$, and $R \subseteq R'$. Now assume towards contradiction that KEM is not X-BIND-P'-R-secure. Therefore, there exists an adversary $\mathcal{A}$ that wins the X-BIND-P'-$R_{\mathcal{A}}^{\mathsf{KEM}}$ game with non-negligible probability. To win the game, $\mathcal{A}$ produces parameter sets P' and R such that

$$\forall p \in P' \,.\, p_0 = p_1 \wedge \exists r \in R \,.\, r_0 \neq r_1.$$

Since KEM is X-BIND-P-Q'-secure and $P \subseteq P'$, it follows that $\forall q \in Q' \,.\, q_0 = q_1$. Since KEM is X-BIND-Q-R'-secure and $Q \subseteq Q' \cup P'$, it follows that $\forall r \in R' \,.\, r_0 = r_1$. This implies $\forall r \in R \,.\, r_0 = r_1$ as $R \subseteq R'$, which contradicts the adversary producing R such that $\exists r \in R \,.\, r_0 \neq r_1$. Therefore, KEM is X-BIND-P'-R-secure. □

A.0.1 Relations between our Properties

Lemma A.5. *Let* KEM *be a KEM that is HON-BIND-CT-PK secure. Then* KEM *is also HON-BIND-CT-K secure.*

Proof. (Sketch) We prove the contraposition. Let $\mathcal{A}$ be an adversary against $\mathit{HON\text{-}BIND\text{-}CT\text{-}K}_{\mathcal{A}}^{\mathsf{KEM}}$. We construct an adversary $\mathcal{B}$ against *HON-BIND-CT-PK*. $\mathcal{B}$ forwards the public keys it is challenged with to $\mathcal{A}$ and answers $\mathcal{A}$'s oracle queries with its own oracle. $\mathcal{A}$ then returns a ciphertext ct that decapsulates to the same output key k for the challenge public keys pk_0 and pk_1. Note that these public keys must be different with overwhelming probability since KEM is IND-CCA-secure. Thus, c is a ciphertext that decapsulates under two different public keys, and $\mathcal{B}$ wins the *HON-BIND-CT-PK* game. □

Lemma A.6. *Let* KEM *be a KEM that is LEAK-BIND-CT-PK secure. Then* KEM *is also LEAK-BIND-CT-K secure.*

Proof. As the key pairs leaked to the adversary are still honestly generated, the proof is analogous to Lemma A.5. □

Proposition A.7. *There exists a KEM scheme* KEM *that is MAL-BIND-CT-PK but not MAL-BIND-CT-K.*

Proof. (Sketch) Recall that the robustness definition for PKEs from [29] corresponds to our *HON-BIND-CT-PK* property for KEMs. Thus, we conjecture that a FO-KEM with a robust PKE, for instance $\mathsf{KEM}_m^{\perp}$, can meet *MAL-BIND-CT-PK* if there are no *weak* keys in the PKE scheme. However, proving properties of a specific KEM is beyond the scope of this paper, and we leave it as future work.
To prove the statement, we create a variant of this KEM where we alter the decapsulation algorithm: For two key pairs $(sk, pk), (sk', pk)$, which both cannot be created by the normal key generation algorithm, decapsulation returns a fixed $k \notin \mathcal{K}$, which is needed for correctness. Note that this new variant is still *MAL-BIND-CT-PK* since the public key of our added key pairs are the same. However, our variant is not *MAL-BIND-CT-K* since we can use the two added key pairs to get the same key for any ciphertext. This new variant is still IND-CCA-secure and correct since the added key pairs cannot be generated by the normal key generation. □

Theorem A.8. *Let* KEM *=* (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. For all* $X \in \{MAL, LEAK, HON\}$ *we have that if* KEM *is X-BIND-K-PK-secure and X-BIND-K, PK-CT-secure, then* KEM *is also X-BIND-K-CT-secure.*

Proof. We choose $P = P' = \{K\}$, $R = R' = \{CT\}$, $Q = \{K, PK\}$, and $Q' = \{PK\}$. Since $Q \subseteq Q' \cup P'$, KEM is also *X-BIND-K-CT* by Theorem A.4. □

Theorem A.9. *Let* KEM *=* (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism. For all* $X \in \{MAL, LEAK, HON\}$ *we have that if* KEM *is X-BIND-K-CT-secure and X-BIND-K, CT-PK-secure, then* KEM *is also X-BIND-K-PK-secure.*

Proof. We choose $P = P' = \{K\}$, $R = R' = \{PK\}$, $Q = \{K, CT\}$, and $Q' = \{CT\}$. Since $Q \subseteq Q' \cup P'$, KEM is also *X-BIND-K-PK* by Theorem A.4. □

B Constructions for our Separating Examples

B.1 Construction – Kyber

Properties: *LEAK-BIND-K-CT*, *LEAK-BIND-K, PK-CT*, *LEAK-BIND-K-PK*, *LEAK-BIND-CT, K-PK*
Vulnerabilities: *HON-BIND-CT-K*, *HON-BIND-CT-PK*
Description: We use the KEM Kyber from [12] without any modifications.
Proofs Sketches and Counterexamples:

- Since Kyber includes public key and ciphertext in the output key computation, it is both *LEAK-BIND-K-PK* and *LEAK-BIND-K-CT*.
- By Lemma 4.4, Kyber is also *LEAK-BIND-K, PK-CT* and *LEAK-BIND-CT, K-PK*.
- Since Kyber is implicitly rejecting, it cannot satisfy *HON-BIND-CT-K* or *HON-BIND-CT-PK*.

B.2 Construction – $\mathsf{KEM}_m^{\perp}$

$\mathsf{Encaps}_m(pk; r)$	$\mathsf{Decaps}_m^{\not\perp}(sk, ct)$	$\mathsf{Decaps}_m^{\perp}(sk, ct)$
$m \leftarrow_r \mathcal{M}$	Parse $(sk', s) \leftarrow sk$	$m \leftarrow \mathsf{Dec}_1(sk, ct)$
$ct \leftarrow \mathsf{Enc}_1(pk, m)$	$m \leftarrow \mathsf{Dec}_1(sk', ct)$	**if** $m = \perp$: **return** $\perp$
$k \leftarrow \mathsf{H}(m)$	**if** $m \neq \perp$:	**else return**
return (ct, k)	**return** $k \leftarrow \mathsf{H}(m)$	$k \leftarrow \mathsf{H}(m)$
	else	
	return $k \leftarrow \mathsf{H}(s, ct)$	

$\mathsf{Enc}_1(pk, m)$	$\mathsf{Dec}_1(sk, ct)$
$ct \leftarrow \mathsf{Enc}(pk, m; \mathsf{G}(m))$	$m \leftarrow \mathsf{Dec}(sk, ct)$
return ct	**if** $m = \perp \vee \mathsf{Enc}(pk, m; \mathsf{G}(m)) \neq ct$:
	return $\perp$
	else return m

Properties: *LEAK-BIND-CT-PK*,*LEAK-BIND-CT-K*,*LEAK-BIND-K*, *PK-CT*,*LEAK-BIND-K*, *CT-PK*
Vulnerabilities: *HON-BIND-K-CT*,*HON-BIND-K-PK*
Description: We use $\mathsf{KEM}_m^{\perp}$ from [30] without any modifications. Note that in contrast to our definition of Encaps, Encaps is deterministic in this construction and uses an explicit source of entropy r to randomize the encapsulation. $\mathsf{KEM}_m^{\perp}$ uses $\mathsf{Encaps}_m(pk; r)$ and $\mathsf{Decaps}_m^{\perp}(sk, ct)$ as shown above. Note that the underlying PKE algorithm $\mathsf{Enc}_1(pk, m)$ is deterministic, as it uses $\mathsf{G}(m)$ as randomness. For more details, we refer the reader to [30].
Proofs Sketches and Counterexamples:

- We conjecture that $\mathsf{KEM}_m^{\perp}$ is *LEAK-BIND-CT-PK* and *LEAK-BIND-CT-K* if the underlying PKE is robust. We leave a proof as future work.
- From Lemma 4.4 we also get that KEM is *LEAK-BIND-K*, *CT-PK*.
- We prove that $\mathsf{KEM}_m^{\perp}$ is *LEAK-BIND-K*, *PK-CT*. Assume towards contradiction that $\mathsf{KEM}_m^{\perp}$ is not *LEAK-BIND-K*, *PK-CT*. Thus, there exists an adversary $\mathcal{A}$ that wins the *MAL-BIND-PK*, *K-CT*$_{\mathcal{A}}^{\mathsf{KEM}_m^{\perp}}$ game with non-negligible probability. On input $(\mathsf{sk}_0, \mathsf{pk}_0)$, $(\mathsf{sk}_1, \mathsf{pk}_1)$, $\mathcal{A}$ creates $(\mathsf{ct}_0, \mathsf{ct}_1)$ such that $\mathsf{pk}_0 = \mathsf{pk}_1$, $\mathsf{ct}_0 \neq \mathsf{ct}_1$, and $\mathsf{KEM.Decaps}(\mathsf{sk}_0, \mathsf{pk}_0, \mathsf{ct}_0) = \mathsf{k}_0 = \mathsf{k}_1 = \mathsf{KEM.Decaps}(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{ct}_1)$. Note that both decapsulation calls accepted and thus we conclude that Dec_1 must have accepted both ciphertexts. In fact, both ciphertexts must contain the same message m because $\mathsf{H}(m_0) = \mathsf{k}_0 = \mathsf{k}_1 = \mathsf{H}(m_1)$, which implies that $m_0 = m_1$ if H is a collision-resistant hash function or a random oracle. Here, m_0 and m_1 are the messages that ct_0 and ct_1 decrypt to. Because decryption succeeded, we know that the re-encryption check $\mathsf{Enc}(\mathsf{pk}_i, m; \mathsf{G}(m)) = \mathsf{ct}_i$ of Dec_1 succeeded for both ct_0 and ct_1. However, because $\mathsf{pk}_0 = \mathsf{pk}_1$, it follows that $\mathsf{ct}_0 = \mathsf{Enc}(\mathsf{pk}_0, m; \mathsf{G}(m)) = \mathsf{Enc}(\mathsf{pk}_1, m; \mathsf{G}(m)) = \mathsf{ct}_1$, which contradicts our assumption that $\mathsf{ct}_0 \neq \mathsf{ct}_1$. Hence, $\mathsf{KEM}_m^{\perp}$ is *LEAK-BIND-K*, *PK-CT*-secure.
- Observe that $\mathsf{KEM}_m^{\perp}$ derives output key k only from m, which only depends on r. Thus, we can create multiple ciphertexts that decapsulate to the same output key k for honestly generated key pairs by re-using r. For an example, see Figure 3. Therefore, $\mathsf{KEM}_m^{\perp}$ is not *HON-BIND-K-CT*. In fact, it is also not *HON-BIND-K-PK*.

B.3 Construction – $\mathsf{KEM}_m^{\perp}$ Variant

Properties: *MAL-BIND-K-PK*, *MAL-BIND-CT-PK*, *MAL-BIND-CT-K*, *MAL-BIND-K*, *CT-PK*
Vulnerabilities: *HON-BIND-K-CT*, *HON-BIND-K*, *PK-CT*
Description: We propose a variant KEM of the $\mathsf{KEM}_m^{\perp}$ scheme shown in Appendix B.2 where encapsulation shortens the randomness r by one bit through a right-shift and hashes the public key into the final key. To maintain correctness, we change the hashing of the decapsulation algorithm.
Proofs Sketches and Counterexamples:

- Note that KEM is *MAL-BIND-K-PK* because it hashes pk into the output key.
- If the underlying PKE of $\mathsf{KEM}_m^{\perp}$ is robust, we conjecture that KEM is *MAL-BIND-CT-PK*. The proof is a straightforward reduction where an adversary against KEM can also break *MAL-BIND-*

$\mathsf{Encaps}(pk; r)$	$\mathsf{Decaps}(sk, ct)$
$r' \leftarrow r \gg 1$	$k \leftarrow \mathsf{KEM}_m^{\perp}.\mathsf{Decaps}(sk, ct)$
$k, ct \leftarrow \mathsf{KEM}_m^{\perp}.\mathsf{Encaps}(pk; r')$	**if** $m = \perp$: **return** $\perp$
$k' \leftarrow \mathsf{H}(k \| pk)$	**else return**
return (ct, k)	$k \leftarrow \mathsf{H}(k \| pk)$

CT-PK of $\mathsf{KEM}_m^{\perp}$.

- By Lemma 4.2 and Lemma 4.7, KEM is also *LEAK-BIND-CT-K*. We conjecture that it also is *MAL-BIND-CT-K* since we constructed the scheme with any *backdoor* values.
- By Lemma 4.4, KEM is also *MAL-BIND-K, CT-PK*.
- KEM is not *HON-BIND-K-CT*: Encapsulating against the same public key with the two values r_1, r_2, which only differ in the last bit, results in the same ciphertext and the same output key.
- KEM is not *HON-BIND-K, PK-CT* since our attack against *HON-BIND-K-CT* works against a single public key.

B.4 Construction – $\mathsf{KEM}_m^{\not\perp}$

$\mathsf{Encaps}_m(pk; r)$	$\mathsf{Decaps}_m^{\not\perp}(sk, ct)$	$\mathsf{Decaps}_m^{\perp}(sk, ct)$
$m \leftarrow_r \mathcal{M}$	Parse $(sk', s) \leftarrow sk$	$m \leftarrow \mathsf{Dec}_1(sk, ct)$
$ct \leftarrow \mathsf{Enc}_1(pk, m)$	$m \leftarrow \mathsf{Dec}_1(sk', ct)$	**if** $m = \perp$: **return** $\perp$
$k \leftarrow \mathsf{H}(m)$	**if** $m \neq \perp$:	**else return**
return (ct, k)	**return** $k \leftarrow \mathsf{H}(m)$	$k \leftarrow \mathsf{H}(m)$
	else	
	return $k \leftarrow \mathsf{H}(s, ct)$	

$\mathsf{Enc}_1(pk, m)$	$\mathsf{Dec}_1(sk, ct)$
$ct \leftarrow \mathsf{Enc}(pk, m; \mathsf{G}(m))$	$m \leftarrow \mathsf{Dec}(sk, ct)$
return ct	**if** $m = \perp \vee \mathsf{Enc}(pk, m; \mathsf{G}(m)) \neq ct$:
	return $\perp$
	else return m

Properties: *LEAK-BIND-K, PK-CT*, *LEAK-BIND-K, CT-PK*
Vulnerabilities: *HON-BIND-K-CT*, *HON-BIND-K-PK*, *HON-BIND-CT-PK*, *HON-BIND-CT-K*
Description: We use $\mathsf{KEM}_m^{\not\perp}$ from [30] without any modifications. $\mathsf{KEM}_m^{\not\perp}$ uses $\mathsf{Encaps}_m(pk; r)$ and $\mathsf{Decaps}_m^{\not\perp}(sk, ct)$ as shown above. Note that the underlying PKE algorithm $\mathsf{Enc}_1(pk, m)$ is deterministic, as it uses $\mathsf{G}(m)$ as randomness. For more details, we refer the reader to [30].
Proofs Sketches and Counterexamples:

- Since the ouput key is only dervied from message m, $\mathsf{KEM}_m^{\not\perp}$ is not *HON-BIND-K-CT* and not *HON-BIND-K-PK*. For more details see Appendix B.2.
- Because $\mathsf{KEM}_m^{\not\perp}$ is implicitly rejecting, it cannot satisfy *HON-BIND-CT-PK* and *HON-BIND-CT-K* by Theorem 4.11.
- Assume towards contradiction that $\mathsf{KEM}_m^{\not\perp}$ is not *LEAK-BIND-K, CT-PK*. Thus, there exists $\mathcal{A}$ that wins the *LEAK-BIND-K, CT-PK*$_{\mathcal{A}}^{\mathsf{KEM}_m^{\not\perp}}$ game with non negligible probability. $\mathcal{A}$ creates a single ciphertext ct such that $\mathsf{k}_0 = \mathsf{Decaps}(\mathsf{sk}_0, \mathsf{pk}_0, \mathsf{ct}) = \mathsf{Decaps}(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{ct}) = \mathsf{k}_1$ for different public keys. We now argue why $\mathsf{k}_0 = \mathsf{k}_1$, for the same ciphertext and different public keys, implies that both decapsulation calls must have accepted. From Lemma 4.10 we know that the rejection key computation must contain the ciphertext and a secret random value that is different for each key pair. Hence, $\mathsf{k}_0 = \mathsf{k}_1$ can only occur with negligible probability in the rejection case since it constitutes a hash/random oracle collision. Thus, both decapsulation calls in the game must have accepted with overwhelming probability. However, in this case, $\mathsf{KEM}_m^{\not\perp}$ behaves exactly like $\mathsf{KEM}_m^{\perp}$ (Appendix B.2). Therefore, we can use $\mathcal{A}$ to win the *LEAK-BIND-K, CT-PK*$_{\mathcal{A}}^{\mathsf{KEM}_m^{\perp}}$ game with

	$\neg$(*HON-BIND-K-CT*)	$\neg$(*HON-BIND-K-PK*)	$\neg$(*HON-BIND-CT-PK*)	$\neg$(*HON-BIND-CT-K*)	$\neg$ (*HON-BIND-K, CT-PK*)	$\neg$ (*HON-BIND-K, PK-CT*)
LEAK-BIND-K-CT	X	Cons. B.5	Cons. B.5	Cons. B.1	Cons. B.5	X Lemma 4.4
LEAK-BIND-K-PK	Cons. B.3	X	Cons. B.1	Cons. B.1	X Lemma 4.4	Cons. B.3
LEAK-BIND-CT-PK	Cons. B.2	Cons. B.2	X	X Lemma 4.6	X Lemma 4.4	Cons. B.3
LEAK-BIND-CT-K	Cons. B.2	Cons. B.2	Cons. B.6	X	Cons. B.6	Cons. B.3
LEAK-BIND-K, CT-PK	Cons. B.2	Cons. B.2	Cons. B.4	Cons. B.4	X	Cons. B.3
LEAK-BIND-K, PK-CT	Cons. B.2	Cons. B.2	Cons. B.4	Cons. B.4	Cons. B.5	X

Table 4: Summary of separating examples between our binding properties. An X means that a separating example does not exists.

non-negligible probability, which contradicts $\mathsf{KEM}_m^{\perp}$'s *LEAK-BIND-K, CT-PK* security.

- The prove for *LEAK-BIND-K, PK-CT* security is analogous to the prove we give in Appendix B.2 up to the argument why both decapsulation calls must have accepted. We now argue why this is also the case for $\mathsf{KEM}_m^{\not\perp}$. Since $\mathsf{k}_0 = \mathsf{k}_1$ and the rejection key computation contains the ciphertexts $\mathsf{ct}_0 \neq \mathsf{ct}_1$, it is clear that the two decapsulations cannot derive the same key when one of them rejects. Thus, both must have accepted.

B.5 Construction – Classic McEliece

Properties: *LEAK-BIND-K-CT*, *LEAK-BIND-K, PK-CT*
Vulnerabilities: *HON-BIND-K-PK*, *HON-BIND-CT-PK*, *HON-BIND-K, CT-PK*
Description: We use the Classic McEliece scheme from [7] without any modifications. Classic McEliece derives the output key as follows: $\mathsf{k} \leftarrow H(m||c||H'(m))$. Here, m is a message from the message space of the underlying PKE, c is the ciphertext created by encapsulating m with a public key, and H, H' are hash functions.
Proofs Sketches and Counterexamples:

- Since Classic McEliece hashes the ciphertext into the the output key it is *LEAK-BIND-K-CT*.
- By Lemma 4.4 we also get that Classic McEliece is *LEAK-BIND-K, PK-CT*.
- From [29], we know that it is possible to construct a single ciphertext c that decrypts to the same message under any Classic McEliece private key. Therefore, Classic McEliece is not *HON-BIND-K-PK* because we can use c to derive the same output key k under any key pair.
- We create a ciphertext c that decrypts to the same m for any key pair, as described in [29]. Since Classic McEliece derives the output key only from the ciphertext c and the message m, c decapsulates to the same output key for every secret key. Thus, Classic McEliece is not *HON-BIND-K, CT-PK*.
- Since Classic McEliece is an implicitly rejecting KEM, it is not *HON-BIND-CT-PK*.

B.6 Construction – Custom Scheme

Properties: *LEAK-BIND-CT-K*
Vulnerabilities: *HON-BIND-CT-PK*, *HON-BIND-K, CT-PK*
Description: Let $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a *LEAK-BIND-CT-K* KEM. We now construct a new KEM KEM' from KEM. Let $\mathsf{KeyGen}' =$

$$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}; \mathsf{pk}' \leftarrow \mathsf{pk} \cdot 2 + b \leftarrow\!\$\ \{0,1\}; \text{return } (\mathsf{sk}, \mathsf{pk}').$$

$$\mathsf{Decaps}(\mathsf{sk}, \mathsf{pk}, \mathsf{ct})' = \mathsf{Decaps}(\mathsf{sk}, \mathsf{pk} \text{ div } 2, \mathsf{ct}).$$

$$\mathsf{Encaps}(\mathsf{pk})' = \mathsf{Encaps}(\mathsf{pk} \text{ div } 2).$$

Additionally, we assume that the adversary can use a version of Encaps where they control the entropy of the probabilistic algorithm and are able to choose the same entropy for multiple distinct encapsulations. KEM' is still *LEAK-BIND-CT-K*, as we can use an adversary $\mathcal{A}'$ against KEM' to build an adversary $\mathcal{A}$ against KEM.
Proofs Sketches and Counterexamples: KEM' is not *HON-BIND-CT-PK* and *HON-BIND-K, CT-PK* as we have two public keys for each ciphertext that yield the same output key if the entropy is chosen by the adversary.

C Tamarin Implementation

To model the key generation of the KEM, the event $GoodKey(\mathsf{pk})$ is introduced, indicating that pk was generated honestly. Properties such as *Correctness* are later specified to apply only to keys that were produced via honest key generation.
Users of the library can employ arbitrary values as secret keys, but they must use the function symbol $kem\_pk(\cdots)$ to generate the corresponding public key.
Encapsulation and decapsulation operations of KEMs are modeled with events $\mathsf{Encaps}(\mathsf{k}, \mathsf{ct}, \mathsf{pk})$ and $\mathsf{Decaps}(\mathsf{pk}, \mathsf{ct}, \mathsf{pk}, \mathsf{sk})$. The input is provided via the premises of the protocol rule annotated with the event, while the output is delivered through persistent facts $\mathsf{!KeyValues}(\mathsf{k})$ and $\mathsf{!CTValues}(\mathsf{ct})$ in the rule's premises. These persistent facts come from our symbolic model of KEMs and represent the key- and ciphertext space, respectively. For instance, when modeling an encapsulation

$$[\mathsf{Alice}(\mathsf{pk}, \mathsf{sk}), \mathsf{!KeyValues}(\mathsf{k}), \mathsf{!CTValues}(\mathsf{ct})] \quad (1)$$
$$-[\mathsf{Encaps}(\mathsf{k}, \mathsf{ct}, \mathsf{pk})]\rightarrow \quad (2)$$
$$[\mathsf{Out}(\mathsf{ct})] \quad (3)$$

$\mathsf{!KeyValues}(\mathsf{k})$ and $\mathsf{!CTValues}(\mathsf{ct})$ are needed to bind the output of the Encaps call on pk.
By default, no restriction is placed on these values, and arbitrary collisions in the key and ciphertext are possible.
To restrict collisions in the key and ciphertext spaces, the computational properties such as *Correctness*, *HON-BIND-K-CT*, and *MAL-BIND-K-CT* are encoded as logical formulas to ensure desired behavior, including key equality and binding properties. Tamarin traces violating these properties are discarded through imposed *restrictions*. For instance, we can encode the *Correctness* property using the following formula:

Definition C.1. *Correctness*

$$\forall\ k1\ k2\ ct\ sk\ \#i\ \#j\ \#k\ .$$
$$\mathsf{Encaps}(k1, ct, kem\_pk(sk))@\#i \wedge\ \mathsf{Decaps}(k2, ct, kem\_pk(sk), sk)@\#j$$
$$\wedge\ \mathsf{GoodKey}(kem\_pk(sk))@\#k$$
$$\Rightarrow\ (k1 = k2)$$

The restriction encodes that an Encaps call and a Decaps call that use the same public key pair–produced by the honest key generation algorithm–and the same ciphertext must also produce the same key.
In the same manner, we can encode our binding properties. For instance, *HON-BIND-K-CT*:

Definition C.2. *HON-BIND-K-CT*

$$\forall\ k\ ct1\ ct2\ pk1\ pk2\ sk1\ sk2\ \#i\ \#j\ \#l\ \#m\ .$$
$$\mathsf{Decaps}(k, ct1, pk1, sk1)@\#i \wedge\ \mathsf{Decaps}(k, ct2, pk2, sk2)@\#j$$
$$\wedge\ \mathsf{GoodKey}(pk1)@\#l \wedge \mathsf{GoodKey}(pk2)@\#m$$
$$\Rightarrow\ (ct1 = ct2)$$

To encode *MAL-BIND-K-CT*, we drop the requirement that the public keys have to be good keys, and we add additional restrictions that enforce the same behavior for pairs of Encaps calls as well as pairs of Encaps and Decaps.
For IND-CCA, it is required that the output key k of an Encaps call with a good key is distinct from any other output key produced by Encaps. We model this with a restriction as well.
Additionally, in a classical symbolic model with function symbols, the adversary can freely use these symbols as well. To also allow this behavior for our event-based computations, we provide protocol rules in our model that allow the adversary to perform these computations. While this would encode the adversary to emulate honest parties, we also give the adversary access to a modified Encaps computation where they can force the output key to be any value they already know. This models the case where the adversary does not use *fresh* randomness but reuses randomness to compute an output key of their choice. Note that this behavior is also why re-encapsulation attacks exist
Optional restrictions are provided to enforce the use of *good keys* in *Encaps* and *Decaps* computations, allowing users to exclude traces where bad keys are used.
Finally, we want to explain why we only implement our properties for $X = HON$ and $X = MAL$ in the symbolic setting. In Section 4.3, we note that *LEAK* and *HON* are indeed different in the computational

model since an adversary can learn intermediate values when computing Decaps themself, as is the case in the *LEAK* setting. This is not the case in the *HON* setting, where they can only observe the result of *honest* Decaps computations done by an oracle. Thus, *LEAK* and *HON* are not equivalent in the computational model. In our KEM model, on the other hand, the output key is not computed from intermediate values but from an atomic fresh value. Thus, an adversary cannot learn any additional information by computing Decaps themself. As a result, *HON* and *LEAK* are equivalent for our KEM model. We leave building a symbolic KEM model that allows for arbitrary combinations of our binding properties, where *LEAK* is not equal to *HON*, as future work. This is challenging because modeling arbitrary partial information leakage in the symbolic model is still an open problem, and modeling the output key as a compound term built from other terms (which could be leaked) inherently satisfies some of our binding properties (see Section 5.1).

D Sigma SK Security Attack

In this section, we explain the attack on SK security in the Σ_0' protocol when the Responder accepts duplicate sessions identifiers and accepts any signature σ_I as long as the session identifier and ciphertext match.

To understand this attack, we want to highlight that the signatures σ_I and σ_R only include c; they *do not* include the actual output key k. Thus, the adversary can choose c such that it decapsulates to different keys for the Initiator and the Responder. This behavior is not excluded by correctness of the KEM, since SK security does not enforce that Initiator and Responder agree on the ephemeral key pk_I; they only need to agree on their respective identities. Thus, the adversary can force a session between A and B where they agree on their identities but disagree on the ephemeral key pk_B. Together with the deficiencies of Σ_0', this allows for an attack on SK security, which can be prevented by using, for instance, a *MAL-BIND-CT-PK*-secure KEM.

The adversary starts a session between Initiator A and Responder B where they agree on their identities and pk_A. After receiving the first message, B computes σ_B and tag_B, which the adversary intercepts. Then, impersonating A, the adversary starts another session with B, where they switch out pk_A for another KEM public key but reuse the session identifier *sid*. Recall that the Responder only accepts this session in the Σ_0' protocol where we do not assume infinite, persistent storage at the Responder side. The Responder B again acts according to the protocol and computes $(c, k) \leftarrow \mathsf{Encaps}(pk_A)$. The adversary then forwards σ_B and tag_B to A, who computes σ_A, tag_A, and finishes their session with B decapsulating k' from c because of the lack of binding properties of KEM. The adversary then forwards these values to B in the session where they disagree on pk_A. Since B accepts any σ_B as long as c and *sid* match, they also finish their session with A computing k. As these sessions agree on the identities but disagree on the final shared key, SK security does not hold.

When honestly generated keys are used, the KEM must at least satisfy *HON-BIND-CT-K*, and, in the presence of maliciously generated keys, the KEM must at least satisfy *MAL-BIND-CT-PK* or *MAL-BIND-CT-K* to prevent this attack. These properties prevent Initiator and Responder from opening the same ciphertext to different output keys.

E Properties of KEM Implementations

In this section, we give an overview of the binding properties of some prominent KEM schemes, i.e., DH-KEM [34, 43], ML-KEM [40], Frodo-KEM [11], Classic McEliece [7], and Kyber [12]. We want to stress that we only *conjecture* these binding properties–we give no proof.

For DH-KEM, we conjecture that it is both *LEAK-BIND-K-CT* and *LEAK-BIND-K-PK* as it includes both values in the derivation of the output key. Therefore, it is also *LEAK-BIND-K, CT-PK* and *LEAK-BIND-K, PK-CT*. While [43] prescribes that a DH-KEM implementation must reject malformed input, this does not make DH-KEM an explicitly rejecting KEM. This is because this validation is not part of the KEM algorithm but should be done by the caller. Inspecting DH-KEM, we notice that it cannot reject at all since exponentiation is well-defined for any valid public key and valid secret key. Therefore, DH-KEM is implicitly rejecting since its Decaps algorithm never returns $\perp$.

For ML-KEM, [5] proves that it is *LEAK-BIND-K, PK-CT*. As it hashes the receiver's public key into the output key, we conjecture that it is *LEAK-BIND-K-PK*. In Theorem A.8, we establish the combination of the former properties implies *LEAK-BIND-K-CT* security. Again, *LEAK-BIND-K, CT-PK* is implied, and the ciphertext cannot bind to anything.

For the properties of Kyber, we refer to Appendix B. With the same reasoning, we conjecture the same properties for Frodo-KEM.
For Classic McEliece, we refer to Appendix B, Cons. B.5.
The following theorems establish that, in the random oracle model, adding the receiver's public key or the ciphertext to the output key derivation makes a KEM *LEAK-BIND-K-PK* or *LEAK-BIND-K-CT* respectively.

Theorem E.1. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism that derives its output key k via* H *which is modeled as a random oracle. If* KEM *uses* H *to compute the output key, and includes ct (or an injective function of ct) at a fixed position in the input bitstring to* H*, then* KEM *is LEAK-BIND-K-CT in the random oracle model.*

Proof. (Sketch) First, we note that H is an injective function since we assume it to be a random oracle. Next, we note that for an adversary to win the $\mathit{LEAK\text{-}BIND\text{-}K\text{-}CT}^{\mathsf{KEM}}$ game, they have to produce k_1, k_2 and ct_1, ct_2 such that $k_1 = k_2 \wedge ct_1 \neq ct_2$. Substituting the output keys with their computation according to KEM, we get $\mathsf{H}(prefix||ct_1||postfix) = \mathsf{H}(prefix'||ct_2||postfix') \wedge ct_1 \neq ct_2$, where $prefix$, $prefix'$, $postfix$, and $postfix'$ are arbitrary bitstrings, as well as $|prefix| = |prefix'|$ and $|postfix| = |postfix'|$. This directly contradicts the injectivity of H. If ct is not directly included in H's input but indirectly through other injective functions (e.g., other random oracles), we use the fact that the composition of injective functions is again an injective function to derive the same contradiction. □

Theorem E.2. *Let* KEM = (KeyGen,Encaps,Decaps) *be a key encapsulation mechanism that derives its output key k via* H *which is modeled as a random oracle. If* KEM *uses* H *to compute the output key, and includes pk (or an injective function of ct) at a fixed position in the input bitstring to* H*, then* KEM *is LEAK-BIND-K-PK in the random oracle model.*

Proof. (Sketch) The proof is analagous to E.1 when substituting ct for pk. □

KEM	*X-BIND-K-CT*	*X-BIND-K-PK*	*X-BIND-K, CT-PK*	*X-BIND-K, PK-CT*
DH-KEM [34, 43]	✓*LEAK* (Theorem E.1)	✓*LEAK* (Theorem E.2)	✓*LEAK* (Theorem 4.5)	✓*LEAK* (Theorem 4.5)
Kyber [12]	✓*LEAK* (Theorem E.1)	✓*LEAK* (Theorem E.2)	✓*LEAK* (Theorem 4.5)	✓*LEAK* (Theorem 4.5)
Frodo-KEM [11]	✓*LEAK* (Theorem E.1)	✓*LEAK* (Theorem E.2)	✓*LEAK* (Theorem 4.5)	✓*LEAK* (Theorem 4.5)
ML-KEM [40]	✓*LEAK* (Theorem E.1)	✓*LEAK* (Theorem A.9)	✓*LEAK* (Theorem 4.5)	✓*LEAK* [5]
Classic McEliece [7]	✓*LEAK* (Theorem E.1)	× [29]	✓*LEAK* (Theorem 4.5)	× [29]

✓= Holds (Proof) ×= Does not hold

Table 5: **Binding properties of popular KEM constructions.** All KEM constructions in this table are implicitly rejecting KEMs, and they therefore do not provide *X-BIND-CT-K* or *X-BIND-CT-PK* by Theorem 4.11. DH-KEM, Kyber, and Frodo-KEM use both ct and pk to derive the the final key, and thus these three KEMs satisfy the remaining four properties. ML-KEM does not use the ciphertext in the final key derivation, but was shown *LEAK-BIND-K, PK-CT* in [5]. By Theorem A.9, ML-KEM is therefore also *LEAK-BIND-K-PK*. Grubbs, Maram, and Paterson showed in [29] that Classic McEliece can achieve neither *X-BIND-K-PK* nor *X-BIND-K, CT-PK*.

F Changelog

- Version 0.1, December 20, 2023: Initial draft.
- Version 0.1.1, December 22, 2023:
  - Fixed typos.
  - Changed symbols for malicious and honest from M, and H to MAL, and HON.
  - Renamed Alex, Blake, and Charlie to A, B, and C.
  - Added Changelog.
- Version 1.0, January 30, 2024:

  - Substantial reworking of the entire paper: updated notation and methodology throughout, and added new content on symbolic analysis.
  - Moved proofs and constructions to the appendices.
  - Updated the generic binding game for the malicious (MAL) setting based on [26].
  - Added a new $LEAK$-variant to the generic binding games.
  - Expanded the comparison to existing binding notions.
  - Introduced a symbolic approach for analyzing security protocols with respect to our binding properties in the framework of the Tamarin prover.
  - Analyzed several case studies using our symbolic approach.
- Version 1.0.1, January 31, 2024: Update Biblio
- Version 1.0.2, February 08, 2024: Update Biblio. Fix citation to wrong paper. Fix typo.
- Version 1.0.3, February 15, 2024: Fix typos and punctuation.
- Version 1.0.4, February 23, 2024: Update relations between our properties
- Version 1.0.5, March 5, 2024: Global rename of $X\text{-}BIND\text{-}PK, K\text{-}CT$ to $X\text{-}BIND\text{-}K, PK\text{-}CT$. Added Appendix E on binding properties of popular KEM implementations.
- Version 1.0.6, April 3, 2024: Fix inconsistencies/bugs in generic binding games (Figure 5)
- Version 1.1.0, May 29, 2024: Add related work section addressing [49]. Expand subsection on implicitly rejecting KEMs. Add explanations regarding our API choices.