Towards KEM Unification

Daniel J. Bernstein[1] and Edoardo Persichetti[2]

[1] Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607–7045, USA
djb@cr.yp.to

[2] Department of Mathematical Sciences
Florida Atlantic University
Boca Raton, FL 33431, USA
epersichetti@fau.edu

Abstract. This paper highlights a particular construction of a correct KEM without failures and without ciphertext expansion from any correct deterministic PKE, and presents a simple tight proof of ROM IND-CCA2 security for the KEM assuming merely OW-CPA security for the PKE. Compared to previous proofs, this proof is simpler, and is also factored into smaller pieces that can be audited independently. In particular, this paper introduces the notion of "IND-Hash" security and shows that this allows a new separation between checking encryptions and randomizing decapsulations. The KEM is easy to implement in constant time, given a constant-time implementation of the PKE.

Keywords: PKE, OW-CPA, OW-Passive, IND-Hash, rigid, KEM, IND-CCA2, ROM

1 Introduction

Out of the 45 encryption/key-exchange submissions to NIST's post-quantum competition (not counting submissions that have been withdrawn at the time of this writing), at least nine—BIG QUAKE [6], Classic McEliece [9], DAGS [5], NTRU-HRSS-KEM [22], NTRU Prime [10], NTS-KEM [3], Odd Manhattan [34], pqRSA [12], and RQC [2]—specify correct KEMs, i.e., KEMs where valid

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was supported by the European Commission under Contract ICT-645622 PQCRYPTO; by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005; and by the U.S. National Science Foundation under grant 1314919. "Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation" (or other funding agencies). Permanent ID of this document: 7502421bbf40a3bf0ad27d5b50c6bdbd75d970f7. Date: 2018.05.28.

ciphertexts are always correctly decrypted.[1] Thanks to Gustavo Banegas for help in collecting this data.

Each of these correct KEMs can be viewed as having two parts:

- A correct PKE is applied to a random input to obtain a ciphertext.
- The KEM does more work aiming for a strong security goal, IND-CCA2. The underlying PKE has a relatively weak security goal.

This might sound straightforward. There is a long tradition of building a correct KEM with strong security goals starting from a correct PKE with weak security goals. The classic example is Shoup's "Simple RSA", also known as "RSA-KEM" [40, Section 20], which sends a ciphertext $r^{65537} \bmod N$ encapsulating the session key $H(r)$. Here N is the public key; r is a uniform random integer modulo N; and the function $r \mapsto r^{65537} \bmod N$ is the underlying PKE. The only extra work in Simple RSA, beyond the PKE, is computing a hash $H(r)$ as a session key.

However, post-quantum PKEs almost always have complications such as invalid ciphertexts. These complications require care in the PKE-to-KEM conversions. See [11, Appendix K] and [13] for recent examples of chosen-ciphertext attacks against post-quantum KEMs[2] that simply hash a random PKE input, as in Simple RSA. The NIST submissions mentioned above use a variety of different PKE-to-KEM conversions that are intended to stop such attacks.

Our basic goal in this paper is to simplify the problem of auditing these PKE-to-KEM conversions, and in particular the proofs that are claimed for these conversions. To avoid gaps between what cryptanalysts are studying and what theorems assume, we focus on "tight" proofs starting from minimal PKE security goals. We want these proofs to be as simple as possible. We want the proofs to be factored as compositions of simpler theorems that can be audited separately. We want the proofs to be spelled out in full detail to support manual verification and, hopefully, formal verification.

Trying to cover all the different conversions that have been used before is *not* one of our goals. On the contrary: because we want auditing to be as simple as possible, we recommend having the community come to consensus on a single PKE-to-KEM conversion, and "tweaking" submissions as necessary to use that conversion.

Tweaking submissions to unify the KEM constructions is allowed by the NIST process and will allow a simpler competition among the underlying PKEs. Before the deadline for round-1 submissions to NIST, one of us (Persichetti) wrote

> My idea is that submitters should not worry too much about which conversion is used at this stage (unless it is weak/breakable, of course) and, exactly as you say, what matters the most is the inherent strength of the underlying one-way function.

[1] A tenth submission, LIMA, originally claimed to be correct ("remove the issue of decryption failures" [42, page 49, under "Advantages"]), but later credited Leixiao Cheng and Yunlei Zhao for discovering an apparently unfixable error in the analysis ("This does not work" [41, page 8]).

[2] One of the two attacked KEMs is a correct KEM; the other is only partially correct.

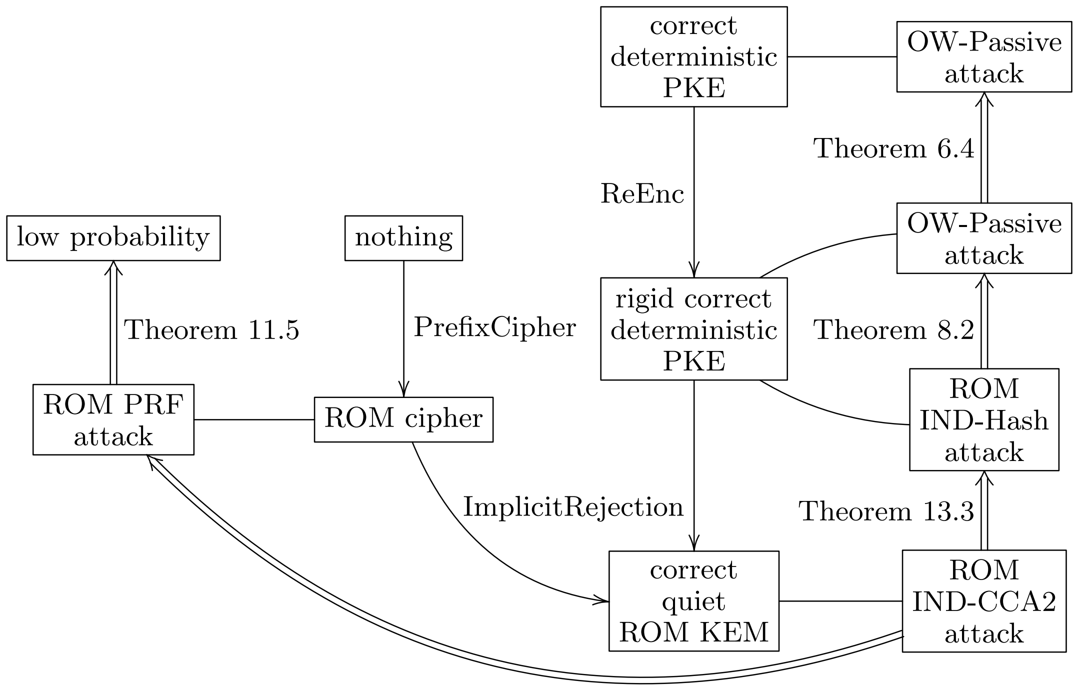

Fig. 1.1. Modules in the construction and security analysis of SimpleKEM. OW-Passive is the same as OW-CPA. All theorems are tight, with a total probability loss of $q/\#\mathsf{Plaintexts} + 2q/\#\mathsf{CipherKeys}$; see Theorem 14.3.

NIST's Dustin Moody responded "That certainly seems reasonable to us. These type of issues could be addressed later on in the process when submissions are allowed minor changes (tweaks)." See [33] and [29].

Here is an example of how we are are prioritizing simplicity for auditors: we categorically recommend "implicit rejection", where *invalid* ciphertexts produce pseudorandom results rather than failures. Submissions using explicit rejection can simply switch to implicit rejection, and then auditors no longer have to worry about explicit rejection. Our reasons for recommending implicit rejection rather than explicit rejection are explained in Section 16.

More specifically, we highlight the following KEM conversion, which we call SimpleKEM, producing a correct quiet KEM from a correct deterministic PKE:

- Encapsulation generates a uniform random plaintext p, the corresponding ciphertext $C = \mathsf{Encrypt}(p, K)$, and a session key $H(1, p, C)$. Here K is the public key for the PKE.
- Decapsulation computes $p = \mathsf{Decrypt}(C, k)$ and verifies $C = \mathsf{Encrypt}(p, K)$. If both steps succeed then decapsulation outputs the session key $H(1, p, C)$; otherwise it outputs $H(0, r, C)$. Here k is the private key for the PKE, and r is an independent secret stored as part of the KEM private key.

Note that decapsulation always outputs a session key, never $\perp$; this is what "quiet" means.

Our proof works when $H(1, p, C)$ is replaced by $H(1, p)$ or, more generally, $H(1, p, \text{any function of } C)$. Also, $H(0, r, C)$ can be replaced by the output of any cipher with secret key r and input C. We recommend $H(1, p, C)$ and $H(0, r, C)$ because this combination simplifies the task of producing constant-time implementations; see Section 15.

A special case of the $H(1, p)$ variant of SimpleKEM, for a particular code-based PKE, was introduced by Persichetti in [32, Section 5.3], modulo minor details. In particular, implicit rejection was introduced in [32, Table 5.4, "pseudorandom function" used in "Dec"]. The $H(1, p)$ and $H(1, p, C)$ variants were stated for any PKE by Hofheinz, Hövelmanns, and Kiltz in [20] under the names "$U^{\not\perp}_m$" and "$U^{\not\perp}$" respectively, again modulo minor details.

We give a counterexample (see Appendix A) to the security claim in [20] for "$U^{\not\perp}_m$". Our proof avoids this problem because we factor it via a different security property, which we call "rigidity". Our proof is simpler than [20], more general than [32], and factored into smaller pieces, as summarized in Figure 1.1. See Section 2 for a more detailed discussion of the history and of what is new here.

2 Background and contributions

"PKE" is an abbreviation for "public-key encryption scheme". The simplest security property of PKEs—and the security property most closely studied by cryptanalysts—is "one-wayness against chosen-plaintext attacks" ("OW-CPA"). This means that an attacker cannot recover a uniform random plaintext, given a random public key and the corresponding ciphertext.

The "chosen-plaintext attacks" terminology is misleading: it suggests, incorrectly, that the attacker is permitted to choose plaintexts. We suggest renaming "OW-CPA" as **OW-Passive**.

Compared to a PKE with OW-Passive security, the traditional view is that users need PKEs with

- additional functionality, namely handling variable-length strings as inputs (whereas the simplest PKEs have a limited set of plaintexts); and
- much more security, namely IND-CCA2 security—which also requires PKEs to be randomized (whereas it is simpler for PKEs to be deterministic).

Fujisaki and Okamoto [17] proposed a generic construction starting from any correct PKE secure against OW-Passive attacks, along with secure symmetric tools, and obtaining a correct variable-input-length PKE secure against a class of IND-CCA2 attacks, namely ROM IND-CCA2 attacks. This result was not the end of the story, for several reasons:

- The proof was not "tight": it allowed ROM IND-CCA2 attacks to be faster than OW-Passive attacks by a considerable polynomial factor, essentially the number of hash queries carried out by the attacker.
- There could be IND-CCA2 attacks faster than any ROM IND-CCA2 attacks. This is not a hypothetical concern in a post-quantum world: even the best,

most thoroughly reviewed b-bit hash function allows high-probability preimage attacks in essentially $2^{b/2}$ operations by Grover's algorithm, whereas ROM preimage attacks require essentially 2^b operations.
- It is not clear how thoroughly the proof has been verified. For comparison, Shoup in [39] announced a serious bug in a heavily cited security claim published seven years earlier regarding another construction of public-key encryption. Shoup also gave strong reasons to believe that the claim could not be proven without additional assumptions.
- Further complicating the picture is that there is some new attention to PKEs that are only *partially* correct—e.g., 99.999999999999999995% correct. See [20].

Sections 2.1 and 2.2 review steps forward in verifiability and tightness by Shoup, Dent, and Hofheinz–Hövelmanns–Kiltz. Section 2.3 explains our improvements in verifiability for tight conversions from PKEs secure against OW-Passive attacks to KEMs secure against ROM IND-CCA2 attacks. Our work also appears to be relevant to QROM IND-CCA2 attacks (see Section 2.4), although this is not our main focus.

As for correctness, we follow the traditional simplification of focusing on correct PKEs. Correctness simplifies auditing, and correct PKEs are already used in quite a few NIST submissions, so this is an important case to consider. Furthermore, most—although not all—other NIST submissions can easily change parameters to provide correctness at a tolerable cost in performance. The study of correct PKEs is also a natural starting point for generalizations to cover PKEs that are only partially correct, if those PKEs are of long-term interest.

2.1. Modularity. Shoup in [38, Section 4.2] and [40, Section 3] proposed building a variable-input-length PKE in two steps: first, build a KEM—a "key-encapsulation mechanism"; second, combine the KEM with symmetric tools to build a variable-input-length PKE. This approach has a small disadvantage, a small advantage, and a much more important advantage:

- The small disadvantage is that KEM ciphertexts do not communicate any user data. More complicated direct constructions of a PKE sometimes have shorter ciphertext lengths, packing some of the user data into the asymmetric part of the ciphertext.
- The small advantage is that the session key communicated by a KEM can be used directly to encrypt and authenticate any number of messages back and forth in the session. This requires the symmetric tools to be authenticated ciphers rather than merely "DEMs", but there are many thoroughly vetted choices of authenticated ciphers.
- The much more important advantage is that constructing a KEM is simpler than constructing a variable-input-length PKE. Shoup thus factored the task of building the final PKE into three simpler subtasks that can be audited independently: constructing a KEM, constructing an authenticated cipher, and combining a KEM with an authenticated cipher.

Our goal for the rest of this paper is to build a correct KEM with IND-CCA2 security, starting from a correct PKE with OW-Passive security; we do not say anything else about *using* the KEM.

2.2. Tightness. Dent in [15] presented proofs that various PKE-to-KEM conversions produce ROM-IND-CCA2-secure KEMs, under various assumptions regarding the underlying PKE. Some of these conversions are tight.

One of Dent's tight conversions, [15, Theorem 8], is particularly attractive because its security assumption for the PKE is merely OW-Passive security. This conversion requires the PKE to be deterministic. This means that encryption is deterministic; equivalently, decryption recovers all of the randomness used in producing a ciphertext. Not all PKEs in the literature are deterministic, but there are enough[3] deterministic PKEs to justify attention to this case, and we focus on this case in this paper.

As in Simple RSA, Dent uses a hash of the PKE plaintext as a session key. Unlike Simple RSA, Dent sends another hash of the PKE plaintext along with the PKE ciphertext. Following [11, Section 2.3] and [9, Section 4.3], we refer to this additional hash as "plaintext confirmation", by analogy to the "confirmation" hashes communicated in various key-exchange protocols. Plaintext confirmation makes clear that whoever generated a ciphertext knows the PKE plaintext; Dent's proof follows this intuition.

Hofheinz, Hövelmanns, and Kiltz in [20] presented several further tight conversions. Three of these start from OW-Passive security of the PKE:

- "S^ℓ" produces much larger ciphertexts, which we do not discuss further.
- "$U^{\not\perp}$", essentially SimpleKEM, has a tight theorem stated and proven in [20, Theorem 3.4]. This theorem starts from "OW-PCA" security, but it is easy to obtain an OW-PCA PKE from a *deterministic* OW-Passive PKE.
- "$U_m^{\not\perp}$", essentially the $H(1, p)$ variant of SimpleKEM, has a tight theorem claimed in [20, Theorem 3.6], again assuming that the PKE is deterministic. No proof is given.

We give a counterexample to [20, Theorem 3.6]; see Appendix A. We also give a counterexample to [20, Theorem 3.5], where a proof was given. This illustrates the importance of having proofs presented in detail and checked carefully. We identify an extra KEM-construction step that ensures security, that is used in SimpleKEM, and that does not appear in [20, Theorems 3.5 and 3.6].

To summarize, at this point in the story there are two different strategies to tightly obtain KEMs with ROM IND-CCA2 security starting from deterministic PKEs with OW-Passive security. One strategy is plaintext confirmation, as in [15]. The other strategy is implicit rejection, as in [32] and [20].

[3] Seven of the nine NIST submissions above—BIG QUAKE, Classic McEliece, DAGS, NTRU-HRSS-KEM, NTRU Prime, NTS-KEM, and pqRSA—naturally include deterministic PKEs. Two of these, BIG QUAKE and DAGS, artificially introduce extra randomness to be able to use a particular strategy for tight IND-CCA2 proofs, but this strategy requires the PKE to target IND-CPA security rather than merely OW-Passive security.

2.3. Contributions of this paper. Given any correct deterministic PKE with OW-Passive security, we prove tight ROM IND-CCA2 security for a correct quiet KEM with the same ciphertext length. This was already done in [20], but our proof has the following advantages:

- We factor the proof of IND-CCA2 security via a security notion that we call "IND-Hash". This produces a natural separation between (1) checking encryptions of oracle queries, to relate IND-Hash security of the PKE to OW-Passive security of the PKE; and (2) handling decapsulation queries, to relate IND-CCA2 security of the KEM to IND-Hash security of the PKE.
- As in [32] and [36], we handle decapsulation queries in a simple way, starting from a uniform random decapsulation function and then easily building a uniform random hash function to match.
- We also factor out two further pieces: (3) "ReEnc", which checks ciphertext validity during decapsulation, and (4) "Derandomize", which converts a randomized PKE into a deterministic PKE, and which we do not need since we start from a deterministic PKE. For comparison, in [20], both of these tasks are handled by one "T" conversion; "T" is factored out of the other proofs, but ReEnc is not factored out of "T". This is important because the theorems stated for "T" are not tight.
- We spell out proofs in full detail, with the goal of supporting verification. For comparison, the previous literature often claims theorems with proofs that are omitted or merely sketched, causing problems for auditors.
- As a smaller point, our SimpleKEM deviates slightly from "$U^{\not\perp}$" in its hash inputs: we use separate hash input spaces for valid ciphertexts and invalid ciphertexts. This simplifies the proofs.
- We also allow $\mathsf{CipherKeys}$, the set of (equally likely) possibilities for r, to be larger than $\mathsf{Plaintexts}$, the set of possibilities for p. For example, in Classic McEliece [9], $\mathsf{CipherKeys}$ is the set of bit vectors of a particular length, and $\mathsf{Plaintexts}$ is the set of *low-weight* bit vectors of a particular length.

Compared to the special case handled earlier in [32], our proofs have the same simplicity, more generality, and more modularity.

We believe that the ROM situation has now reached a satisfactory conceptual state for the important case of correct deterministic PKEs. We recommend our ROM proofs as targets for formal verification.

2.4. Quantum reductions. The central proof technique used by Fujisaki and Okamoto is to inspect each hash query from an IND-CCA2 attacker, checking whether the query encrypts to the challenge ciphertext—in which case the query breaks OW-Passive security. The same technique is repeated in the other proofs mentioned above. However, a *quantum* algorithm attacking IND-CCA2 can call the hash function on a superposition of queries—perhaps a superposition of all possible plaintexts—and then it is not clear what to inspect.

Targhi and Unruh in [43] showed that plaintext confirmation allows a QROM IND-CCA2 security proof. There are two caveats here:

- The plaintext-confirmation hash is required to map to a space as large as the plaintext space, as emphasized in [21, Section 5]. This produces a 141-byte expansion in the ciphertexts in [21].
- Even with a hash of this size, the theorem is far from tight.

The lack of tightness was quantified in [20]. One source of looseness, as in the non-quantum case, is handling randomized PKEs (see [20, Theorem 4.4]); we focus on the deterministic case, where this issue disappears. A more problematic source of looseness, specific to the quantum case, is Unruh's "one-way to hiding" lemma; see [20, Lemma 4.1, Theorem 4.5, and Theorem 4.6]. Quantitatively, under the reasonable assumption that the attacker can afford 2^{64} hash queries, an OW-Passive attack that succeeds with probability merely 2^{-128}—not a particularly attractive target for cryptanalysts—is consistent with an IND-CCA2 attack that succeeds with probability 1. Hamburg has sketched an approach to improve 2^{-128} to 2^{-64}, but this is still far below 1.

Saito, Xagawa, and Yamakawa in [36] showed that implicit rejection allows a *tight* QROM IND-CCA2 security proof, assuming that the underlying PKE is deterministic. However, this theorem starts with a ciphertext-unrecognizability security assumption stronger than OW-Passive: roughly, indistinguishability of the legitimate ciphertexts for a particular public key from a much larger space of ciphertexts. There are some citations in [36] to previous statements of special cases of this assumption, but this again raises questions regarding what cryptanalysts have actually studied.

Jiang, Zhang, Chen, Wang, and Ma in [23], independently of [36], showed that implicit rejection allows a QROM IND-CCA2 security proof starting from OW-Passive security, without a plaintext-confirmation hash. However, this theorem is still not tight.

To summarize, the QROM situation is still in flux and has not reached a satisfactory state. The only known way to avoid considerable looseness is to assume something beyond OW-Passive security for the underlying PKE.

We point out that a ROM version of the main theorem of [36] factors into two parts:

- Implicit rejection plus ROM IND-Hash security tightly implies ROM IND-CCA2 security. See Section 13.
- Ciphertext unrecognizability tightly implies ROM IND-Hash security.

This modularization allows auditing of [36] to share work with auditing of our theorems. We expect the tight QROM analysis of [36] to factor similarly via QROM IND-Hash security. These observations suggest that further QROM efforts can and should focus on ways to obtain QROM IND-Hash security. An interesting possibility to consider is the following: QROM IND-Hash security cannot be tightly obtained from OW-Passive security; QROM IND-Hash security is the closest approximation to OW-Passive security that allows a tight connection to QROM IND-CCA2 security; QROM IND-Hash security is the right target for future cryptanalysis.

3 Notation and conventions

3.1. Truth values. The $[S]$ notation means 1 if the statement S is true, otherwise 0.

3.2. Functions. When X and Y are sets, X^Y is the set of functions from Y to X.

3.3. Lazy evaluation of uniform random functions. When we write "Generate a uniform random element $H \in X^Y$", where X is a nonempty finite set and Y is a finite set, we do not mean that we immediately take the time to generate and record a uniform random value $H(y) \in X$ for each $y \in Y$. Instead we delay generating $H(y)$ until it is needed.

In other words, H is an oracle that maintains an associative array of pairs $(y, H(y)) \in Y \times X$ where each index y has been previously queried. If y is queried and is an index in the array, the oracle returns the same $H(y)$. If y is queried and is not already an index in the array, the oracle generates a uniform random element $H(y)$ of X, adds $(y, H(y))$ to the array, and returns $H(y)$.

In cost metrics that account for storage, one should eliminate the array of H values (as in [8, "Notes on low-memory attacks"] and [4]) and instead simulate a uniform random H as the output of a cipher using a uniform random key. Handling a cipher key is cheaper than handling a possibly very large array of uniform random values, and if the cipher is secure then the results are indistinguishable. We state theorems for the uniform random case, and leave it to the reader to deduce theorems regarding these simulations.

3.4. Lazy evaluation of more general functions. If we define, e.g., $G \in (X \times X)^Y$ by $G(y) = (H(y), H(y))$ with H as above, we do not mean that all $G(y)$ values are immediately tabulated. Instead we delay generating each $G(y)$ until it is needed.

3.5. Oracles and other inputs. When we pass a function such as the above H or G as an input to an algorithm, we again do not mean that values of H and G are immediately tabulated. Instead we give the algorithm access to oracles that compute H and G on demand.

Similarly, we delay reading other inputs until they are needed. The exact order of operations does not matter for any of the analysis in this paper (although it does matter for some cost metrics, such as storage). When we refer to one operation happening before another (as in Definition 7.3), we mean that there is data flow from one to the other.

3.6. Reductions. We follow the recommendations of [14, Appendix B.6] to state "provable-security theorems in a way that minimizes the hassle of switching to a new cost metric". Specifically, we maintain a separation between (1) theorems analyzing success probabilities of reductions, (2) definitions of the reductions before the theorems, and (3) analyses of the costs of the same reductions.

All of our reductions—CheckEncrypt in Section 8, RandomDecap in Section 13, and IR in Section 13—have some overhead beyond the costs of the original algorithms. The overhead is bounded by a linear number of simple operations. “Linear” is a small constant per query plus a small constant for initialization; “simple” includes hashing, computing the provided PKE operations such as $\mathsf{Encrypt}$, computing the function F defined later, generating uniform random elements of specified sets such as SKeys, and comparing elements of these sets.

We have not stated formal cost theorems in this paper. We caution the reader that simple-sounding operations can be extremely expensive, depending on the cost metric and on details of the operations. For example, it is an easy exercise to define a nonempty finite set of short strings for which nobody knows a feasible way to generate *any* element of that set, never mind a uniform random element. Despite these caveats, we assert that the simple operations used in our reductions are reasonably cheap for typical examples of PKEs.

4 Correct deterministic PKEs

This section reviews the definition of a PKE, for the important special case of PKEs that are correct and deterministic. **Deterministic** refers to the fact that $\mathsf{Encrypt}$ (see below) is deterministic. **Correct** refers to the fact that $\mathsf{Decrypt}(\mathsf{Encrypt}(p, K), k)$ always outputs p.

Definition 4.1 (correct deterministic PKE). *A* **correct deterministic PKE** *is defined as a tuple*

$$(\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

where

- $\mathsf{PublicKeys}$, $\mathsf{PrivateKeys}$, $\mathsf{Plaintexts}$, $\mathsf{Ciphertexts}$ *are nonempty finite sets;*
- $\perp \notin \mathsf{Plaintexts}$*;*
- KeyGen *is an algorithm mapping* $\{()\}$ *to* $\mathsf{PublicKeys} \times \mathsf{PrivateKeys}$*;*
- $\mathsf{Encrypt}$ *is a deterministic algorithm mapping* $\mathsf{Plaintexts} \times \mathsf{PublicKeys}$ *to* $\mathsf{Ciphertexts}$*;*
- $\mathsf{Decrypt}$ *is an algorithm mapping* $\mathsf{Ciphertexts} \times \mathsf{PrivateKeys}$ *to* $\mathsf{Plaintexts} \cup \{\perp\}$*; and*
- $\mathsf{Decrypt}(\mathsf{Encrypt}(p, K), k) = p$ *for every* (K, k) *output by* $\mathsf{KeyGen}()$ *and every* $p \in \mathsf{Plaintexts}$.

5 OW-Passive (“OW-CPA”) security for a PKE

This section reviews “OW-CPA” security, which we call “OW-Passive” security as explained earlier. Informally, OW-Passive security of a PKE X means that $\mathrm{PrPassive}(X, A)$ defined below is small for every feasible attack A, i.e., that A has trouble finding a uniform random plaintext given a public key and the corresponding ciphertext. “OW-Passive security” is an abbreviation for “one-wayness against passive attacks”. For theorems we work directly with PrPassive.

It is tempting to write simply "OW", but the literature also uses "OW" to describe variants of the same notion that give more power to the attacker. For example, "one-wayness against plaintext-checking attacks" ("OW-PCA") gives the attacker access to an oracle for the function $(p, C) \mapsto [\mathsf{Decrypt}(C, k) = p]$. For a correct deterministic PKE, $\mathsf{Decrypt}(C, k)$ is guaranteed to be p if $C = \mathsf{Encrypt}(p, K)$, but it could also be p for other choices of C, so OW-PCA security could be easier to break than OW-Passive security. This gap disappears for "rigid" correct deterministic PKEs; see Section 6 below.

Definition 5.1 (RunPassive and PrPassive). *Let*

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a correct deterministic PKE. Let A be an algorithm. Then $\mathrm{PrPassive}(X, A)$ *is defined as* $\Pr[\mathrm{RunPassive}(X, A) = 1]$*, where* $\mathrm{RunPassive}(X, A)$ *is defined as the following algorithm:*

- *Compute $(K, k) \leftarrow \mathsf{KeyGen}()$.*
- *Generate a uniform random $p^* \in \mathsf{Plaintexts}$.*
- *Compute $C^* \leftarrow \mathsf{Encrypt}(p^*, K)$.*
- *Output $[A(K, C^*) = p^*]$.*

Note that Decrypt is not used in the definitions of RunPassive and PrPassive. One could define these security notions for a streamlined PKE concept that omits Decrypt and omits private keys. Similar comments apply to Section 7.

6 Rigidity and reencryption

The "T" transformation from [20] converts a correct PKE with OW-Passive security to a correct deterministic PKE with "OW-PCA" security. This section observes that "T" has a stronger property, and then factors "T" into two parts, with useful consequences for simplicity and tightness:

- The stronger property is that the output of "T" is what we call a "rigid" correct deterministic PKE. "Rigid" means that $\mathsf{Decrypt}(C, k) = p \in \mathsf{Plaintexts}$ if and only if $\mathsf{Encrypt}(p, K) = C$. OW-PCA security is then the same as OW-Passive security, so one can avoid defining and discussing OW-PCA security.
- The factorization of "T" is as follows. First, "Derandomize" converts a correct PKE into a correct deterministic PKE, by using a hash of the input as the randomness for encryption. Second, "ReEnc" converts a correct deterministic PKE into a rigid correct deterministic PKE. In this paper, we start with a correct deterministic PKE, so we have no need for Derandomize: we simply apply ReEnc.
- The reductions known for the "T" transformation in general (see [20] and [36]) are not tight. However, when the input is a correct deterministic PKE, one can *tightly* obtain a rigid correct deterministic PKE by simply applying ReEnc. The reduction here is trivial: see Theorem 6.4 below.

One can also view ReEnc as a special case of "T", namely the case that the original PKE is already deterministic, so Derandomize has no effect.

For some PKEs, rigidity can be achieved at lower cost than ReEnc: there are speedups for the decryption in ReEnc, or the private key can be compressed. A notable example is the McEliece code-based encryption scheme. See generally [9, Section 2.5], in particular the explanation of why testing "$C_0 = He$" does not take "quadratic space".

Definition 6.1 (rigidity). *Let*

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a correct deterministic PKE. X is **rigid** *if* $\mathsf{Encrypt}(p, K) = C$ *for every* (K, k) *output by* $\mathsf{KeyGen}()$*, every* $C \in \mathsf{Ciphertexts}$*, and every* $p \in \mathsf{Plaintexts}$ *output by* $\mathsf{Decrypt}(C, k)$*.*

The definition places requirements on any plaintext returned by $\mathsf{Decrypt}(C, k)$. It does not prohibit $\mathsf{Decrypt}(C, k)$ returning $\perp$: on the contrary, it has the effect of requiring $\mathsf{Decrypt}(C, k)$ to return $\perp$ unless C has the form $\mathsf{Encrypt}(p, K)$. In other words, $\mathsf{Decrypt}$ returns $\perp$ as often as it can without sacrificing correctness, and is thus entirely determined by $\mathsf{Encrypt}$ for this key pair: $\mathsf{Decrypt}(C, k)$ is the unique $p \in \mathsf{Plaintexts}$ such that $\mathsf{Encrypt}(p, K) = C$, or $\perp$ if no such p exists.

Definition 6.2 (ReEnc). *Let*

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a correct deterministic PKE. Then $\mathrm{ReEnc}(X)$ *is defined as*

$$(\mathsf{PublicKeys}, \mathsf{PrivateKeys}', \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}', \mathsf{Encrypt}, \mathsf{Decrypt}')$$

where $\mathsf{PrivateKeys}' = \mathsf{PrivateKeys} \times \mathsf{PublicKeys}$*;* $\mathsf{KeyGen}'()$ *computes* $(K, k) \leftarrow \mathsf{KeyGen}()$ *and then outputs* (K, k') *where* $k' = (k, K)$*; and* $\mathsf{Decrypt}'(C, k')$ *is the following algorithm:*

- *Write* k' *as* (k, K) *where* $k \in \mathsf{PrivateKeys}$ *and* $K \in \mathsf{PublicKeys}$*.*
- *Compute* $p \leftarrow \mathsf{Decrypt}(C, k)$*.*
- *If* $p = \perp$*, output* $\perp$ *and stop.*
- *If* $\mathsf{Encrypt}(p, K) \neq C$*, output* $\perp$ *and stop.*
- *Output* p*.*

Theorem 6.3 (correctness and rigidity of Reencrypt). *Let X be a correct deterministic PKE. Then* $\mathrm{ReEnc}(X)$ *is a rigid correct deterministic PKE.*

Proof. As before write

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}).$$

$\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}$ are nonempty finite sets by assumption, so $\mathsf{PublicKeys}, \mathsf{PrivateKeys}', \mathsf{Plaintexts}, \mathsf{Ciphertexts}$ are nonempty finite sets.

$\bot \notin \mathsf{Plaintexts}$ by assumption.

KeyGen maps $\{()\}$ to $\mathsf{PublicKeys} \times \mathsf{PrivateKeys}$ by assumption, so KeyGen' maps $\{()\}$ to $\mathsf{PublicKeys} \times (\mathsf{PrivateKeys} \times \mathsf{PublicKeys}) = \mathsf{PublicKeys} \times \mathsf{PrivateKeys}'$.

$\mathsf{Encrypt}$ is a deterministic algorithm mapping $\mathsf{Plaintexts} \times \mathsf{PublicKeys}$ to $\mathsf{Ciphertexts}$ by assumption.

$\mathsf{Decrypt}$ is an algorithm mapping $\mathsf{Ciphertexts} \times \mathsf{PrivateKeys}$ to $\mathsf{Plaintexts} \cup \{\bot\}$, so $\mathsf{Decrypt}'$ is an algorithm mapping $\mathsf{Ciphertexts} \times \mathsf{PrivateKeys}'$ to $\mathsf{Plaintexts} \cup \{\bot\}$.

To see correctness, observe that if $C = \mathsf{Encrypt}(p, K)$ for $p \in \mathsf{Plaintexts}$ then $\mathsf{Decrypt}(C, k) = p$ since the original PKE is correct, and then $\mathsf{Encrypt}(p, K)$ again outputs C since the original PKE is deterministic, so $\mathsf{Decrypt}'(C, k)$ outputs p.

To see rigidity, observe that the only way for $\mathsf{Decrypt}'(C, k)$ to output $p \in \mathsf{Plaintexts}$ is for p to be output by $\mathsf{Decrypt}(C, k)$ with $\mathsf{Encrypt}(p, K) = C$. □

Theorem 6.4 (security of Reencrypt). *Let X be a correct deterministic PKE. Let A be an algorithm. Then* $\mathrm{PrPassive}(X, A) = \mathrm{PrPassive}(\mathrm{ReEnc}(X), A)$.

In other words, the OW-Passive security of $\mathrm{ReEnc}(X)$ is the same as the OW-Passive security of X.

Proof. RunPassive does not use $\mathsf{Decrypt}$ (or $\mathsf{Decrypt}'$) or k (or k'), and $\mathrm{ReEnc}(X)$ is otherwise identical to X. □

7 IND-Hash security for a PKE

This section defines "ROM IND-Hash" security, compares ROM IND-Hash attacks to OW-Passive attacks, and briefly discusses QROM IND-Hash security. We have not found IND-Hash security in the literature.

Informally, ROM IND-Hash security of a PKE X relative to a set SKeys means that the quantity $\mathrm{DistHash}(X, A, \mathsf{SKeys})$ defined below is small for every feasible non-cheating attack A. DistHash, in turn, measures the ability of A to distinguish a session key from random, where the session key is obtained by hashing a ciphertext. A is given the ciphertext and access to the hash function (D below), but is not allowed to apply the hash function to the ciphertext after seeing the ciphertext—this would be cheating. A is also given the public key and access to another function (H below) that encrypts an input and then applies the first hash function to the result—even if the result is the ciphertext that A has seen; this is *not* cheating.

DistHash is automatically 0 in the trivial case $\#\mathsf{SKeys} = 1$: i.e., ROM IND-Hash security is automatic when there is only one possible session key. Our concern is with ROM IND-Hash security for much larger $\#\mathsf{SKeys}$.

Definition 7.1 (RunHash and DistHash). *Let*

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a correct deterministic PKE. Let A be an algorithm. Let SKeys *be a nonempty finite set. Then* $\mathrm{DistHash}(X, A, \mathsf{SKeys})$ *is defined as*

$$|\Pr[\mathrm{RunHash}_1(X, A, \mathsf{SKeys}) = 1] - \Pr[\mathrm{RunHash}_0(X, A, \mathsf{SKeys}) = 1]|$$

where $\mathrm{RunHash}_b(X, A, \mathsf{SKeys})$ *is the following algorithm:*

- *Compute* $(K, k) \leftarrow \mathsf{KeyGen}()$*.*
- *Generate a uniform random* $p^* \in \mathsf{Plaintexts}$*.*
- *Compute* $C^* \leftarrow \mathsf{Encrypt}(p^*, K)$*.*
- *Generate a uniform random* $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$*.*
- *Define* $H \in \mathsf{SKeys}^{\mathsf{Plaintexts}}$ *as follows:* $H(p) = D(\mathsf{Encrypt}(p, K))$ *for each* $p \in \mathsf{Plaintexts}$*.*
- *Generate a uniform random* $s_0 \in \mathsf{SKeys}$*.*
- *Compute* $s_1 \leftarrow D(C^*)$*.*
- *Return* $A(H, K, D, C^*, s_b)$*.*

7.2. Cheating. Consider the following algorithm A: on input (H, K, D, C, s), output $[s = D(C)]$. Then $\mathrm{RunHash}_1(X, A, \mathsf{SKeys})$ always outputs 1, while $\mathrm{RunHash}_0(X, A, \mathsf{SKeys})$ outputs 1 only if $s_0 = s_1$, which occurs with probability $1/\#\mathsf{SKeys}$. Thus $\mathrm{DistHash}(X, A, \mathsf{SKeys}) = 1 - 1/\#\mathsf{SKeys} \geq 1/2$, assuming $\#\mathsf{SKeys} > 1$.

As mentioned above, ROM IND-Hash security considers only non-cheating attacks, where "cheating" is defined as follows. This type of definition is (modulo input positioning) typically integrated into the definition of IND-CCA2 advantage, but we prefer to factor the definition out of both IND-Hash and IND-CCA2.

Definition 7.3 (cheating). *Let A be an algorithm. We define A as* **cheating** *if, after reading its fourth input or its fifth input, it uses its fourth input as a query to its third input.*

7.4. Comparing IND-Hash to OW-Passive. The obvious way to attack IND-Hash security is to attack OW-Passive security. Given an OW-Passive attack B, consider the following IND-Hash attack A:

- Input (H, K, D, C, s).
- Compute $p \leftarrow B(K, C)$.
- Output $[\mathsf{Encrypt}(p, K) = C$ and $H(p) = s]$.

This is a non-cheating attack, since A does not use the D oracle. If B succeeds, i.e., if $\mathsf{Encrypt}(p, K) = C$, then $H(p) = D(\mathsf{Encrypt}(p, K)) = D(C) = s_1$. Hence $\mathrm{RunHash}_1(X, A, \mathsf{SKeys}) = 1$ with probability $\mathrm{PrPassive}(X, B)$, while $\mathrm{RunHash}_0(X, A, \mathsf{SKeys}) = 1$ with probability $\mathrm{PrPassive}(X, B)/\#\mathsf{SKeys}$; so $\mathrm{DistHash}(X, A, \mathsf{SKeys}) = \mathrm{PrPassive}(X, B)(1 - 1/\#\mathsf{SKeys})$.

Intuitively, there is very little gap between OW-Passive security and IND-Hash security. Both s_0 and $s_1 = D(C^*) = H(p^*)$ are generated uniformly at random; swapping s_0 with s_1 is invisible to an attack that does not call D on input C^* and that does not call H on input p^*. The only way for an attack to call D on input C^* (without cheating) is to blindly guess before seeing C^*, and the only way for an attack to call H on input p^* is to break OW-Passive security. The proof in Section 8 follows this intuition.

7.5. Quantum queries. Our focus is ROM security but we briefly sketch an analogous QROM IND-Hash definition. The only difference is that A is allowed to query its first input, H, in superposition. A is *not* allowed to query its third input, D, in superposition. In our application to IND-CCA2 security, H is a hash that the attacker can compute privately, while D is the legitimate user's decapsulation oracle.

8 CheckEncrypt: ROM IND-Hash security from OW-Passive security

The point of this section is that a ROM IND-Hash attack implies an OW-Passive attack, with similar speed and almost exactly the same success probability.

Definition 8.1 (CheckEncrypt). *Let*

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a correct deterministic PKE. Let SKeys *be a nonempty finite set. Let A be an algorithm. Then* $\mathrm{CheckEncrypt}(X, A, \mathsf{SKeys})$ *is defined as the following algorithm:*

- *Input* $K \in \mathsf{PublicKeys}$.
- *Input* $C^* \in \mathsf{Ciphertexts}$.
- *Generate a uniform random* $s \in \mathsf{SKeys}$.
- *Generate a uniform random* $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$.
- *Define* $H \in \mathsf{SKeys}^{\mathsf{Plaintexts}}$ *as follows:* $H(p) = D(\mathsf{Encrypt}(p, K))$ *for each* $p \in \mathsf{Plaintexts}$.
- *Run* $A(H, K, D, C^*, s)$.
- *For each query p that A makes to H: If* $\mathsf{Encrypt}(p, K) = C^*$, *output p and stop.*

Theorem 8.2 (IND-Hash security from OW-Passive security). *Let X be a correct deterministic PKE. Let* SKeys *be a nonempty finite set. Let A be a non-cheating algorithm that performs at most q queries to its third input. Define* $B = \mathrm{CheckEncrypt}(X, A, \mathsf{SKeys})$. *Then*

$$\mathrm{DistHash}(X, A, \mathsf{SKeys}) \le \mathrm{PrPassive}(X, B) + \frac{q}{\#\mathsf{Plaintexts}}.$$

Proof. Recall that $\mathrm{RunPassive}(X, B)$ computes $(K, k) \leftarrow \mathsf{KeyGen}()$, generates a uniform random $p^* \in \mathsf{Plaintexts}$, computes $C^* \leftarrow \mathsf{Encrypt}(p^*, K)$, and checks whether p^* is the output of $B(K, C^*)$.

By definition, $B(K, C^*)$ runs $A(H, K, D, C^*, s)$ for a uniform random $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$, a uniform random $s \in \mathsf{SKeys}$, and $H(p)$ defined as $D(\mathsf{Encrypt}(p, K))$. The output of $B(K, C^*)$ (if any) is the first query p to H such that $\mathsf{Encrypt}(p, K) = C^*$, i.e., such that $\mathsf{Encrypt}(p, K) = \mathsf{Encrypt}(p^*, K)$, i.e., such that $p = p^*$ (since X is a correct deterministic PKE). In other words, $\mathrm{RunPassive}(X, B)$ outputs 1 if and only if $A(H, K, D, C^*, s)$ queries H on p^*.

For comparison, $\mathrm{RunHash}_0(X, A, \mathsf{SKeys})$ is the following algorithm:

- Compute $(K, k) \leftarrow \mathsf{KeyGen}()$.
- Generate a uniform random $p^* \in \mathsf{Plaintexts}$.
- Compute $C^* \leftarrow \mathsf{Encrypt}(p^*, K)$.
- Generate a uniform random $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$.
- Define $H \in \mathsf{SKeys}^{\mathsf{Plaintexts}}$ as follows: $H(p) = D(\mathsf{Encrypt}(p, K))$ for each $p \in \mathsf{Plaintexts}$.
- Generate a uniform random $s_0 \in \mathsf{SKeys}$.
- Return $A(H, K, D, C^*, s_0)$.

The event that A queries H on p^* inside $\text{RunHash}_0(X, A, \mathsf{SKeys})$ is exactly the event that $\text{RunPassive}(X, B)$ outputs 1: the algorithms are the same, except for relabeling s as s_0.

The probability that $\text{RunHash}_0(X, A, \mathsf{SKeys})$ returns 1 is at most the sum of the following three probabilities:

- The probability that it returns 1 with A querying H on p^*. This probability is at most $\text{PrPassive}(X, B)$.
- The probability that it returns 1 with A querying D on C^*. This probability is at most $q/\#\mathsf{Plaintexts}$: since A is non-cheating, it cannot query D on C^* after seeing C^* or s_0, and before this its queries are independent of p^*, so p^* matches each query with probability at most $1/\#\mathsf{Plaintexts}$.
- The probability that it returns 1 without A querying H on p^* and without A querying D on C^*. Define δ as this probability.

$\text{RunHash}_1(X, A, \mathsf{SKeys})$ works the same way, except that it calls $A(H, K, D, C^*, s_1)$ instead of $A(H, K, D, C^*, s_0)$. Both s_0 and $s_1 = D(C^*) = H(p^*)$ are uniform random elements of SKeys, independent of all other values of D and H, so the only way for A to tell the difference is to query D on C^* or query H on p^*. The probability that A queries H on p^* is again at most $\text{PrPassive}(X, B)$, the probability that A queries D on C^* is again at most $q/\#\mathsf{Plaintexts}$, and the probability that $\text{RunHash}_1(X, A, \mathsf{SKeys})$ returns 1 without these two types of queries is the same δ as before.

Both $\Pr[\text{RunHash}_0(X, A, \mathsf{SKeys})]$ and $\Pr[\text{RunHash}_1(X, A, \mathsf{SKeys})]$ are thus between δ and $\delta + \text{PrPassive}(X, B) + q/\#\mathsf{Plaintexts}$. The absolute difference $\text{DistHash}(X, A, \mathsf{SKeys})$ is at most $\text{PrPassive}(X, B) + q/\#\mathsf{Plaintexts}$. □

9 Correct quiet KEMs

This section reviews the definition of a ROM KEM, in particular for KEMs that are correct and "quiet".

Correct refers to the fact that decapsulating C always outputs s whenever encapsulation outputs (C, s). This is analogous to the correctness concept for PKEs in Section 4.

The literature generally permits KEM decapsulation to output a symbol $\perp$ outside SKeys, the same way that PKE decryption can return a symbol $\perp$ outside

$\mathsf{Plaintexts}$. We consider only **quiet** KEMs, meaning that the output of decapsulation is always in SKeys; this restriction simplifies the KEM interface. We have not found previous terminology for this concept.

Choosing a particular deterministic algorithm H mapping $\mathsf{HashInputs}$ to SKeys, and substituting this into Encap and Decap, produces a simpler type of object, also called a KEM. This paper analyzes ROM security, i.e., the chance of breaking a uniform random function H given oracle access to H; beware that there could be faster attacks against any particular choice of H.

Definition 9.1 (correct quiet ROM KEM). *A* **correct quiet ROM KEM** *is defined as a tuple*

$$(\mathsf{HashInputs}, \mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{SKeys}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$$

where

- $\mathsf{HashInputs}, \mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{SKeys}, \mathsf{Ciphertexts}$ *are nonempty finite sets;*
- KeyGen *is an algorithm mapping* $\{()\}$ *to* $\mathsf{PublicKeys} \times \mathsf{PrivateKeys}$*;*
- Encap *is an algorithm mapping* $\mathsf{SKeys}^{\mathsf{HashInputs}} \times \mathsf{PublicKeys}$ *to* $\mathsf{Ciphertexts} \times \mathsf{SKeys}$*;*
- Decap *is an algorithm mapping* $\mathsf{SKeys}^{\mathsf{HashInputs}} \times \mathsf{Ciphertexts} \times \mathsf{PrivateKeys}$ *to* SKeys*; and*
- $\mathsf{Decap}(H, C, k) = s$ *for every* $H \in \mathsf{SKeys}^{\mathsf{HashInputs}}$*, every* (K, k) *output by* $\mathsf{KeyGen}()$*, and every* (C, s) *output by* $\mathsf{Encap}(H, K)$*.*

10 IND-CCA2 security for a KEM

This section reviews the definition of ROM IND-CCA2 security. Informally, ROM IND-CCA2 security of a ROM KEM Y means that $\mathrm{DistCCA}(Y, A)$ defined below is small for every feasible non-cheating attack A. $\mathrm{DistCCA}(Y, A)$, in turn, is the chance that A can distinguish a session key from uniform. A is given access to the public key, the ciphertext generated by the same encapsulation that generated the session key, and a decapsulation oracle. For theorems we work directly with the definition of DistCCA. Non-cheating has the same definition as in Section 7.

Definition 10.1 (RunCCA and DistCCA). *Let*

$$Y = (\mathsf{HashInputs}, \mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{SKeys}, \mathsf{Ciphertexts}, \\ \mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$$

be a correct quiet ROM KEM. Let A be an algorithm. Then $\mathrm{DistCCA}(Y, A)$ *is defined as*

$$|\Pr[\mathrm{RunCCA}_1(Y, A) = 1] - \Pr[\mathrm{RunCCA}_0(Y, A) = 1]|,$$

where $\mathrm{RunCCA}_b(Y, A)$ *is defined as the following algorithm:*

- *Generate a uniform random* $H \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.
- *Compute* $(K, k) \leftarrow \mathsf{KeyGen}()$.
- *Define* $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$ *by* $D(C) = \mathsf{Decap}(H, C, k)$ *for each* $C \in \mathsf{Ciphertexts}$.
- *Compute* $(C^*, s_1) \leftarrow \mathsf{Encap}(H, K)$.
- *Generate a uniform random* $s_0 \in \mathsf{SKeys}$.
- *Output* $A(H, K, D, C^*, s_b)$.

IND-CCA2 security is often called "IND-CCA" security. However, this risks confusion with the weaker concept of "IND-CCA1" security, in which the attacker is not permitted any access to the decapsulation oracle D after receiving the challenge ciphertext.

Sometimes the literature defines IND-CCA2 success probabilities (modulo notation) as $\Pr[\mathrm{RunCCA}_b(Y, A) = b] - 1/2$, where b is chosen uniformly at random. This produces a result between $-1/2$ and $1/2$, exactly half of the difference $\Pr[\mathrm{RunCCA}_1(Y, A) = 1] - \Pr[\mathrm{RunCCA}_0(Y, A) = 1]$. We prefer working with the absolute difference, which is always between 0 and 1. We have also defined IND-Hash in this way.

11 ROM ciphers, ROM PRF security, and PrefixCipher

Implicit rejection can use any cipher to generate pseudorandom results from invalid ciphertexts. We recommend a particular choice of cipher to simplify constant-time implementations, as noted in Section 1 and explained in Section 15: specifically, we will use a hash function for valid ciphertexts, and we recommend using the same hash function with a secret prefix as a cipher. This recommendation is almost identical to how implicit rejection is handled in [20], except for the slight deviation mentioned in Section 1.

Even though we are recommending a particular cipher, we still modularize the security analysis of this cipher so that it can be audited separately. Formally, this is the analysis of "ROM security" of a "ROM cipher", which is not exactly the same as analysis of security of a cipher: the cipher and the attacker are permitted access to a uniform random hash function. We emphasize that a ROM cipher is much more general than what the literature calls an "ideal cipher": for example, AES is a ROM cipher (making no use of the supplied hash function) and is certainly not an ideal cipher.

Secretly prefixed hash functions are often criticized in the literature as allowing "length-extension attacks" for "Merkle–Damgård" hash functions such as SHA-256. Two convincing ways to avoid this problem are (1) to encode ciphertexts as constant-length strings and (2) to use SHA-3 instead of SHA-2: e.g., SHA3-256 or SHAKE256 instead of SHA-256.

Definition 11.1 (ROM cipher). *A* **ROM cipher** *is defined as a tuple*

$$(\mathsf{HashInputs}, \mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs}, \mathsf{Map})$$

where $\mathsf{HashInputs}$, $\mathsf{CipherKeys}$, Inputs, $\mathsf{Outputs}$ *are nonempty finite sets and* Map *is a deterministic algorithm mapping* $\mathsf{Outputs}^{\mathsf{HashInputs}} \times \mathsf{CipherKeys} \times \mathsf{Inputs}$ *to* $\mathsf{Outputs}$.

Definition 11.2 (RunPRF and DistPRF). *Let*

$$R = (\mathsf{HashInputs}, \mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs}, \mathsf{Map})$$

be a ROM cipher, and let A be an algorithm. Then $\mathrm{DistPRF}(R, A)$ *is defined as*

$$|\Pr[\mathrm{RunPRF}_1(R, A) = 1] - \Pr[\mathrm{RunPRF}_0(R, A) = 1]|,$$

where $\mathrm{RunPRF}_b(R, A)$ *is defined as the following algorithm:*

- *Generate a uniform random* $H \in \mathsf{Outputs}^{\mathsf{HashInputs}}$.
- *Generate a uniform random* $r \in \mathsf{CipherKeys}$.
- *Define* $E_1 \in \mathsf{Outputs}^{\mathsf{Inputs}}$ *by* $E_1(x) = \mathsf{Map}(H, r, x)$ *for each* $x \in \mathsf{Inputs}$.
- *Generate a uniform random* $E_0 \in \mathsf{Outputs}^{\mathsf{Inputs}}$.
- *Output* $A(H, E_b)$.

Definition 11.3 (PrefixCipher). *Let* $\mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs}$ *be nonempty finite sets. Then* $\mathrm{PrefixCipher}(\mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs})$ *is defined as the tuple*

$$(\mathsf{HashInputs}, \mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs}, \mathsf{Map})$$

where $\mathsf{HashInputs} = \mathsf{CipherKeys} \times \mathsf{Inputs}$ *and* $\mathsf{Map}(H, r, x) = H(r, x)$.

Theorem 11.4 (PrefixCipher correctness). *Let* $\mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs}$ *be nonempty finite sets. Define* $R = \mathrm{PrefixCipher}(\mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs})$. *Then R is a ROM cipher.*

Proof. $\mathsf{CipherKeys} \times \mathsf{Inputs}$ is a nonempty finite set, and Map is a deterministic algorithm mapping $\mathsf{Outputs}^{\mathsf{HashInputs}} \times \mathsf{CipherKeys} \times \mathsf{Inputs}$ to $\mathsf{Outputs}$. □

Theorem 11.5 (PrefixCipher security). *Let* $\mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs}$ *be nonempty finite sets. Define* $R = \mathrm{PrefixCipher}(\mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs})$. *Let A be an algorithm that performs at most q queries to its first input. Then* $\mathrm{DistPRF}(R, A) \le q/\#\mathsf{CipherKeys}$.

Proof. By definition $E_1(x) = \mathsf{Map}(H, r, x) = H(r, x)$. This is the only information that A obtains regarding r inside $\mathrm{RunPRF}_1(R, A)$.

In particular, the first hash query is independent of r and thus has chance only $1/\#\mathsf{CipherKeys}$ of using r, i.e., having the form $(r, \dots)$. If this does not occur, then the next query has conditional probability at most $1/(\#\mathsf{CipherKeys}-1)$, and thus absolute probability at most $1/\#\mathsf{CipherKeys}$, of using r. Et cetera. Overall A has probability at most $q/\#\mathsf{CipherKeys}$ of using r in a hash query. The same is true inside $\mathrm{RunPRF}_0(R, A)$.

Write δ for the probability that A outputs 1 *without* using r in a hash query. This probability is the same in $\mathrm{RunPRF}_1(R, A)$ and $\mathrm{RunPRF}_0(R, A)$: if r is not used in a hash query then the query results are independent of all E_1 outputs, and also independent of all E_0 outputs.

Hence $\Pr[\mathrm{RunPRF}_b(R, A) = 1]$ is between δ and $\delta + q/\#\mathsf{CipherKeys}$. The distance between $b = 0$ and $b = 1$ is at most $q/\#\mathsf{CipherKeys}$. □

12 Implicit rejection

This section combines a rigid correct deterministic PKE and a ROM cipher into a correct quiet ROM KEM. The KEM has a further parameter, a function mapping plaintext-ciphertext pairs to a set $\mathsf{HashExtras}$; this unifies "$U^{\not\perp}$" and "$U_m^{\not\perp}$" from [20].

Definition 12.1 (ImplicitRejection). *Let*

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a rigid correct deterministic PKE. Let $\mathsf{HashExtras}$ *and* SKeys *be nonempty finite sets. Let*

$$R = (\mathsf{HashInputs}, \mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs}, \mathsf{Map})$$

be a ROM cipher with $\mathsf{Inputs} = \mathsf{Ciphertexts}$*,* $\mathsf{Outputs} = \mathsf{SKeys}$*, and* $\mathsf{HashInputs} \supseteq \mathsf{Plaintexts} \times \mathsf{HashExtras}$*. Let F be a deterministic algorithm mapping* $\mathsf{Plaintexts} \times \mathsf{Ciphertexts}$ *to* $\mathsf{HashExtras}$*. Then* $\mathrm{ImplicitRejection}(X, R, F)$ *is defined as*

$$(\mathsf{HashInputs}', \mathsf{PublicKeys}, \mathsf{PrivateKeys}', \mathsf{SKeys}, \mathsf{Ciphertexts}, \mathsf{KeyGen}', \mathsf{Encap}, \mathsf{Decap})$$

where the components are defined as follows:

- $\mathsf{HashInputs}' = \{0, 1\} \times \mathsf{HashInputs}$.
- $\mathsf{PrivateKeys}' = \mathsf{PrivateKeys} \times \mathsf{CipherKeys}$.
- $\mathsf{KeyGen}'()$ *computes* $(K, k) \leftarrow \mathsf{KeyGen}()$*, generates a uniform random* $r \in \mathsf{CipherKeys}$*, defines* $k' = (k, r)$*, and outputs* (K, k')*.*
- $\mathsf{Encap}(H, K)$ *generates a uniform random* $p \in \mathsf{Plaintexts}$*, computes* $C \leftarrow \mathsf{Encrypt}(p, K)$*, and outputs* $(C, H(1, p, F(p, C)))$*.*
- $\mathsf{Decap}(H, C, k')$ *parses k' as (k, r); computes* $p \leftarrow \mathsf{Decrypt}(C, k)$*; outputs* $\mathsf{Map}(H_0, r, C)$ *if $p = \perp$, where $H_0(\dots) = H(0, \dots)$; otherwise outputs* $H(1, p, F(p, C))$*.*

12.2. Examples of extra hash inputs. In the situation of Definition 12.1, write $Y = \mathrm{ImplicitRejection}(X, R, F)$. We highlight three special cases of interest for the function F:

- $F(p, C) = ()$, with $\mathsf{HashExtras} = \{()\}$. Then Y uses $H(1, p)$ as a session key. This is similar to "$U_m^{\not\perp}$" from [20], except that Y handles implicit rejection by a separate cipher instead of by hashing.
- $F(p, C) = C$, with $\mathsf{HashExtras} = \mathsf{Ciphertexts}$. Then Y uses $H(1, p, C)$ as a session key. This is similar to "$U^{\not\perp}$" from [20].
- $F(p, C)$ is an intermediate result in recomputing C from p, but not an intermediate result that is easy to compute from C alone: consider, e.g., a PKE that computes C by rounding an intermediate result. What makes this choice of F potentially interesting is that it would force implementors to compute this intermediate result, reducing the incentives for implementors to avoid the recomputation required by ReEnc from Section 6.

The proof techniques also apply to a further generalization in which F takes the public key K as an additional input, and in which Map allows a function of K as an additional input.

Theorem 12.3 (correctness of ImplicitRejection). *In the setting of Definition 12.1,* ImplicitRejection(X, R, F) *is a correct quiet ROM KEM.*

Proof. **Sets.** $\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}$ are nonempty finite sets by definition of PKE; $\mathsf{HashInputs}, \mathsf{CipherKeys}, \mathsf{Outputs} = \mathsf{SKeys}$ are nonempty finite sets by definition of cipher; so $\mathsf{HashInputs}' = \{0,1\} \times \mathsf{HashInputs}$ and $\mathsf{PrivateKeys}' = \mathsf{PrivateKeys} \times \mathsf{CipherKeys}$ are also nonempty finite sets.

Key generation. $\mathsf{KeyGen}()$ produces $K \in \mathsf{PublicKeys}$ and $k \in \mathsf{PrivateKeys}$, so $\mathsf{KeyGen}'()$ produces $K \in \mathsf{PublicKeys}$ and $k' = (k, r) \in \mathsf{PrivateKeys} \times \mathsf{CipherKeys} = \mathsf{PrivateKeys}'$.

Encapsulation. Say $H \in \mathsf{SKeys}^{\mathsf{HashInputs}'}$ and $K \in \mathsf{PublicKeys}$. Then $\mathsf{Encap}(H, K)$ outputs an element of $\mathsf{Ciphertexts} \times \mathsf{SKeys}$:

- $p \in \mathsf{Plaintexts}$ by construction.
- $\mathsf{Encrypt}$ maps $\mathsf{Plaintexts} \times \mathsf{PublicKeys}$ to $\mathsf{Ciphertexts}$ by definition of PKE, so $C = \mathsf{Encrypt}(p, K) \in \mathsf{Ciphertexts}$.
- F maps $\mathsf{Plaintexts} \times \mathsf{Ciphertexts}$ to $\mathsf{HashExtras}$ by hypothesis, so $F(p, C) \in \mathsf{HashExtras}$.
- $(p, F(p, C)) \in \mathsf{Plaintexts} \times \mathsf{HashExtras} \subseteq \mathsf{HashInputs}$.
- $(1, p, F(p, C)) \in \mathsf{HashInputs}'$.
- $H(1, p, F(p, C)) \in \mathsf{SKeys}$.
- $(C, H(1, p, F(p, C))) \in \mathsf{Ciphertexts} \times \mathsf{SKeys}$.

Decapsulation. Say $H \in \mathsf{SKeys}^{\mathsf{HashInputs}'}$, $C \in \mathsf{Ciphertexts}$, and $k' = (k, r) \in \mathsf{PrivateKeys}'$, where $k \in \mathsf{PrivateKeys}$ and $r \in \mathsf{CipherKeys}$. Then $\mathsf{Decap}(H, C, k')$ outputs an element of SKeys:

- $\mathsf{Decrypt}$ maps $\mathsf{Ciphertexts} \times \mathsf{PrivateKeys}$ to $\mathsf{Plaintexts} \cup \{\perp\}$ by definition of PKE, so $p \in \mathsf{Plaintexts} \cup \{\perp\}$.
- If $p \in \mathsf{Plaintexts}$ then $\mathsf{Decap}(H, C, k')$ outputs $H(1, p, F(p, C)) \in \mathsf{SKeys}$.
- Otherwise $H_0 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$, and $\mathsf{Decap}(H, C, k')$ outputs $\mathsf{Map}(H_0, r, C) \in \mathsf{SKeys}$.

Correctness. If $(K, k') \leftarrow \mathsf{KeyGen}'()$ and $(C, s) \leftarrow \mathsf{Encap}(H, K)$, then $\mathsf{Decap}(H, C, k') = s$. Indeed, k' has the form (k, r) where $(K, k) \leftarrow \mathsf{KeyGen}()$, and $C = \mathsf{Encrypt}(p, K)$ for the $p \in \mathsf{Plaintexts}$ generated in Encap, so $\mathsf{Decrypt}(C, k) = p$ by definition of (correct) PKE, so $\mathsf{Decap}(H, C, k') = H(1, p, F(p, C)) = s$. □

13 RandomDecap: ROM IND-CCA2 security from ROM IND-Hash security

This section derives the ROM IND-CCA2 security of ImplicitRejection(X, R, F) tightly from the ROM IND-Hash security of X and the ROM PRF security of R. The point of the proof is that the following two procedures produce the same distribution of the pair (H, D):

- H is a uniform random function, and D is the corresponding IND-CCA2 decapsulation oracle that (1) for valid ciphertexts, decrypts and applies H; (2) for invalid ciphertexts, generates uniform random results.
- D is a uniform random function, and H is the corresponding IND-Hash hash function that encrypts and applies D.

We emphasize that this proof does *not* check whether encryptions match a challenge ciphertext—a separate proof technique used in Theorem 8.2 to relate IND-Hash to OW-Passive.

Specializing Theorem 13.3 to $R = \text{PrefixCipher}(\mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs})$ guarantees ROM PRF security of R by Theorem 11.5, assuming $\#\mathsf{CipherKeys}$ is large enough. The special case $F(p, C) = C$, which we recommend, produces essentially [20, Theorem 3.4], but we use IND-Hash security and ROM PRF security to factor the proof into three conceptually separate parts. The special case $F(p, C) = ()$ fills in a proof of essentially [20, Theorem 3.6].

We highlight two differences between our assumptions and the assumptions in [20]. First, our proofs rely on the assumption that X is rigid, while [20, Theorems 3.4 and 3.6] do not make this assumption; see Appendix A for a counterexample to [20, Theorem 3.6] exploiting this. Second, [20] allows partially correct PKEs, while we focus on the simpler case of correct PKEs, as noted earlier.

Definition 13.1 (RandomDecap). *In the setting of Definition 12.1, let A be an algorithm. Then* RandomDecap(A) *is defined as the following algorithm:*

- *Input H', K, D, C^*, s.*
- *Define $H_1(p, x)$ as $H'(p)$ if $p \in \mathsf{Plaintexts}$ and $x = F(p, \mathsf{Encrypt}(p, K))$.*
- *Generate each other $H_1(\dots)$ uniformly at random.*
- *Generate a uniform random $H_0 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.*
- *Define $H(0, \dots) = H_0(\dots)$ and $H(1, \dots) = H_1(\dots)$.*
- *Output $A(H, K, D, C^*, s)$.*

Note that RandomDecap(A) is non-cheating if A is non-cheating.

Definition 13.2 (IR). *In the setting of Definition 12.1, let A be an algorithm. Then* IR$_b(A)$ *is defined as the following algorithm:*

- *Input $H_0 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.*
- *Input $E \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$.*
- *Generate a uniform random $H_1 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.*
- *Define $H(0, \dots) = H_0(\dots)$ and $H(1, \dots) = H_1(\dots)$.*
- *Compute $(K, k) \leftarrow \mathsf{KeyGen}()$.*
- *Define $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$ as follows: $D(C) = E(C)$ if $\mathsf{Decrypt}(C, k) = \perp$; otherwise $D(C) = H_1(p, F(p, C))$ where $p = \mathsf{Decrypt}(C, k)$.*
- *Generate a uniform random $p^* \in \mathsf{Plaintexts}$.*
- *Compute $C^* \leftarrow \mathsf{Encrypt}(p^*, K)$.*
- *Compute $s_1 = H_1(p^*, F(p^*, C^*))$.*
- *Generate a uniform random $s_0 \in \mathsf{SKeys}$.*
- *Output $A(H, K, D, C^*, s_b)$.*

Theorem 13.3 (IND-CCA2 security from IND-Hash security). *In the setting of Definition 12.1, define* $Y = \text{ImplicitRejection}(X, R, F)$*. Let* A *be a non-cheating algorithm. Define* $A' = \text{RandomDecap}(A)$*. Then* $\text{DistCCA}(Y, A) \le \text{DistHash}(X, A', \mathsf{SKeys}) + \text{DistPRF}(R, \text{IR}_0(A)) + \text{DistPRF}(R, \text{IR}_1(A))$*.*

Proof. Substitute the definition of $\text{ImplicitRejection}(X, R, F)$ into the definition of $\text{RunCCA}_b(Y, A)$, and note that generating a uniform random $H \in \mathsf{SKeys}^{\{0,1\}\times\mathsf{HashInputs}}$ is equivalent to generating independent uniform random $H_0, H_1 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$. The conclusion is that $\text{RunCCA}_b(Y, A)$ works as follows:

- Generate a uniform random $H_0 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.
- Generate a uniform random $H_1 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.
- Define $H(0, \dots) = H_0(\dots)$ and $H(1, \dots) = H_1(\dots)$.
- Compute $(K, k') \leftarrow \mathsf{KeyGen}'()$: i.e., compute $(K, k) \leftarrow \mathsf{KeyGen}()$, generate a uniform random $r \in \mathsf{CipherKeys}$, and define $k' = (k, r)$.
- Define $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$ by $D(C) = \mathsf{Decap}(H, C, k')$ for each $C \in \mathsf{Ciphertexts}$; i.e., $D(C) = \mathsf{Map}(H_0, r, C)$ if $\mathsf{Decrypt}(C, k) = \perp$; otherwise $D(C) = H_1(p, F(p, C))$ where $p = \mathsf{Decrypt}(C, k)$.
- Compute $(C^*, s_1) \leftarrow \mathsf{Encap}(H, K)$; i.e., generate a uniform random $p^* \in \mathsf{Plaintexts}$, compute $C^* \leftarrow \mathsf{Encrypt}(p^*, K)$, and compute $s_1 = H_1(p^*, F(p^*, C^*))$.
- Generate a uniform random $s_0 \in \mathsf{SKeys}$.
- Output $A(H, K, D, C^*, s_b)$.

This is, by definition of IR_b, the same as the following:

- Generate a uniform random $H_0 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.
- Generate a uniform random $r \in \mathsf{CipherKeys}$.
- Define $E_1 \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$ by $E_1(C) = \mathsf{Map}(H_0, r, C)$ for each $C \in \mathsf{Ciphertexts}$.
- Output $\text{IR}_b(A)(H_0, E_1)$.

This is exactly $\text{RunPRF}_1(R, \text{IR}_b(A))$. To summarize so far, $\text{RunCCA}_b(Y, A) = \text{RunPRF}_1(R, \text{IR}_b(A))$. Hence

$$\begin{aligned}
&\text{DistCCA}(Y, A) \\
&\qquad = |\Pr[\text{RunCCA}_1(Y, A) = 1] - \Pr[\text{RunCCA}_0(Y, A) = 1]| \\
&\qquad = |\Pr[\text{RunPRF}_1(R, \text{IR}_1(A)) = 1] - \Pr[\text{RunPRF}_1(R, \text{IR}_0(A)) = 1]|.
\end{aligned}$$

Furthermore, $\Pr[\text{RunPRF}_1(R, \text{IR}_b(A)) = 1]$ is within $\text{DistPRF}(R, \text{IR}_b(A))$ of $\Pr[\text{RunPRF}_0(R, \text{IR}_b(A)) = 1]$; recall that RunPRF_0 is the same as RunPRF_1 except that E_1 is replaced by a uniform random $E_0 \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$. Hence

$$\text{DistCCA}(Y, A) \le \delta + \text{DistPRF}(R, \text{IR}_0(A)) + \text{DistPRF}(R, \text{IR}_1(A))$$

where $\delta = |\Pr[\text{RunPRF}_0(R, \text{IR}_1(A)) = 1] - \Pr[\text{RunPRF}_0(R, \text{IR}_0(A)) = 1]|$. We will finish the proof by showing that δ is exactly $\text{DistHash}(X, A', \mathsf{SKeys})$. For reference we restate $\text{RunPRF}_0(R, \text{IR}_b(A))$:

- Generate a uniform random $H_0 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.
- Generate a uniform random $E_0 \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$.
- Generate a uniform random $H_1 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.
- Define $H(0, \dots) = H_0(\dots)$ and $H(1, \dots) = H_1(\dots)$.
- Compute $(K, k) \leftarrow \mathsf{KeyGen}()$.
- Define $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$ as follows: $D(C) = E_0(C)$ if $\mathsf{Decrypt}(C, k) = \bot$; otherwise $D(C) = H_1(p, F(p, C))$ where $p = \mathsf{Decrypt}(C, k)$.
- Generate a uniform random $p^* \in \mathsf{Plaintexts}$.
- Compute $C^* \leftarrow \mathsf{Encrypt}(p^*, K)$.
- Compute $s_1 = H_1(p^*, F(p^*, C^*))$.
- Generate a uniform random $s_0 \in \mathsf{SKeys}$.
- Output $A(H, K, D, C^*, s_b)$.

Observe that, for each (K, k), different inputs to D cannot produce colliding inputs to H_1: if $(p, F(p, C)) = (p', F(p', C'))$ where $p = \mathsf{Decrypt}(C, k)$ and $p' = \mathsf{Decrypt}(C', k)$ then in particular $p = p'$ (no matter what F is) so $C = \mathsf{Encrypt}(p, K) = \mathsf{Encrypt}(p', K) = C'$. This is where we are using the hypothesis that X is rigid.

Consequently, inside $\mathrm{RunPRF}_0(R, \mathrm{IR}_b(A))$, the values of D are the values on distinct inputs of E_0 and H_1, which are independent uniform random functions. Hence D is a uniform random element of $\mathsf{SKeys}^{\mathsf{Ciphertexts}}$. This is independent of the uniform random element $H_1 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$, except for the relationship $D(C) = H_1(p, F(p, C))$ where $p = \mathsf{Decrypt}(C, k)$, i.e., $D(C) = H_1(p, F(p, C))$ where $C = \mathsf{Encrypt}(p, K)$.

Now replace the constructions of E_0, H_1, D by the following steps:

- Generate a uniform random $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$.
- Define $H'(p) = D(\mathsf{Encrypt}(p, K))$ for each $p \in \mathsf{Plaintexts}$.
- Define $H_1(p, x)$ as $H'(p)$ if $p \in \mathsf{Plaintexts}$ and $x = F(p, \mathsf{Encrypt}(p, K))$.
- Generate each other $H_1(\cdots)$ uniformly at random.

This generates the same distribution of pairs (D, H_1): two uniform random functions that are independent except for the relationship $H_1(p, F(p, C)) = D(C)$ where $C = \mathsf{Encrypt}(p, K)$. There is no effect on the probability that the algorithm outputs 1.

This replacement changes $\mathrm{RunPRF}_0(R, \mathrm{IR}_b(A))$ into the following algorithm:

- Compute $(K, k) \leftarrow \mathsf{KeyGen}()$.
- Generate a uniform random $p^* \in \mathsf{Plaintexts}$.
- Compute $C^* \leftarrow \mathsf{Encrypt}(p^*, K)$.
- Generate a uniform random $D \in \mathsf{SKeys}^{\mathsf{Ciphertexts}}$.
- Define $H'(p) = D(\mathsf{Encrypt}(p, K))$ for each $p \in \mathsf{Plaintexts}$.
- Generate a uniform random $s_0 \in \mathsf{SKeys}$.
- Compute $s_1 = D(C^*)$.
- Define $H_1(p, x)$ as $H'(p)$ if $p \in \mathsf{Plaintexts}$ and $x = F(p, \mathsf{Encrypt}(p, K))$.
- Generate each other $H_1(\dots)$ uniformly at random.
- Generate a uniform random $H_0 \in \mathsf{SKeys}^{\mathsf{HashInputs}}$.

- Define $H(0,\dots) = H_0(\dots)$ and $H(1,\dots) = H_1(\dots)$.
- Output $A(H, K, D, C^*, s_b)$.

This algorithm is exactly $\mathrm{RunHash}_b(X, A', \mathsf{SKeys})$, with H in the definition of $\mathrm{RunHash}_b$ renamed H'. The chance that this algorithm outputs 1 has distance exactly $\mathrm{DistHash}(X, A', \mathsf{SKeys})$ between the cases $b = 0$ and $b = 1$. □

14 SimpleKEM

This section formally defines SimpleKEM, and combines the previous theorems to show that ROM IND-CCA2 security for SimpleKEM follows tightly from OW-Passive security for the underlying PKE.

Definition 14.1 (SimpleKEM). *Let* SKeys *and* $\mathsf{CipherKeys}$ *be nonempty finite sets. Let*

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a correct deterministic PKE with $\mathsf{Plaintexts} \subseteq \mathsf{CipherKeys}$. *Then* $\mathrm{SimpleKEM}(X, \mathsf{SKeys}, \mathsf{CipherKeys})$ *is defined as*

$$(\mathsf{HashInputs}', \mathsf{PublicKeys}, \mathsf{PrivateKeys}'', \mathsf{SKeys}, \mathsf{Ciphertexts}, \mathsf{KeyGen}'', \mathsf{Encap}, \mathsf{Decap})$$

where

- $\mathsf{HashInputs}' = \{0,1\} \times \mathsf{CipherKeys} \times \mathsf{Ciphertexts}$.
- $\mathsf{PrivateKeys}'' = \mathsf{PrivateKeys} \times \mathsf{PublicKeys} \times \mathsf{CipherKeys}$.
- $\mathsf{KeyGen}''()$ *computes* $(K, k) \leftarrow \mathsf{KeyGen}()$, *generates a uniform random* $r \in \mathsf{CipherKeys}$, *defines* $k'' = (k, K, r)$, *and outputs* (K, k'').
- $\mathsf{Encap}(H, K)$ *generates a uniform random* $p \in \mathsf{Plaintexts}$, *computes* $C \leftarrow \mathsf{Encrypt}(p, K)$, *and outputs* $(C, H(1, p, C))$.
- $\mathsf{Decap}(H, C, k'')$ *parses* k'' *as* (k, K, r); *computes* $p \leftarrow \mathsf{Decrypt}(C, k)$; *outputs* $H(1, p, C)$ *if* $p \in \mathsf{Plaintexts}$ *and* $\mathsf{Encrypt}(p, K) = C$; *otherwise outputs* $H(0, r, C)$.

Theorem 14.2 (SimpleKEM decomposition). *Let* SKeys *and* $\mathsf{CipherKeys}$ *be nonempty finite sets. Let*

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a correct deterministic PKE with $\mathsf{Plaintexts} \subseteq \mathsf{CipherKeys}$. *Define* $\mathsf{HashExtras} = \mathsf{Ciphertexts}$. *Define* $F \in \mathsf{HashExtras}^{\mathsf{Plaintexts} \times \mathsf{Ciphertexts}}$ *by* $F(p, C) = C$. *Define* $R = \mathrm{PrefixCipher}(\mathsf{CipherKeys}, \mathsf{Ciphertexts}, \mathsf{SKeys})$. *Then*

$$\mathrm{SimpleKEM}(X, \mathsf{SKeys}, \mathsf{CipherKeys}) = \mathrm{ImplicitRejection}(\mathrm{ReEnc}(X), R, F),$$

and $\mathrm{SimpleKEM}(X, \mathsf{SKeys}, \mathsf{CipherKeys})$ *is a correct quiet ROM KEM.*

The decomposition via PrefixCipher, ReEnc, and ImplicitRejection is overkill for the conclusion that SimpleKEM($X, \mathsf{SKeys}, \mathsf{CipherKeys}$) is a correct quiet ROM KEM, but the same decomposition is also reused for the security analysis.

Proof. Define $X' = \text{ReEnc}(X)$. Recall that

$X' = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}', \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}', \mathsf{Encrypt}, \mathsf{Decrypt}')$

where $\mathsf{PrivateKeys}' = \mathsf{PrivateKeys} \times \mathsf{PublicKeys}$; $\mathsf{KeyGen}'()$ computes $(K, k) \leftarrow \mathsf{KeyGen}()$ and then outputs (K, k') where $k' = (k, K)$; and $\mathsf{Decrypt}'(C, k')$ outputs the p returned by $\mathsf{Decrypt}(C, k)$ if $p \in \mathsf{Plaintexts}$ and $\mathsf{Encrypt}(p, K) = C$, otherwise $\perp$. By Theorem 6.3, X' is a rigid correct deterministic PKE.

$\mathsf{Ciphertexts}$ is a nonempty finite set by definition of PKE, and SKeys is a nonempty finite set by hypothesis, so by definition

$$R = (\mathsf{HashInputs}, \mathsf{CipherKeys}, \mathsf{Inputs}, \mathsf{Outputs}, \mathsf{Map})$$

where $\mathsf{Inputs} = \mathsf{Ciphertexts}$, $\mathsf{Outputs} = \mathsf{SKeys}$, $\mathsf{HashInputs} = \mathsf{CipherKeys} \times \mathsf{Ciphertexts}$, and $\mathsf{Map}(H, r, x) = H(r, x)$; and R is a ROM cipher by Theorem 11.4.

The conditions of Definition 12.1 are satisfied: X' is a rigid correct deterministic PKE; $\mathsf{HashExtras} = \mathsf{Ciphertexts}$ is a nonempty finite set by definition of PKE; SKeys is a nonempty finite set by assumption; R is a ROM cipher with $\mathsf{Inputs} = \mathsf{Ciphertexts}$, $\mathsf{Outputs} = \mathsf{SKeys}$, and $\mathsf{HashInputs} \supseteq \mathsf{Plaintexts} \times \mathsf{HashExtras}$. By definition ImplicitRejection(X', R, F) is

$$(\mathsf{HashInputs}', \mathsf{PublicKeys}, \mathsf{PrivateKeys}'', \mathsf{SKeys}, \mathsf{Ciphertexts}, \\ \mathsf{KeyGen}'', \mathsf{Encap}, \mathsf{Decap})$$

where

- $\mathsf{HashInputs}' = \{0, 1\} \times \mathsf{HashInputs} = \{0, 1\} \times \mathsf{CipherKeys} \times \mathsf{Ciphertexts}$.
- $\mathsf{PrivateKeys}'' = \mathsf{PrivateKeys}' \times \mathsf{CipherKeys} = \mathsf{PrivateKeys} \times \mathsf{PublicKeys} \times \mathsf{CipherKeys}$.
- $\mathsf{KeyGen}''()$ computes $(K, k') \leftarrow \mathsf{KeyGen}'()$, generates a uniform random $r \in \mathsf{CipherKeys}$, defines $k'' = (k', r)$, and outputs (K, k''). Internally $\mathsf{KeyGen}'()$ computes $(K, k) \leftarrow \mathsf{KeyGen}()$ and sets $k' = (k, K)$, so $k'' = (k, K, r)$.
- $\mathsf{Encap}(H, K)$ generates a uniform random $p \in \mathsf{Plaintexts}$, computes $C \leftarrow \mathsf{Encrypt}(p, K)$, and outputs $(C, H(1, p, F(p, C))) = (C, H(1, p, C))$.
- $\mathsf{Decap}(H, C, k'')$ parses k'' as (k', r); computes $p \leftarrow \mathsf{Decrypt}'(C, k')$; outputs $H(1, p, F(p, C))$ if $p \in \mathsf{Plaintexts}$; otherwise outputs $\mathsf{Map}(H_0, r, C)$, where $H_0(\dots) = H(0, \dots)$. In other words, it parses k'' as (k, K, r); computes $p \leftarrow \mathsf{Decrypt}(C, k)$; outputs $H(1, p, C)$ if $p \in \mathsf{Plaintexts}$ and $\mathsf{Encrypt}(p, K) = C$; otherwise outputs $H_0(r, C) = H(0, r, C)$.

This is exactly SimpleKEM($X, \mathsf{SKeys}, \mathsf{CipherKeys}$). Finally, by Theorem 12.3, this is a correct quiet ROM KEM. □

Theorem 14.3 (SimpleKEM security). *In the setting of Theorem 14.2, define* $X' = \mathrm{ReEnc}(X)$ *and* $Y = \mathrm{SimpleKEM}(X, \mathsf{SKeys}, \mathsf{CipherKeys})$*. Let* A *be a non-cheating algorithm that performs at most* q_{Hash0} *queries of the form* $(0, \dots)$ *to its first input and at most* q_{Decap} *queries to its third input. Define* $A' = \mathrm{RandomDecap}(A)$ *and* $B = \mathrm{CheckEncrypt}(X', A', \mathsf{SKeys})$*. Then*

$$\mathrm{DistCCA}(Y, A) \le \mathrm{PrPassive}(X, B) + \frac{q_{\mathrm{Decap}}}{\#\mathsf{Plaintexts}} + \frac{2q_{\mathrm{Hash0}}}{\#\mathsf{CipherKeys}}.$$

Proof. $Y = \mathrm{ImplicitRejection}(X', R, F)$ by Theorem 14.2. The hypotheses of Theorem 13.3 are satisfied, so $\mathrm{DistCCA}(Y, A) \le \mathrm{DistHash}(X', A', \mathsf{SKeys}) + \mathrm{DistPRF}(R, \mathrm{IR}_0(A)) + \mathrm{DistPRF}(R, \mathrm{IR}_1(A))$.

The only way for $\mathrm{IR}_b(A)$ to query its first input, H_0 in Definition 13.2, is for A to query its first input on $(0, \dots)$. This occurs at most q_{Hash0} times. Hence $\mathrm{DistPRF}(R, \mathrm{IR}_b(A)) \le q_{\mathrm{Hash0}}/\#\mathsf{CipherKeys}$ by Theorem 11.5.

The only way for $A' = \mathrm{RandomDecap}(A)$ to query its third input is for A to query its third input. This occurs at most q_{Decap} times. Also note that A' is non-cheating. Hence

$$\mathrm{DistHash}(X', A', \mathsf{SKeys}) \le \mathrm{PrPassive}(X', B) + \frac{q_{\mathrm{Decap}}}{\#\mathsf{Plaintexts}}$$

by Theorem 8.2; and $\mathrm{PrPassive}(X', B) = \mathrm{PrPassive}(X, B)$ by Theorem 6.4. □

15 Constant-time algorithms

This section presents an algorithm for $\mathrm{SimpleKEM}(X, \dots)$ that follows the standard discipline—verified by existing tools—of avoiding data flow from secrets to array indices and branch conditions. This section assumes that such algorithms are already provided for the underlying PKE X and the hash function H. The techniques here are standard, but the details of the resulting algorithms are important for comparison to alternatives.

There is no difficulty in key generation and encapsulation. The task here is to eliminate the two branches in decapsulation. First, $p \leftarrow \mathsf{Decrypt}(C, k)$ could fail, i.e., p could be $\perp$. Second, even if decryption succeeds, the test $C = \mathsf{Encrypt}(p, K)$ could fail.

We assume that elements of $\mathsf{CipherKeys}$, including elements of $\mathsf{Plaintexts}$, are encoded as constant-length byte strings. We also assume that the output of $\mathsf{Decrypt}(C, k)$ in $\mathsf{Plaintexts} \cup \{\perp\}$ is encoded as an element (b, x) of $\{0, 1\} \times \mathsf{Plaintexts}$: specifically, $p \in \mathsf{Plaintexts}$ is encoded as $(1, p)$, and $\perp$ is encoded as $(0, z)$ for a standard choice of $z \in \mathsf{Plaintexts}$.

Whether or not $b = 1$, we perform each of the following steps, using a constant-time algorithm for each step:

- Compute $C' = \mathsf{Encrypt}(x, K)$.
- Compute $c = [C = C']$. There are various well-known algorithms for constant-time comparison: e.g., computing $C \oplus C'$ where $\oplus$ is xor of a representation as constant-length byte strings; then or'ing the resulting bits together; then complementing the result.

- Set $b \leftarrow bc$. At this point $b = 1$ if and only if decryption and reencryption both succeeded.
- Replace x with r if $b = 0$ in constant time. There are various well-known algorithms for constant-time conditional moves: e.g., setting $x \leftarrow b(x \oplus r) \oplus r$, where $b(x \oplus r)$ means using b to mask each bit of $x \oplus r$.
- Compute $H(b, x, C)$.

For comparison, a constant-time algorithm for explicit rejection would have

- the same main computations: $\mathsf{Decrypt}$, $\mathsf{Encrypt}$, H;
- minor simplifications: it would skip the replacement of x with r if $b = 0$, and would skip the generation and storage of r; and
- minor complications: it would replace the output with a standard choice of $z' \in \mathsf{SKeys}$ if $b = 0$, and it would have b as an additional output.

A *variable-time* algorithm for explicit rejection can skip the management of b in favor of storing the same information in the instruction pointer: it immediately returns $\perp$ if $\mathsf{Decrypt}(C, k)$ does, and it immediately returns $\perp$ if $C \neq C'$. We do not recommend this algorithm: the leak of information through timing is dangerous.

16 The case for implicit rejection

Many NIST submissions cite [20], which gives proofs for both implicit rejection and explicit rejection. These submissions are split between choosing implicit rejection (Frodo, KINDI, Kyber, LAC, Lizard, NewHope, Round2, Saber, SIKE, and Titanium) and choosing explicit rejection (BIG QUAKE, DAGS, EMBLEM, HQC, Lepton, LIMA, LOCKER, RQC, and ThreeBears). We have found only a few brief discussions of the reasons for these choices.

This section analyzes various arguments for and against implicit rejection. Our overall assessment is that implicit rejection has a much stronger case. ThreeBears intentionally selected explicit rejection (see below), but all other use of explicit rejection appears to be explainable by the fact that implicit rejection is quite new. We have not found any reference to implicit rejection before [32] in 2012, and the first direct comparison between implicit and explicit rejection was [20] in 2017.

16.1. Arguments for implicit rejection. We have three central reasons for recommending implicit rejection. First, implicit rejection enables various proof strategies and theorems that are not known to be achievable with explicit rejection, while there is nothing the other way around. In particular, tight KEM constructions with explicit rejection appear to require ciphertext expansion (for plaintext confirmation) with more complicated proofs, or stronger assumptions than OW-Passive for the underlying PKE.

Second, quiet KEMs cannot use explicit rejection. Quiet KEMs have a simpler API than non-quiet KEMs. Quiet KEMs remove the need for applications to check for KEM failures: as the Kyber submission comments, a KEM with implicit rejection is “safe to use even if higher level protocols fail to check the return value”.

Third, explicit rejection provides a noticeable incentive for implementors to switch from constant-time algorithms to simpler variable-time algorithms. This is a bad incentive, since variable-time algorithms complicate auditing and presumably create security problems. This incentive is smaller for implicit rejection. See Section 15.

16.2. Arguments for explicit rejection, and counterarguments. Better proofs are not necessarily correlated with better security against attacks. We consider five arguments that explicit rejection could be more secure, or otherwise more desirable, than implicit rejection:

- Implicit rejection outputs more information than explicit rejection, namely cipher output under a secret key. One could argue that this is a risk.
- Explicit rejection has been studied for many more years than implicit rejection.
- Implicit rejection is another “target for side-channel analysis”.
- Explicit rejection is “faster” than implicit rejection.
- Explicit rejection is “simpler” than implicit rejection.

The last three items are stated in the ThreeBears submission [**18**, page 21]; [**23**] also comments that explicit rejection has “relatively simple decapsulation”.

We now comment on each of these arguments.

Regarding the extra outputs: The central cipher security goal is indistinguishability of the output from uniform, implying that this extra information is useless—something that the attacker could simply have generated at random. There is a long history of using standard hash functions as ciphers and conjecturing that they meet the indistinguishability goal; see, e.g., [**7**]. The risk of these conjectures being disproven appears to be far less probable than commonly accepted risks in public-key cryptography.

Regarding age: There are many successful chosen-ciphertext attacks that, in retrospect, exploited explicit rejection and would have been stopped by implicit rejection. The question is not merely whether something has been studied, but how well it has resisted attack.

Regarding side-channel attacks: The bigger picture is that decapsulation handles secret data (the private key; private-key-dependent failure conditions; secret plaintexts) and must be defended against side-channel attacks in environments that allow such attacks. There is extensive literature on defenses, and we have not heard reasons to believe that implicit rejection changes the difficulty of defending the computation. Note that valid computations apply the same hash function to secret data, so the hashing needs to be defended in any case.

Regarding speed: There is a slight speedup, but it is not clear why this speedup is beneficial. Rejection occurs only when one is under attack (plus rare occasions that an accidental bit flip was not caught by the network’s error-correcting codes), and there is no obvious benefit to giving the attacker faster rejections.

Finally, regarding simplicity: Constant-time implementations are no simpler for explicit rejection than for implicit rejection. See Section 15. *Variable-time* implementations are slightly simpler, but this is not a positive feature; see our third argument for implicit rejection.

17 Plaintext confirmation

The general strategy of a chosen-ciphertext attack is to (1) modify the target ciphertext in various ways and (2) inspect the results of decapsulation of the modified ciphertexts. The two different tight ROM proof strategies mentioned in Section 1, plaintext confirmation and implicit rejection, correspond to two different defenses against these attacks, on top of the basic defenses provided by rigidity and hashing:

- Rigidity means that a modified ciphertext cannot produce the same plaintext.
- Sometimes a modified ciphertext produces a modified plaintext. The resulting session keys—strong hashes of the modified plaintexts—look random, having no obvious relationship to the target session key. (The role of hashing here is emphasized in [**38**].)
- Sometimes modified ciphertexts are invalid. Implicit rejection also produces random-looking session keys in these cases, so it hides the pattern of valid ciphertexts from the attacker.
- Plaintext confirmation stops an earlier stage of the attack. With plaintext confirmation, the only way for the attacker to produce a new valid ciphertext is to already know the plaintext, and decapsulation simply provides the same plaintext again, fed through a public hash function. The pattern of valid modified ciphertexts is thus nonexistent.

One can use both defenses together. For example, Classic McEliece [**9**] combines plaintext confirmation with implicit rejection. The original PKE X is first extended with a plaintext-confirmation hash H_2 to produce a new PKE "$X_2 = \text{ConfirmPlaintext}(X, H_2)$", and then X_2 is fed through SimpleKEM to produce a KEM.

OW-Passive security of X tightly implies OW-Passive security of X_2, which in turn implies (by our theorems) ROM IND-CCA2 security for the KEM. For comparison, OW-Passive security of X also implies (again by our theorems) ROM IND-CCA2 security for the KEM obtained by applying SimpleKEM directly to X. This comparison does not show any advantages for the dual-defense construction in [**9**].

On the other hand, the actual goal is more than ROM IND-CCA2 security. Perhaps combining both defenses allows a tight proof of QROM IND-CCA2 security; this would add confidence that is missing without such a proof. Given

- the fact that the defenses target different aspects of attacks,
- the small costs of computing and communicating an extra hash, and

- the current lack of understanding of QROM security,

it seems difficult to justify a recommendation against the dual-defense construction.

References

[1] Carlisle Adams, Jan Camenisch (editors), *Selected areas in cryptography—SAC 2017, 24th international conference, Ottawa, ON, Canada, August 16–18, 2017, revised selected papers*, Lecture Notes in Computer Science, 10719, Springer, 2018. ISBN 978-3-319-72564-2. See [11].

[2] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Gilles Zémor, *Rank Quasi-Cyclic (RQC)*, "Documentation describing the scheme" (2017). URL: `http://pqc-rqc.org/documentation.html`. Citations in this document: §1.

[3] Martin Albrecht, Carlos Cid, Kenneth G. Paterson, CJ Tjhai, Martin Tomlinson, *NTS-KEM*, "The main document submitted to NIST" (2017). URL: `https://nts-kem.io/`. Citations in this document: §1.

[4] Benedikt Auerbach, David Cash, Manuel Fersch, Eike Kiltz, *Memory-tight reductions*, in Crypto 2017 [**26**] (2017), 101–132. Citations in this document: §3.3.

[5] Gustavo Banegas, Paulo S. L. M. Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiécoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, Duc Tri Nguyen, Edoardo Persichetti, Jefferson E. Ricardini, *DAGS: key encapsulation from dyadic GS codes*, "Specification" (2017). URL: `https://www.dags-project.org/#files`. Citations in this document: §1.

[6] Magali Bardet, Elise Barelli, Olivier Blazy, Rodolfo Canto-Torres, Alain Couvreur, Philippe Gaborit, Ayoub Otmani, Nicolas Sendrier, Jean-Pierre Tillich, *BIG QUAKE: BInary Goppa QUAsi-cyclic Key Encapsulation*, "Submitted version" (2017). URL: `https://bigquake.inria.fr/documentation/`. Citations in this document: §1.

[7] Mihir Bellare, Ran Canetti, Hugo Krawczyk, *Keying hash functions for message authentication*, in Crypto 1996 [**28**] (1996), 1–15. URL: `https://cseweb.ucsd.edu/~mihir/papers/hmac.html`. Citations in this document: §16.2.

[8] Daniel J. Bernstein, *Extending the Salsa20 nonce*, Workshop Record of Symmetric Key Encryption Workshop 2011 (2011). URL: `https://cr.yp.to/papers.html#xsalsa`. Citations in this document: §3.3.

[9] Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Wen Wang, *Classic McEliece: conservative code-based cryptography*, "Supporting Documentation" (2017). URL: `https://classic.mceliece.org/nist.html`. Citations in this document: §1, §2.2, §2.3, §6, §17, §17.

[10] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, Christine van Vredendaal, *NTRU Prime*, "Supporting Documentation" (2017). URL: `https://ntruprime.cr.yp.to/nist.html`. Citations in this document: §1.

[11] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, Christine van Vredendaal, *NTRU Prime: reducing attack surface at low cost*, in SAC 2017 [**1**] (2018), 235–260. URL: `https://ntruprime.cr.yp.to/papers.html`. Citations in this document: §1, §2.2.

[12] Daniel J. Bernstein, Josh Fried, Nadia Heninger, Paul Lou, Luke Valenta, *Post-quantum RSA* (2017). URL: https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. Citations in this document: §1.

[13] Daniel J. Bernstein, Leon Groot Bruinderink, Tanja Lange, Lorenz Panny, *HILA5 Pindakaas: On the CCA security of lattice-based encryption with error correction*, in Africacrypt 2018 [**24**] (2018), 203–216. Citations in this document: §1.

[14] Daniel J. Bernstein, Tanja Lange, *Non-uniform cracks in the concrete: the power of free precomputation*, in Asiacrypt 2013 [**37**] (2013), 321–340. URL: https://eprint.iacr.org/2012/318. Citations in this document: §3.6.

[15] Alexander W. Dent, *A designer's guide to KEMs*, in [**31**] (2003), 133–151. URL: https://eprint.iacr.org/2002/174. Citations in this document: §2.2, §2.2, §2.2.

[16] Wieland Fischer, Naofumi Homma (editors), *Cryptographic hardware and embedded systems—CHES 2017—19th international conference, Taipei, Taiwan, September 25–28, 2017, proceedings*, Lecture Notes in Computer Science, 10529, Springer, 2017. ISBN 978-3-319-66786-7. See [21].

[17] Eiichiro Fujisaki, Tatsuaki Okamoto, *Secure integration of asymmetric and symmetric encryption schemes*, in Crypto 1999 [**44**] (1999), 537–554. Citations in this document: §2.

[18] Mike Hamburg, *Post-quantum cryptography proposal: ThreeBears* (2017). URL: https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. Citations in this document: §16.2.

[19] Martin Hirt, Adam D. Smith (editors), *Theory of cryptography—14th international conference, TCC 2016-B, Beijing, China, October 31–November 3, 2016, proceedings, part II*, Lecture Notes in Computer Science, 9986, Springer, 2016. ISBN 978-3-662-53643-8. See [43].

[20] Dennis Hofheinz, Kathrin Hövelmanns, Eike Kiltz, *A modular analysis of the Fujisaki-Okamoto transformation*, in TCC 2017-1 [**25**] (2017), 341–371. URL: https://eprint.iacr.org/2017/604. Citations in this document: §1, §1, §1, §2, §2.2, §2.2, §2.2, §2.2, §2.2, §2.2, §2.2, §2.3, §2.3, §2.4, §2.4, §2.4, §6, §6, §11, §12, §12.2, §12.2, §13, §13, §13, §13, §13, §13, §16, §16, §A, §A, §A.1, §A.2, §A.2, §A.2, §A.2, §A.2, §A.3, §A.3, §A.3, §A.3, §A.3, §A.3, §A.3, §A.3, §A.4, §A.4, §A.4.

[21] Andreas Hülsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, *High-speed key encapsulation from NTRU*, in CHES 2017 [**16**] (2017), 232–252. Citations in this document: §2.4, §2.4.

[22] Andreas Hülsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, *NTRU-HRSS-KEM: algorithm specifications and supporting documentation* (2017). URL: https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. Citations in this document: §1.

[23] Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, Zhi Ma, *Post-quantum IND-CCA-secure KEM without additional hash* (2017). URL: https://eprint.iacr.org/2017/1096. Citations in this document: §2.4, §16.2.

[24] Antoine Joux, Abderrahmane Nitaj, Tajjeeddine Rachidi (editors), *Progress in cryptology—AFRICACRYPT 2018—10th international conference on cryptology in Africa, Marrakesh, Morocco, May 7–9, 2018, proceedings*, Lecture Notes in Computer Science, 10831, Springer, 2018. ISBN 978-3-319-89338-9. See [13].

[25] Yael Kalai, Leonid Reyzin (editors), *Theory of cryptography—15th international conference, TCC 2017, Baltimore, MD, USA, November 12–15, 2017, proceedings, part I*, Lecture Notes in Computer Science, 10677, Springer, 2017. ISBN 978-3-319-70499-9. See [20].

[26] Jonathan Katz, Hovav Shacham (editors), *Advances in cryptology—CRYPTO 2017—37th annual international cryptology conference, Santa Barbara, CA, USA, August 20–24, 2017, proceedings, part I*, Lecture Notes in Computer Science, 10401, Springer, 2017. ISBN 978-3-319-63687-0. See [4].

[27] Joe Kilian (editor), *Advances in cryptology—CRYPTO 2001, 21st annual international cryptology conference, Santa Barbara, California, USA, August 19–23, 2001, proceedings*, Lecture Notes in Computer Science, 2139, Springer, 2001. ISBN 3-540-42456-3. MR 2003d:94002. See [39].

[28] Neal Koblitz (editor), *Advances in cryptology—CRYPTO '96, 16th annual international cryptology conference, Santa Barbara, California, USA, August 18–22, 1996, proceedings*, Lecture Notes in Computer Science, 1009, Springer, 1996. ISBN 3-540-61512-1. See [7].

[29] Dustin Moody, *Re: qrom security etc.* (2017). URL: https://groups.google.com/a/list.nist.gov/d/msg/pqc-forum/WxRmVPhAENw/I5yhtr9dAgAJ. Citations in this document: §1.

[30] Jesper Buus Nielsen, Vincent Rijmen (editors), *Advances in cryptology—EUROCRYPT 2018—37th annual international conference on the theory and applications of cryptographic techniques, Tel Aviv, Israel, April 29–May 3, 2018, proceedings, part III*, Lecture Notes in Computer Science, 10822, Springer, 2018. ISBN 978-3-319-78371-0. See [36].

[31] Kenneth G. Paterson (editor), *Cryptography and coding, 9th IMA international conference, Cirencester, UK, December 16–18, 2003, proceedings*, Lecture Notes in Computer Science, 2898, Springer, 2003. ISBN 3-540-20663-9. See [15].

[32] Edoardo Persichetti, *Improving the efficiency of code-based cryptography*, Ph.D. thesis, 2012. URL: http://persichetti.webs.com/Thesis%20Final.pdf. Citations in this document: §1, §1, §1, §2.2, §2.3, §2.3, §16.

[33] Edoardo Persichetti, *Re: qrom security etc.* (2017). URL: https://groups.google.com/a/list.nist.gov/d/msg/pqc-forum/WxRmVPhAENw/4cI-7n9cAgAJ. Citations in this document: §1.

[34] Thomas Plantard, *Odd Manhattan's algorithm specifications and supporting documentation* (2017). URL: https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. Citations in this document: §1.

[35] Bart Preneel (editor), *Advances in cryptology—EUROCRYPT 2000, international conference on the theory and application of cryptographic techniques, Bruges, Belgium, May 14–18, 2000, proceeding*, Lecture Notes in Computer Science, 1807, Springer, 2000. ISBN 3-540-67517-5. See [38].

[36] Tsunekazu Saito, Keita Xagawa, Takashi Yamakawa, *Tightly-secure key-encapsulation mechanism in the quantum random oracle model*, in Eurocrypt 2018 [**30**] (2018), 520–551. URL: https://eprint.iacr.org/2017/1005. Citations in this document: §2.3, §2.4, §2.4, §2.4, §2.4, §2.4, §2.4, §6.

[37] Kazue Sako, Palash Sarkar (editors), *Advances in cryptology—Asiacrypt 2013—19th international conference on the theory and application of cryptology and information security, Bengaluru, India, December 1–5, 2013, proceedings, part II*, Lecture Notes in Computer Science, 8270, Springer, 2013. ISBN 978-3-642-42045-0. See [14].

[38] Victor Shoup, *Using hash functions as a hedge against chosen ciphertext attack*, in Eurocrypt 2000 [**35**] (2000), 275–288. Citations in this document: §2.1, §17.

[39] Victor Shoup, *OAEP reconsidered*, in Crypto 2001 [**27**] (2001), 239–259. URL: http://shoup.net/papers/oaep.pdf. Citations in this document: §2.

[40] Victor Shoup, *A proposal for an ISO standard for public key encryption*, version 2.1 (2001). URL: http://shoup.net/papers/iso-2_1.pdf. Citations in this document: §1, §2.1.

[41] Nigel P. Smart, *LIMA: a PQC encryption scheme*, talk slides (2018). URL: https://csrc.nist.gov/CSRC/media/Presentations/Lima/images-media/LIMA-April2018.pdf. Citations in this document: §1.

[42] Nigel P. Smart, Martin R. Albrecht, Yehuda Lindell, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, *LIMA: a PQC encryption scheme*, "PDF" under "Submission" (2017). URL: https://lima-pq.github.io/. Citations in this document: §1.

[43] Ehsan Ebrahimi Targhi, Dominique Unruh, *Post-quantum security of the Fujisaki-Okamoto and OAEP transforms*, in TCC 2016-B [19] (2016), 192–216. Citations in this document: §2.4.

[44] Michael J. Wiener (editor), *Advances in cryptology—CRYPTO '99, 19th annual international cryptology conference, Santa Barbara, California, USA, August 15–19, 1999, proceedings*, Lecture Notes in Computer Science, 1666, Springer, 1999. ISBN 3-540-66347-9. See [17].

A The importance of rigidity

This appendix gives counterexamples to [20, Theorem 3.6] and [20, Theorem 3.5]. We make the plausible assumption that there exists some efficient correct deterministic PKE with OW-Passive security.

A.1. The example. Let

$$X = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

be a correct deterministic PKE with $\#\mathsf{Plaintexts} \geq 2$. We build a non-rigid correct deterministic PKE

$$X' = (\mathsf{PublicKeys}, \mathsf{PrivateKeys}, \mathsf{Plaintexts}, \mathsf{Ciphertexts}', \mathsf{KeyGen}, \mathsf{Encrypt}', \mathsf{Decrypt}')$$

as follows:

- $\mathsf{Ciphertexts}' = \{0,1\} \times \mathsf{Ciphertexts}$.
- $\mathsf{Encrypt}'(p, K) = (0, \mathsf{Encrypt}(p, K))$.
- $\mathsf{Decrypt}'((0, C), k) = \mathsf{Decrypt}(C, k)$.
- $\mathsf{Decrypt}'((1, C), k) = \mathsf{Decrypt}(C, k)$.

This is non-rigid because the encryption of p, namely $(0, C)$ where $C = \mathsf{Encrypt}(p, K)$, is not the only ciphertext that decrypts to p: the ciphertext $(1, C)$ also decrypts to p.

The "$U_m^{\perp}$" and "$U_m^{\not\perp}$" transformations in [20], given X' as input, produce KEMs with the following behavior on legitimate inputs:

- Encapsulation chooses a uniform random plaintext p, outputs $H(p)$ as a session key, and outputs $\mathsf{Encrypt}'(p, K) = (0, \mathsf{Encrypt}(p, K))$ as a ciphertext.

- Decapsulation of C produces $H(p)$ as a session key, where $p = \mathsf{Decrypt}'(C, k)$, assuming $p \in \mathsf{Plaintexts}$.

The transformations vary in how decapsulation handles invalid inputs, but we will not need any invalid inputs for the following attack.

The attack, given a challenge ciphertext $C^* = (0, \mathsf{Encrypt}(p^*, K))$, decapsulates the different ciphertext $C = (1, \mathsf{Encrypt}(p^*, K))$. This ciphertext also decrypts to p^*, so decapsulation produces $H(p^*)$. The attack outputs 1 if the output of decapsulation matches the challenge session key, otherwise 0. This breaks IND-CCA2 with probability $1 - 1/\#\mathsf{SKeys}$.

This attack can be generalized beyond X': all one needs is the ability to modify a ciphertext to produce another ciphertext decrypting to the same plaintext. This ability is naturally provided by, e.g., various lattice-based PKEs. One can also loosen the correctness assumption: occasional decryption failures do not noticeably affect the attack.

A.2. Easy to stop, but still important. SimpleKEM applies ReEnc to convert a correct deterministic PKE into a rigid correct deterministic PKE, and then applies ImplicitRejection to obtain a KEM. The ReEnc step prevents any other ciphertext from decrypting to the same plaintext, so it stops the above attack.

Similarly, if one feeds the above PKE X' through "T" from [20] before applying "$U_m^{\not\perp}$", then the attack disappears, since "T" also prevents any other ciphertext from decrypting to the same plaintext.

However, [20, Theorem 3.6] claims security properties for "$U_m^{\not\perp}$" by itself (the same way that we claim security properties for ImplicitRejection by itself), without the extra protection provided by first applying "T". If a KEM designer is starting from a deterministic PKE and sees [20, Theorem 3.6], which claims applicability to any deterministic PKE, then why should the KEM designer bother to use "T"? The attack shows that skipping "T" is dangerous, but this is not what [20, Theorem 3.6] says.

Even if a KEM designer does use both "T" and "$U_m^{\not\perp}$", there is an auditing problem. Auditors should be able to separately audit the security properties claimed for "T" by itself, the security properties claimed for "$U_m^{\not\perp}$" by itself, and the composition of those properties; but the security properties claimed for "$U_m^{\not\perp}$" by itself are not correct.

A fix is to change the interface between "T" and "$U_m^{\not\perp}$": define rigidity, observe that the output of "T" is rigid, and add rigidity as an assumption to [20, Theorem 3.6]. Of course, an auditor will still want to see a proof of the theorem.

A.3. Details of [20, Theorem 3.6]. We now review the details of [20, paper version "September 26, 2017", Theorem 3.6], along with the underlying definitions, to show that our example is a counterexample.

A "public-key encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$" is defined in [20, page 7] as consisting of "three algorithms and a finite message space $\mathcal{M}$ (which we assume to be efficiently recognizable)":

- Gen outputs "a key pair (pk, sk), where pk also defines a randomness space $\mathcal{R} = \mathcal{R}(pk)$". We define Gen_1 as KeyGen from our deterministic PKE X, and $\mathcal{R}_1 = \{()\}$.
- Enc, given pk and $m \in \mathcal{M}$, outputs "an encryption $c \leftarrow \mathsf{Enc}(pk, m)$ of m under the public key pk". We define $\mathcal{M} = \mathsf{Plaintexts}$ and $\mathsf{Enc}_1(pk, m) = \mathsf{Encrypt}'(m, pk) = (0, \mathsf{Encrypt}(m, pk))$.
- Dec, given sk and a ciphertext c, outputs "either a message $m = \mathsf{Dec}(sk, c) \in \mathcal{M}$ or a special symbol $\perp \notin \mathcal{M}$". We define $\mathsf{Dec}_1(sk, c) = \mathsf{Decrypt}'(c, sk)$.

There is then a definition of "δ-correct" saying that the probability of $\mathsf{Dec}(sk, c) \neq m$ given $c \leftarrow \mathsf{Enc}(pk, m)$, maximized over plaintexts and then averaged over keys, is at most δ. We start with a correct PKE, so we take $\delta_1 = 0$.

"$U^{\not\perp}$" is defined in [20, page 5] as follows:

- Start from "an encryption scheme PKE_1 and a hash function H". We take PKE_1 to be specifically the $(\mathsf{Gen}_1, \mathsf{Enc}_1, \mathsf{Dec}_1)$ defined above, and we take H to be an RO mapping to a set SKeys of size at least 3, so the IND-CCA attack stated above works with probability at least 2/3.
- Encapsulation, given pk, computes $c \leftarrow \mathsf{Enc}_1(pk, m)$ and $K \leftarrow H(c, m)$, where "m is picked at random from the message space". Later "$U^{\not\perp}_m$", which we focus on, is defined as replacing "$K = H(c, m)$" with "$K = H(m)$": i.e., it computes ciphertext $c = \mathsf{Enc}_1(pk, m) = (0, \mathsf{Encrypt}(m, pk))$ and session key $H(m)$.
- Decapsulation, given sk and c, returns "$H(c, m)$" if "$m \neq \perp$", where $m = \mathsf{Dec}_1(sk, c)$; it also has a definition when $m = \perp$, which does not matter for us. Again "$U^{\not\perp}_m$" replaces $H(c, m)$ with $H(m)$.

Notice that decapsulation does not reencrypt m, so it behaves the same way for ciphertext $(0, \mathsf{Encrypt}(m, pk))$ and ciphertext $(1, \mathsf{Encrypt}(m, pk))$.

Formally, these definitions of "$U^{\not\perp}$" and "$U^{\not\perp}_m$" are incorrect, since they gloss over the inclusion of a random seed in the private key. This is fixed in the formal definition of "$U^{\not\perp}$" in [20, page 16, Figure 12]. The definitions of "$U^{\perp}$" and "$U^{\perp}_m$" are also formally presented as figures, and reencryption does not appear in any of these three figures. The definition of "$U^{\not\perp}_m$" is not formally presented, but there is nothing to suggest that it should include reencryption. The composition of "T" with "$U^{\not\perp}_m$" is reviewed in [20, page 21, Figure 18]; this composition includes reencryption because "T" does. Beware that implicit rejection for "$U^{\not\perp}_m$" is stated incorrectly in [20, page 21, Figure 18].

[20, Theorem 3.6] makes the following assumptions:

- PKE_1 is δ_1-correct. This is true for us with $\delta_1 = 0$, as noted above, since we start from a correct PKE.
- Various "q" quantities are upper bounds on how frequently various types of queries occur. Except for q_D mentioned below, all of these are multiplied by δ_1, which is 0 in our case, so we skip the details.
- Enc_1 is deterministic. This is true for us, since we start from a deterministic PKE.

- B is an IND-CCA adversary against the resulting KEM. We take the attack described above.
- B issues at most q_D decapsulation queries. We take $q_D = 1$: the attack issues one decapsulation query.

The conclusion is that "there exists an OW-CPA adversary A against PKE_1", with running time "about that of B", where the IND-CCA advantage of B is at most the OW-CPA advantage of A, plus $q_d/|\mathcal{M}| = 1/\#\mathsf{Plaintexts}$, plus $\delta_1 = 0$ times various quantities.

In our case the IND-CCA advantage of B is at least $2/3$, so what the theorem claims is that the OW-CPA advantage of A is at least $1/6$; this is where we use $\#\mathsf{Plaintexts} \geq 2$. This in turn implies an OW-CPA attack against X with advantage at least $1/6$: given a challenge ciphertext for X, simply insert a leading 0 to obtain a challenge ciphertext for PKE_1.

To summarize: The theorem, applied to our example, claims that there is an efficient OW-CPA attack with advantage at least $1/6$ against any correct deterministic PKE with at least 2 plaintexts. Formally, efficiency (e.g., the word "about" in the theorem) is undefined, but there is no reasonable definition for which this claim is plausible, never mind proven.

A.4. Details of [20, Theorem 3.5]. We have focused on "$U_m^{\not\perp}$"; but, as noted above, the attack does not depend on the difference between "$U_m^{\perp}$" and "$U_m^{\not\perp}$", i.e., the difference between explicit rejection and implicit rejection.

Security for "$U_m^{\perp}$" is claimed in [20, Theorem 3.5], which makes a stronger assumption than OW-CPA security for PKE_1. Specifically, it assumes "OW-VA" security, which gives the attacker the same type of challenge as OW-CPA security plus access to a "ciphertext-validity oracle" that reveals whether PKE_1 decrypts a ciphertext (other than the challenge ciphertext) to a valid plaintext.

To produce a counterexample, we tweak decryption in our definition of PKE_1 to be as far as possible from rigidity. Specifically, fix a plaintext z; whenever $\mathsf{Decrypt}$ returns $\perp$, redefine Dec_1 to return z rather than $\perp$. This has no impact on the IND-CCA2 attack, since $\mathsf{Decrypt}$ never returns $\perp$ inside the attack. A ciphertext-validity oracle for PKE_1 is now trivial to construct—all ciphertexts are valid—so an OW-VA attack implies an OW-CPA attack; and an OW-CPA attack for the tweaked system implies an OW-CPA attack for the original system, since changes in decryption are invisible to OW-CPA.

Presumably the proof claimed for [20, Theorem 3.5] somehow uses rigidity. We have not taken the time to check the proof. Reducing the number of KEM constructions will improve auditability, as noted in Section 1, and in particular we recommend focusing on implicit rejection.

A.5. Other transformations. The attack does not apply to "$U^{\not\perp}$", where the session key is a hash of both plaintext and ciphertext: modifying the ciphertext produces a random-looking session key. This might indicate an additional reason, beyond Section 15, to include the ciphertext in the hash, as SimpleKEM does. Perhaps proof techniques for hashed ciphertexts, in combination with proof techniques for plaintext confirmation and proof techniques for implicit rejection, are a path towards stronger QROM results.