Verifiable Decapsulation: Recognizing Faulty Implementations of Post-Quantum KEMs

Lewis Glabush[1], Felix Günther[2], Kathrin Hövelmanns[3], and Douglas Stebila[4]

[1] École Polytechnique Fédérale de Lausanne, Switzerland
`lewis.glabush@epfl.ch`
[2] IBM Research Europe – Zurich, Switzerland
`mail@felixguenther.info`
[3] Eindhoven University of Technology, The Netherlands
`kathrin@hoevelmanns.net`
[4] University of Waterloo, Canada
`dstebila@uwaterloo.ca`

Abstract. Cryptographic schemes often contain verification steps that are essential for security. Yet, faulty implementations missing these steps can easily go unnoticed, as the schemes might still function correctly. A prominent instance of such a verification step is the re-encryption check in the Fujisaki–Okamoto (FO) transform that plays a prominent role in the post-quantum key encapsulation mechanisms (KEMs) considered in NIST's PQC standardization process. In KEMs built from FO, decapsulation performs a re-encryption check that is essential for security, but not for functionality. In other words, it will go unnoticed if this essential step is omitted or wrongly implemented, opening the door for key recovery attacks. Notably, such an implementation flaw was present in HQC's reference implementation and was only noticed after 19 months.

In this work, we develop a modified FO transform that binds re-encryption to functionality, ensuring that a faulty implementation which skips re-encryption will be exposed through basic correctness tests. We do so by adapting the "verifiable verification" methodology of Fischlin and Günther (CCS 2023) to the context of FO-based KEMs. More concretely, by exporting an unpredictable confirmation code from the public key encryption and embedding it into the key derivation function, we can confirm that (most of) the re-encryption step was indeed performed during decapsulation. We formalize this concept, establish modified FO transforms, and prove how unpredictable PKE confirmation codes turn into noticeable correctness errors for faulty implementations. We show how to apply this technique to ML-KEM and HQC, both with negligible overhead, by leveraging the entropy lost through ciphertext compression or truncation. We confirm that our approach works through mathematical proofs, as well as experimental data. Our experiments show that the implementation flaw in HQC's reference implementation indeed makes basic test cases when following our approach.

Keywords: Key encapsulation mechanism, public-key encryption, Fujisaki–Okamoto transformation, NIST, ML-KEM, HQC, post-quantum security, QROM

1 Introduction

A key encapsulation mechanism (KEM) allows two parties to establish a shared secret key using only public communication. Post-quantum KEMs, like NIST's recent standard ML-KEM [21], are at

We thank Bénédikt Tran, Bertram Poettering, Serge Vaudenay, Spencer Wilson, and Vadim Lyubashevsky for insightful discussions. K.H. was supported by an NWO VENI grant (Project No. VI.Veni.222.397). D.S was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grant RGPIN-2022-03187 and NSERC Alliance grant ALLRP 578463-22.

the heart of the current transition to quantum-safe cryptography. The most widespread approach used in constructing post-quantum KEMs is to design a public-key encryption (PKE) scheme and then apply the Fujisaki–Okamoto (FO) transform [8, 9] that turns any weakly secure PKE scheme into an IND-CCA-secure KEM using de-randomization and a re-encryption check.

The FO transform. In a nutshell, the FO transform builds KEM encapsulation by sampling a random message m, PKE-encrypting m into a ciphertext c with randomness r derived from m, and outputting c and a shared secret derived from m. Decapsulation PKE-decrypts c into m', derives randomness r' from m', *re-encrypts* m' using r' into c', and *checks* that $c = c'$ before accepting with the key derived from m'. This is what is called the FO re-encryption check.

Brittle cryptography. When implementing cryptographic schemes and protocols, every detail matters and even small implementation bugs can have devastating effects on security. In that regard, cryptographic code can be particularly brittle, especially when some cryptographic verification steps are accidentally skipped or wrongly executed: from a functional point of view, such bugs can be entirely unnoticeable, yet render an implementation completely insecure. A prime example of such a bug was the "goto fail" bug [24] in Apple's SSL/TLS library, where a misplaced `goto fail;` code line resulted in crucial certificate validation steps being skipped, causing invalid certificates to pass validation.

The FO re-encryption check is prone to similar implementation flaws: skipping or wrongly implementing this check could go unnoticed (as honestly generated ciphertexts decrypt correctly), yet may open the door even for key recovery attacks (when the IND-CCA security guarantee of the FO transform becomes voided). Indeed, such an implementation flaw was present until recently in the reference implementation of HQC [1], one of the Round 4 submissions in NIST's PQC standardization process, and was picked up by downstream users of that code, such as `liboqs` [22, 27]. The decapsulation code would accept malformed ciphertexts due to an unfortunate interaction of flawed calculations during re-encryption and a logical flaw checking its result.[5] Despite common test cases for correctness ("Does the shared secret output by encapsulation equal the shared secret output by decapsulation?"), this flaw was only noticed after 19 months [23].

Making decapsulation recognize implementation faults. In this work, we ask how to better protect implementations of FO-transformed KEMs from accidentally skipping or wrongly implementing the re-encryption check. Put differently, we want to devise a cryptographic approach to surface such implementation flaws in a noticeable manner, offering some assurance that a correct implementation does not miss certain security-crucial steps.

Our approach follows a methodology recently introduced by Fischlin and Günther [7], which in turn is inspired by an idea of Heninger [10], namely to minimize the impact of inevitable human errors in implementations by "tying security to basic functionality." In essence, a security-critical implementation flaw should ideally noticeably affect the scheme's functionality, meaning that it will be noticed early on in basic correctness or interoperability tests.

Fischlin and Günther [7] introduced the notion of *confirmation codes*, output by a cryptographic scheme (in addition to its regular outputs). Confirmation codes are to be designed in a way that they are "unpredictable": a faulty-but-benign implementation of critical verification steps should be

[5] During the re-encryption part of HQC's decapsulation code, the pseudorandom seed for derandomization did not include all the inputs used in the corresponding derandomized encapsulation, which lead to the wrong ciphertext being reconstructed. However, the complex constant-time code to check equality of the reconstructed and original ciphertexts *also* had a flaw that effectively flipped the logical result, meaning the ciphertext would be accepted as valid, and the correct shared secret (based on the decapsulated secret and the original ciphertext) would be output.

unlikely to compute them correctly. Such unpredictable confirmation codes can then be used in an overall protocol to ensure that the protocol will not function correctly (e.g., sender and receiver will be unable to establish a connection) if one side uses a faulty implementation. Note that unpredictability need not meet cryptographically small bounds, such as predictability probability less than 2^{-128}: unpredictability of even 2^{-1} means that a faulty-but-benign implementation will fail interoperability or correctness tests half the time, already flagging concerns about the implementation.

We transfer the idea of confirmation codes to the context of FO-based KEMs. By exporting an unpredictable confirmation code from the underlying PKE scheme and embedding it into the KEM's key derivation function, we can confirm that the re-encryption step was indeed performed during decapsulation. Formally, we augment a PKE encryption algorithm with a confirmation code output and establish modified FO transforms (following the modularization by Hofheinz, Hövelmanns, and Kiltz [12]) which integrate this code into the KEM key derivation. Figure 1 illustrates the fault-recognizing KEM resulting from our confirmation-code-augmented FO transforms. We prove that unpredictable PKE confirmation codes turn into noticeable correctness errors on the KEM level when re-encryption is faultily implemented. We finally show how to apply this technique to ML-KEM and HQC with negligible overhead by leveraging the entropy lost due to rounding or truncating low-order ciphertext bits; the recipient can only recover this entropy if they indeed implement (nearly all of) re-encryption. We experimentally confirm that our approach works: it surfaces the flaw in HQC's reference implementation by making basic test cases fail.

1.1 Contributions and Technical Overview

Definitions and security notions for PKEs and KEMs. We begin by defining the syntax and a security notion for confirmation-code-augmented public key encryption schemes, called *confirmation code unpredictability* (cUP). Compared to [7], we suggest a technically refined approach to capture the idea of a flawed implementation in which not all steps are executed. We model this through giving the adversary access to all inputs to the algorithm (which would of course allow them to trivially recompute everything the algorithm computes) but restricting the adversary's access to some function F used by the algorithm. Looking ahead to our modifications to the FO transform, this limited access translates to the adversary being unable to fully/correctly implement the re-encryption step which depends on F. We define a related notion for KEMs, called *faulty implementation correctness* (fCOR), which models the ability of an adversarial faulty implementation to produce correct shared secrets during decapsulation, again with limited access to some function F used by the algorithm.

Let us emphasize that the core idea here is to catch faulty-*but-benign* implementations. We are not interested in capturing *malicious* implementations, which may of course skip re-encryption but make targeted calls to the restricted function F. The goal of our work, and the general idea of tying security to basic functionality [10] and making cryptographic operations verifiable [7], is to make accidental implementation errors be noticed to assist developers.

Modular FO transforms for confirmation-code-augmented PKEs. We then present a sequence of modular transformations that build a fault-recognizing KEM from a PKE scheme with unpredictable confirmation codes such that the KEM has noticeable correctness errors if the FO re-encryption check is faultily implemented. Following [12] modularizing FO into transforms T and U, we give confirmation-code-augmented versions of these transforms TC and UC and the resulting confirmation-code-augmented FO transform FOC, for both the explicit and implicit rejection case. We show that TC maintains unpredictability and that FOC, like FO, turns a weakly (OW-CPA/IND-CPA) secure PKE scheme into an IND-CCA KEM; we provide these results in both the random oracle and quantum random oracle models. Most importantly, we prove that when using our FOC

```
Encaps(pk)
    $m \leftarrow\!\$\; \mathcal{M}$
    $(c, \mathsf{cd}) \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$
    $K \leftarrow \mathsf{KDF}(m, \mathsf{pk}, \mathsf{cd})$
    return $(K, c)$

Decaps(sk, c)
    $m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
    $(c', \mathsf{cd}') \leftarrow \mathsf{Enc}(\mathsf{pk}, m')$
    check $c' = c$
    $K' \leftarrow \mathsf{KDF}(m', \mathsf{pk}, \mathsf{cd}')$
    return $K'$
```

Fig. 1: Simplified encapsulation and decapsulation algorithms of a fault-recognizing KEM, using our confirmation-code-augmented FO transform on a PKE scheme where encryption (Enc) additionally outputs a confirmation code cd. We highlight differences to a regular FO transform in violet boxes.

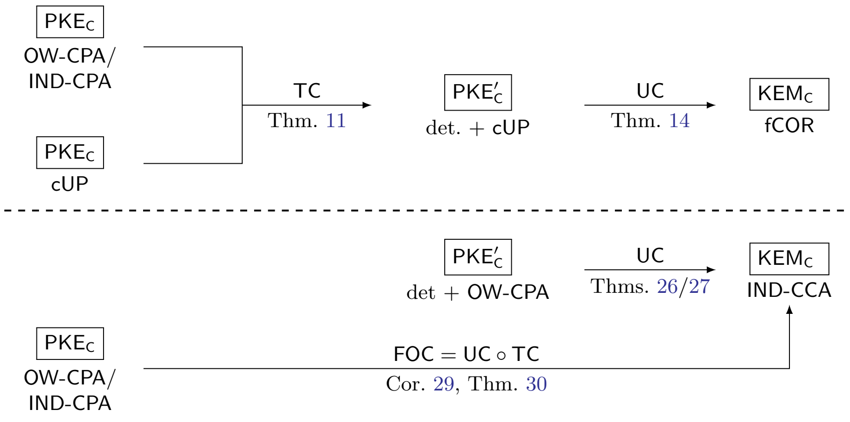


Fig. 2: Summary of our transformations to build a fault-recognizing KEM KEM$_C$ from a confirmation-code-augmented PKE PKE$_C$ using our FOC = UC $\circ$ TC transform. **Top:** Sequence of transformations/theorems to achieve faulty implementation correctness (fCOR) for KEM$_C$ from confirmation-code unpredictability (cUP) and security of PKE$_C$. **Bottom:** Sequence of transformations/theorems to achieve IND-CCA-security for KEM$_C$ from CPA security of PKE$_C$.

transforms on a PKE scheme with unpredictable confirmation codes (cUP), the resulting fault-recognizing KEM (illustrated in Fig. 1) will have noticeable correctness errors when re-encryption is faultily implemented (fCOR). Figure 2 summarizes these transformations and results.

Application to ML-KEM and HQC. We present a minimally modified version of ML-KEM [21] that augments the underlying PKE with confirmation codes based on the uncompressed module-LWE ciphertexts and includes them in the key derivation, following our FOC = UC $\circ$ TC transform. This introduces negligible overhead: using short confirmation codes of only 12–20 bytes ensures that a single correctness test fails with probability at least $\sim 1/3$, which will be highly noticeable in practice, and incurs at most 3.4% overhead in our experimental implementation.

We also present a variant of HQC [1] with confirmation codes. Here we use the last byte of a ciphertext component before truncation as the confirmation code; experimentally this incurs at most 0.25% overhead on runtime. We confirmed that adding a confirmation code to HQC would have caught the bug HQC's reference implementation [23]: with the confirmation code added, basic correctness tests (comparing whether the shared secret output by encapsulation equals the shared secret output by decapsulation) failed immediately in our experiments.

1.2 Related Work

The idea of tying security to basic functionality as a means to hedge against implementation errors was presented by Heninger [10] in a talk at the Workshop on Attacks in Cryptography 2 (WAC2) affiliated with Crypto 2019. Heninger discussed various flaws in deployed cryptographic implementations, suggesting that the "fragility under human error should be a cryptographic design consideration." Inspired by this idea, Fischlin and Günther [7] introduced a methodology for "verifiable verification", supporting the detection of implementation flaws on a cryptographic level. By letting verification algorithms output a confirmation code that can be used in higher-level protocols, they showed how to make faulty verification steps surface on the protocol level as noticeable correctness errors. The focus of [7] is on signature and MAC verification as well as elliptic curve point validation, and their usage in key exchange protocols. Our work applies the methodology to post-quantum KEMs and the FO transform, where the re-encryption step is similarly prone to implementation errors or accidental omission (possibly even intentional "optimization"), as also suggested by Heninger in a later talk at the Real World Post Quantum Workshop 2024 [11] and evidenced by the implementation flaw in HQC's reference implementation [23].

During the NIST post-quantum standardization process, there has been a lot of work on the security analysis of FO-like transformations, with immense progress made stemming from the development of new QROM techniques. FO comes in two flavours with respect to the rejection mode used to reject invalid ciphertexts. (More details on this in Remark 5 in Sect. 2.2.) The FO variants with implicit rejection were proven secure against quantum attackers much earlier [12], with follow-up proofs allowing for successively better parameters [5, 15, 17, 19, 26], using successively more refined QROM techniques. For explicit rejection, there now also exist results if the base encryption scheme is probabilistic. Whereas the first proofs for explicit rejection [6, 18] did not allow for the same parameters as the ones for implicit rejection, two more recent results [14, 16] closed this gap by making use of an emerging new quantum proof technique, the extractable quantum random oracle [6]. If the base PKE scheme is deterministic, currently no security proofs for explicitly rejecting variants are known.

2 Preliminaries

In this section we recall the necessary background from previous literature that we will use throughout the paper: we recall definitions and notions for PKE schemes and KEMs (deferring the standard $\mathsf{OW\text{-}CPA}$, $\mathsf{IND\text{-}CPA}$, and $\mathsf{IND\text{-}CCA}$ security notions to Appendix A), the Fujisaki-Okamoto transformation, and a helper result for the quantum-accessible ROM.

2.1 Public-Key Encryption

A public-key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms and a finite message space $\mathcal{M}$. The key generation algorithm KeyGen outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, with pk defining a randomness space $\mathcal{R} = \mathcal{R}(\mathsf{pk})$. The encryption algorithm Enc, on input pk and a message $m \in \mathcal{M}$, produces an encryption $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ of m under the public key pk. If necessary, we explicitly specify the used randomness of encryption by writing $c = \mathsf{Enc}(\mathsf{pk}, m; r)$, where $r \leftarrow_\$ \mathcal{R}$. The decryption algorithm Dec, on input sk and a ciphertext c, yields either a message $m = \mathsf{Dec}(\mathsf{sk}, c) \in \mathcal{M}$ or a special symbol $\perp \notin \mathcal{M}$ to show that c is not a valid ciphertext.

In the security analysis of FO-based KEMs, it is usually necessary to utilize notions of *correctness*. This is used to analyze the likelihood of a particular class of chosen-ciphertext attacks, where attackers found a failing ciphertext c, meaning c decrypts to a different message than its originating one, and uses this to obtain some leakage on the secret key. We now recall the information-theoretic correctness notion that was given in [12].

```
Game FFP-ATK
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$
    $m \leftarrow \mathcal{A}^{O_{\mathsf{ATK}}, \mathsf{G}'}(\mathsf{pk})$
    $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$
    $m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
    return $[\![m' \neq m]\!]$

ODec($c$)
    $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
    return $m$
```

Fig. 3: Games FFP-ATK for a deterministic PKE, where ATK $\in \{\mathsf{CPA}, \mathsf{CCA}\}$. O_{ATK} is the decryption oracle present in the respective game (see Definition 2) and G' is a random oracle, provided if it is used in the definition of PKE.

Definition 1 (PKE correctness [12]). *We call a public-key encryption scheme* PKE *δ-correct if* $\mathbf{E}\left[\max_{m \in \mathcal{M}} \Pr[\mathsf{Dec}(\mathsf{sk}, c) \neq m | c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)]\right] \leq \delta$*, where the expectation is taken over* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$.

This definition reflects that even a (possibly unbounded) adversary with access to the key pair cannot find failing messages with a probability higher than δ. A computational (game-based) approach that does not hand over the secret key was introduced in [14], via the *Find Failing Plaintext* (FFP) notions below.

Definition 2 (FFP-ATK). *Let* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a deterministic public-key encryption scheme. For* ATK $\in \{\mathsf{CPA}, \mathsf{CCA}\}$*, we define* FFP-ATK *games as in Fig. 3, where*

$$\mathsf{O}_{\mathsf{ATK}} = \begin{cases} - & \textit{if } \mathsf{ATK} = \mathsf{CPA} \\ \mathsf{ODec} & \textit{if } \mathsf{ATK} = \mathsf{CCA} \end{cases} .$$

We define the FFP-ATK *advantage function of an adversary $\mathcal{A}$ against* PKE *as* $\mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{FFP\text{-}ATK}}(\mathcal{A}) = \Pr[\mathsf{FFP\text{-}ATK}_{\mathsf{PKE}}^{\mathcal{A}} \Rightarrow 1]$.

Definition 3 (Injectivity of PKE schemes [5]). *A deterministic PKE scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is ϵ-injective if* $\Pr[\mathsf{Enc}(\mathsf{pk}, m)$ *is not injective* $: (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()] \leq \epsilon$*. We say* PKE *is injective if* $\epsilon = 0$.

In the security analysis of FO-KEMs, it will also be necessary to capture that adversaries might be able to create valid ciphertexts without knowing the respective plaintext. This will involve the following notion about the entropy (or *spreadness*) of the PKE scheme.

Definition 4 (γ-spreadness). *We say that* PKE *is γ-spread iff for all key pairs* $(\mathsf{pk}, \mathsf{sk}) \in \mathrm{supp}(\mathsf{KeyGen})$ *and all messages $m \in \mathcal{M}$ it holds that* $\max_{c \in \mathcal{C}} \Pr[\mathsf{Enc}(\mathsf{pk}, m) = c] \leq 2^{-\gamma}$*, where the probability is taken over the internal randomness* Enc.

2.2 Fujisaki–Okamoto (FO) Transformation

In this section, we recall the definition of the FO transform as the composition of the two following transformations:

- the derandomizing T-*transform* that additionally adds a re-encryption check to the decryption procedure; and
- the PKE-to-KEM U_m-*transforms* that derive session keys from a randomly chosen message m, which they encrypt using PKE. The two variants of U_m vary in their responses to invalid ciphertexts ($\mathsf{U}_m^{\perp}$ returns $\perp$, while $\mathsf{U}_m^{\not\perp}$ returns pseudo-random values).

```
$\mathsf{KeyGen}'$:
01 $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KeyGen}$
02 $\mathsf{sk}' \leftarrow (\mathsf{sk}, \mathsf{pk})$
03 return $(\mathsf{pk}, \mathsf{sk}')$

$\mathsf{Enc}'(\mathsf{pk}, m)$:
04 $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m; \mathsf{G}'(m))$
05 return $c$

$\mathsf{Dec}'(\mathsf{sk}' = (\mathsf{pk}, \mathsf{sk}), c)$:
06 $m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
07 if $m' = \perp$ or $c \neq \mathsf{Enc}'(\mathsf{pk}, m')$
08     return $\perp$
09 else
10     return $m'$
```

Fig. 4: Algorithms of $\mathsf{T}[\mathsf{PKE}, \mathsf{G}']$.

```
$\mathsf{KeyGen}^{\not\perp}$
01 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$
02 $s \leftarrow_{\$} \mathcal{M}$
03 $\mathsf{sk}' \leftarrow (\mathsf{sk}, s)$
04 return $(\mathsf{pk}, \mathsf{sk}')$

$\mathsf{Decaps}_m^{\perp}(\mathsf{sk}, c)$
05 $m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
06 if $m' = \perp$
07     return $\perp$
08 else
09     return $K \leftarrow \mathsf{G}(m')$

$\mathsf{Encaps}_m(\mathsf{pk})$
10 $m \leftarrow_{\$} \mathcal{M}$
11 $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$
12 $K \leftarrow \mathsf{G}(m)$
13 return $(K, c)$

$\mathsf{Decaps}_m^{\not\perp}(\mathsf{sk}', c)$
14 Parse $(\mathsf{sk}, s) \leftarrow \mathsf{sk}'$
15 $m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
16 if $m' = \perp$
17     return $K \leftarrow \mathsf{G}(s, c)$
18 else
19     return $K \leftarrow \mathsf{G}(m')$
```

Fig. 5: 'Explicit rejection' KEM $\mathsf{KEM}_m^{\perp} = (\mathsf{KeyGen}, \mathsf{Encaps}_m, \mathsf{Decaps}_m^{\perp})$, and 'implicit rejection' KEM $\mathsf{KEM}_m^{\not\perp} = (\mathsf{KeyGen}^{\not\perp}, \mathsf{Encaps}_m, \mathsf{Decaps}_m^{\not\perp})$, obtained from PKE scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

The T-transform: To a PKE scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and a hash function $\mathsf{G}' : \mathcal{M} \rightarrow \mathcal{R}$, we associate PKE scheme

$$\mathsf{PKE}' = \mathsf{T}[\mathsf{PKE}, \mathsf{G}'] = (\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}') \ ,$$

with the algorithms defined in Fig. 4.

The U_m-transforms: To a PKE scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and a hash function $\mathsf{G} : \mathcal{M} \rightarrow \mathcal{K}$, we associate key encapsulation mechanism

$$\mathsf{KEM}_m^{\perp} = \mathsf{U}_m^{\perp}[\mathsf{PKE}, \mathsf{G}] = (\mathsf{KeyGen}, \mathsf{Encaps}_m, \mathsf{Decaps}_m^{\perp}) \ ,$$
$$\mathsf{KEM}_m^{\not\perp} = \mathsf{U}_m^{\not\perp}[\mathsf{PKE}, \mathsf{G}] = (\mathsf{KeyGen}^{\not\perp}, \mathsf{Encaps}_m, \mathsf{Decaps}_m^{\not\perp})$$

where all algorithms are defined in Fig. 5.

Remark 5 (Rejection mode). In the literature, $\mathsf{KEM}_m^{\perp}$ is often called 'KEM with explicit rejection' because decapsulation returns the dedicated failure symbol $\perp$ upon decryption failure. In turn, $\mathsf{KEM}_m^{\not\perp}$ is often called 'KEM with implicit rejection'. This variant differs only from $\mathsf{KEM}_m^{\perp}$ in that it reacts to invalid ciphertexts by returning a pseudo-random value instead of the dedicated failure symbol $\perp$ (compare line 17 to line 07).

2.3 QROM helpers

We first recall one of the recent OWtH lemmata, Double-Sided OWtH [5, Lemma 5]. This lemma is used in the security analysis of FO-like KEMs to argue that a hash value (the output session key) is indistinguishable from random.

Lemma 6 (Double-Sided OWtH). *Let $\mathsf{G}, \mathsf{H} : X \to Y$ be random functions, let z be a random value, and let $S \subset X$ be a random set such that $\forall x \in X \setminus S$, $\mathsf{G}(x) = \mathsf{H}(x)$. $(\mathsf{G}, \mathsf{H}, S, z)$ may have arbitrary joint distribution. Let $\mathcal{A}^{\mathsf{H}}$ be a quantum oracle algorithm. Let $f : X \to W \subset \{0,1\}^n$ be any function, and let $f(S)$ denote the image of S under f. Let Ev be an arbitrary classical event. We will define another quantum oracle algorithm $\mathcal{B}^{\mathsf{G},\mathsf{H}}(z)$. This $\mathcal{B}$ runs in about the same amount of time as $\mathcal{A}$, but when $\mathcal{A}$ queries H, $\mathcal{B}$ queries both G and H, and also runs f twice. Let*

$$P_{\text{left}} = Pr[\mathsf{Ev} : \mathcal{A}^{\mathsf{H}}(z)], P_{\text{right}} = Pr[\mathsf{Ev} : A^{\mathsf{G}}(z)], P_{\text{extract}} = Pr[\mathcal{B}^{\mathsf{G},\mathsf{H}}(z) \in f(S)].$$

If $f(S) = \{w^*\}$ is a single element, then $\mathcal{B}$ will only return $\perp$ or w^*, and furthermore, $|P_{\text{left}} - P_{\text{right}}| \leq 2\sqrt{P_{\text{extract}}}$.

3 Confirmation-code-augmented PKE Schemes

In Sect. 4, we will augment variants of FO to tie functionality to security, using confirmation codes. This section provides the necessary formalism: in Sect. 3.1, we introduce the syntax and security notions for PKE schemes with added confirmation codes. To ease the analysis of probabilistic PKE schemes, in Sect. 3.2 we provide a confirmation-code-augmented counterpart of the derandomizing T-transform (which is part of the overall FO transform) and prove the necessary security results.

3.1 Confirmation-code-augmented PKE

We start by introducing the syntax of confirmation-code-augmented PKE (which we formalize as PKE_C), then adapt relevant standard definitions to the augmented setting, and finally formalize the main idea of confirmation codes as confirmation code unpredictability (cUP).

Definition 7 (Confirmation-code-augmented PKE). *A confirmation-code-augmented PKE scheme is a triplet of algorithms $\mathsf{PKE}_\mathsf{C} = (\mathsf{KeyGen}, \mathsf{Enc}_\mathsf{C}, \mathsf{Dec})$, together with message space $\mathcal{M}$, encryption randomness space $\mathcal{R}$ and confirmation code space $\mathcal{CD}$, such that*

- Enc_C *is an algorithm that takes as input a public key* pk, *a message $m \in \mathcal{M}$ and encryption randomness $r \in \mathcal{R}$, and outputs a ciphertext c, together with a confirmation code* $\mathsf{cd} \in \mathcal{CD}$.
- $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a* PKE *scheme, where* Enc *denotes the algorithm which runs* Enc_C *and outputs only c.*

We will later leverage that randomized PKE schemes involve some function F *that we will model as a random oracle. We will make this oracle explicit in games. In such cases, we write* $\mathsf{KeyGen}^\mathsf{F}$, $\mathsf{Enc}_\mathsf{C}^\mathsf{F}$, *and* Enc^F *to indicate the algorithms have (oracle) access to function* F. *(Deterministic algorithm* Dec *will not use* F.*)*

While the definition might suggest that adding code confirmation introduces new computational steps and hence require additional resources, we will be able to instantiate confirmation codes by using intermediate values that were picked along the way while encrypting (see Sect. 5).

Definition 8 (Correctness and basic security for code-augmented PKE). *We say that a confirmation-code-augmented PKE scheme* $\mathsf{PKE}_\mathsf{C} = (\mathsf{KeyGen}, \mathsf{Enc}_\mathsf{C}, \mathsf{Dec})$ *satisfies δ-correctness or a security definition (*$\mathsf{IND\text{-}CPA}$, $\mathsf{OW\text{-}CPA}$, *etc.) if its associated PKE scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *does.*

Confirmation Code Unpredictability (cUP) The idea of confirmation codes is that it should be unlikely that a flawed implementation produces them by accident. Inspired by the "verifiable

```
$\mathrm{Exp}^{\mathsf{cUP}}_{\mathsf{PKE_C},\mathsf{F},\mathsf{max}}(\mathcal{A})$
01 $(\mathsf{pk},\mathsf{sk}) \leftarrow\$ \mathsf{KeyGen}^{\mathsf{F}}()$
02 $(m, r) \leftarrow\$ \mathcal{M} \times \mathcal{R}$
03 $(c, \mathsf{cd}) \leftarrow \mathsf{Enc}^{\mathsf{F}}_{\mathsf{C}}(\mathsf{pk}, m; r)$
04 $\mathsf{cd}' \leftarrow \mathcal{A}^{\bar{\mathsf{F}}}(\mathsf{pk}, \mathsf{sk}, c, m, r)$
05 return $[\![\mathsf{cd} = \mathsf{cd}']\!]$

$\bar{\mathsf{F}}(x)$ //at most max many queries
06 return $\mathsf{F}(x)$
```

Fig. 6: Confirmation code unpredictability experiment cUP for $\mathsf{PKE_C}$ involving function F. The definition for deterministic schemes simply drops the randomness r in lines 03 and 04.

verification" methodology of Fischlin and Günther [7], we formalize this property as *confirmation code UnPredictability* (cUP), in a technically revised form.

Intuitively, our notion reflects the probability that a faulty implementation of FO re-encryption (treated as adversary $\mathcal{A}$) produces the same confirmation code as the original encryption call. Formalizing this intuition turns out to be challenging. A first straw-man proposal for a cUP definition may be to ask the adversary $\mathcal{A}$ to compute the confirmation code that a fresh run of $\mathsf{Enc_C}$ would have produced, when just seeing the public key and the resulting ciphertext. This definition however permits unsuited confirmation codes: the encrypted message itself, or the randomness used to encrypt it, would be deemed good ("unpredictable") confirmation codes. Yet, in the FO re-encryption setting we are interested in, both message and randomness are recovered upon decapsulation, and a flawed implementation can easily use them without running re-encryption. The problem is that, similar to the setting of verifiable verification discussed in [7], the FO re-encryption process—by design—has all the inputs to Enc required to compute the correct confirmation code, in particular the public key, message, and randomness. (Arguably, it even has the secret key, given that re-encryption happens inside the decapsulation algorithm.) If we were to provide those inputs to the cUP adversary $\mathcal{A}$ without restrictions, $\mathcal{A}$ could trivially compute the correct confirmation code, which would thus render the definition moot.

The question is hence how to restrict an cUP adversary in a way that captures accidentally skipped re-encryption in an FO decapsulation implementation, and that accounts for the fact that $\mathcal{A}$ should have access to all encryption inputs by design. Our proposal, formalized in Fig. 6, provides $\mathcal{A}$ with all these inputs (namely, public and secret key, ciphertext, message, and encryption randomness). The restriction is *which computations* $\mathcal{A}$ may perform; this is the technically novel revision of the unpredictability approach of [7].

More specifically, we abstractly limit $\mathcal{A}$'s access to a function F that is involved in the PKE scheme to certain number max of calls, modeling F as a random oracle. If re-encryption and code computation now involve F, the intuition behind limiting access to F is the following: an erroneous implementation that skips re-encryption would not accidentally compute some inner values resulting from calls to F that happen during that re-encryption step.[6]

For a concrete example, we will later see that the PKE encryption step in $\mathsf{ML\text{-}KEM}$ samples error vectors using SHAKE as an extendable output function on the encryption randomness and a counter. Modeling SHAKE as a random oracle, this allows us to use the entropy in the error vectors that is lost due to rounding lower-order bits of the ciphertext as confirmation code. A flawed

[6] Let us stress once more that we are not interested in capturing *malicious* implementations—which may of course skip re-encryption but make targeted calls to F—but in capturing accidental implementation errors.

implementation skipping re-encryption would have to accidentally also call SHAKE on the right inputs and compute the uncompressed ciphertexts to still produce the correct confirmation codes. We argue that this approach indeed captures the intuition of unpredictable confirmation codes.

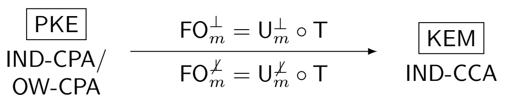
Definition 9 (cUP)**.** *Let* $\mathsf{PKE}_\mathsf{C}^\mathsf{F} = (\mathsf{KeyGen}, \mathsf{Enc}_\mathsf{C}, \mathsf{Dec})$ *be a PKE scheme that is confirmation-code-augmented and involves some function* F*. We define the* cUP *experiment for* PKE_C*, modeling* F *as a random oracle and parameterized by a number* $\mathsf{max} \in \mathbb{N}$ *in Fig. 6 and the* cUP *advantage function of an adversary* $\mathcal{A}$ *against* PKE_C *as*

$$\mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{PKE}_\mathsf{C},\mathsf{F},\mathsf{max}}(\mathcal{A}) = \Pr[\mathrm{Exp}^{\mathsf{cUP}}_{\mathsf{PKE}_\mathsf{C},\mathsf{F},\mathsf{max}}(\mathcal{A}) \Rightarrow 1] \ .$$

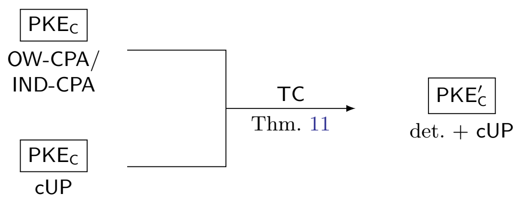
The experiment limits $\mathcal{A}$*'s access to* F *(via oracle* $\bar{\mathsf{F}}$*) to at most* max *queries.*

3.2 cUP Security of the Confirmation-augmented T-Transform

Notably, the FO PKE-to-KEM transformation requires a deterministic PKE scheme. To enable $\mathsf{IND\text{-}CCA}$-secure KEMs based on probabilistic PKE schemes, the underlying PKE scheme first gets derandomized, which is formalized via the T-transform.



To enable our augmentation approach for probabilistic PKE schemes, one would thus have to design a confirmation code for the T-derandomized scheme. Designing a code for the T-derandomized scheme $\mathsf{T}[\mathsf{PKE}]$ and justifying its unpredictability might be an annoying task, given that $\mathsf{T}[\mathsf{PKE}]$ is usually analyzed in the (quantum) ROM. It might be much preferable if one could simply design a code for the underlying probabilistic scheme. To simplify the design process, we thus develop a confirmation-code-augmented counterpart TC to the T-transform that behaves exactly like T, except that it additionally turns the underlying confirmation code into one that is suitable for $\mathsf{T}[\mathsf{PKE}]$. We then show that TC maintains code unpredictability.



TC is called TC because it resembles the derandomization transformation T from [12], except that it derandomizes a code-augmented encryption algorithm Enc_C instead of a plain encryption algorithm Enc, and except that it does not introduce a re-encryption check during decryption.

Definition 10 (Transformation TC**).** *Let* $\mathsf{PKE}_\mathsf{C} = (\mathsf{KeyGen}, \mathsf{Enc}_\mathsf{C}, \mathsf{Dec})$ *be a confirmation-code-augmented PKE scheme with message space* $\mathcal{M}$ *and encryption randomness space* $\mathcal{R}$*, and let* $\mathsf{G}' : \mathcal{M} \to \mathcal{R}$ *be a hash function. To* PKE_C *and a hash function* $\mathsf{G}' : \mathcal{M} \to \mathcal{R}$*, we associate*

$$\mathsf{TC}[\mathsf{PKE}_\mathsf{C}, \mathsf{G}'] = (\mathsf{KeyGen}, \mathsf{Enc}'_\mathsf{C}, \mathsf{Dec}) \ .$$

Algorithm Enc'_C *is given in Fig. 7.*

```
$\mathsf{Enc}'_{\mathsf{C}}(\mathsf{pk}, m)$
07 $(c, \mathsf{cd}) \leftarrow \mathsf{Enc}_{\mathsf{C}}(\mathsf{pk}, m; \mathsf{G}'(m))$
08 return $(c, \mathsf{cd})$
```

Fig. 7: Derandomized code-augmented encryption algorithm $\mathsf{Enc}'_{\mathsf{C}}$ of $\mathsf{TC}[\mathsf{PKE}_{\mathsf{C}}, \mathsf{G}']$

Code unpredictability of TC in the ROM. Intuitively, the following theorem states that TC maintains code unpredictability, provided the underlying scheme is passively secure. We do not consider quantum attackers: code unpredictability is just a stepping stone towards our ultimate goal, noticeable correctness errors in faulty implementations (see next section). The intuition for cUP thus is likewise that $\mathcal{A}$ models a "faulty-but-benign" implementation, and such implementations are unlikely to use a quantum computer. (We note, however, that the result could also be obtained in the quantum-accessible ROM.)

Theorem 11 ($\mathsf{PKE}_{\mathsf{C}}$ cUP **and** PKE OW-CPA/IND-CPA $\overset{\text{ROM}}{\Rightarrow}$ $\mathsf{TC}[\mathsf{PKE}, \mathsf{G}']$ cUP)**.** *Let* $\mathsf{PKE}_{\mathsf{C}} = (\mathsf{KeyGen}, \mathsf{Enc}_{\mathsf{C}}, \mathsf{Dec})$ *be a confirmation-code-augmented PKE scheme, and let* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *denote its associated plain PKE scheme. For any* cUP *adversary* $\mathcal{A}$ *against* $\mathsf{TC}[\mathsf{PKE}_{\mathsf{C}}, \mathsf{G}']$*, where we model* G' *as a random oracle, there exists a* OW-CPA *adversary* $\mathcal{B}$ *against* $\mathsf{T}[\mathsf{PKE}, \mathsf{G}']$ *such that*

$$\mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{TC}[\mathsf{PKE}_{\mathsf{C}},\mathsf{G}']}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{PKE}_{\mathsf{C}}}(\mathcal{A}) + \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{T}[\mathsf{PKE},\mathsf{G}']}(\mathcal{B}) \ ,$$

and according to [12], there exists a OW-CPA *adversary* $\mathcal{C}_O$ *and an* IND-CPA *adversary* $\mathcal{C}_I$ *against* PKE *such that*

$$\mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{T}[\mathsf{PKE},\mathsf{G}']}(\mathcal{B}) \leq \begin{cases} (q_{\mathsf{G}} + 1) \cdot \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{C}_O) \\ 3 \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{C}_I) + \frac{2q_{\mathsf{G}}+1}{|\mathcal{M}|} \end{cases} ,$$

where q_{G} *denotes the number of queries made by* $\mathcal{A}$ *to the random oracle* G'*. The running time of* $\mathcal{B}$*,* $\mathcal{C}_O$ *and* $\mathcal{C}_I$ *is about that of* $\mathcal{A}$*.*

To a cUP adversary $\mathcal{A}$ trying to predict the code belonging to a ciphertext, there is not too much of a difference between attacking $\mathsf{PKE}_{\mathsf{C}}$ or its derandomized version $\mathsf{TC}[\mathsf{PKE}_{\mathsf{C}}, \mathsf{G}']$: in the random oracle model, a challenge taken from $\mathsf{TC}[\mathsf{PKE}_{\mathsf{C}}, \mathsf{G}']$ only differs from a challenge taken from $\mathsf{PKE}_{\mathsf{C}}$ if $\mathcal{A}$ queries G' on the challenge plaintext, thereby breaking OW-CPA security of $\mathsf{T}[\mathsf{PKE}, \mathsf{G}']$. The probability of breaking $\mathsf{T}[\mathsf{PKE}, \mathsf{G}']$ was already analyzed in [12, Thms. 3.1, 3.2], we recall these results in Appendix B for convenience. For completeness, we formally prove Thm. 11 in Appendix C.

4 Confirmation-code-augmented FO-Transform

In this section, we show how to slightly modify FO-like transforms in a way such that implementations can no longer accidentally skip the re-encryption step without triggering functionality issues. We start with formalizing the property we want to achieve, faulty implementation correctness (fCOR), in Sect. 4.1. We then present our modified FO-like transforms for deterministic schemes in Sect. 4.2, and show that they achieve the fCOR property. (We also show that our augmentation approach does not degrade security: the modified transforms still achieve IND-CCA security with security bounds that are similar to their non-modified predecessors.) Lastly, we do the same for *probabilistic* schemes in Sect. 4.3.

```
$\underline{\mathrm{Exp}^{\mathsf{fCOR}}_{\mathsf{KEM},\mathsf{F},\mathsf{max}}(\mathcal{A})}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KEM.KeyGen}^{\mathsf{O},\mathsf{F}}()$
$(c, K_0) \leftarrow_{\$} \mathsf{KEM.Encaps}^{\mathsf{O},\mathsf{F}}(\mathsf{pk})$
$K_0' \leftarrow \mathsf{KEM.Decaps}^{\mathsf{O},\mathsf{F}}(\mathsf{sk}, c)$
if $K_0' \neq K_0$ //correctness error
    return 1 // is a trivial win
$K_1 \leftarrow \mathcal{A}^{\mathsf{O},\bar{\mathsf{F}}}(\mathsf{pk}, \mathsf{sk}, c)$
return $[\![K_0 = K_1]\!]$

$\underline{\bar{\mathsf{F}}(x)}$ //at most max many queries
return $\mathsf{F}(x)$
```

```
$\underline{\mathrm{Exp}^{\mathsf{fCOR}}_{\mathsf{KEM}^{\not\perp}_{\mathsf{C}},\mathsf{F},\mathsf{max}}(\mathcal{A})}$
$(\mathsf{pk}, (\mathsf{sk}, h, z)) \leftarrow_{\$} \mathsf{KEM}^{\not\perp}_{\mathsf{C}}.\mathsf{KeyGen}^{\mathsf{F}}()$

//$\mathsf{KEM}^{\not\perp}_{\mathsf{C}}.\mathsf{Encaps}^{\mathsf{F}}(\mathsf{pk})$
$m \leftarrow_{\$} \mathcal{M}$
$(c, \mathsf{cd}_0) \leftarrow \mathsf{PKE_C}.\mathsf{Enc}^{\mathsf{F}}_{\mathsf{C}}(\mathsf{pk}, m)$
$K_0 \leftarrow \mathsf{G}(m, \mathsf{H}(\mathsf{pk}), \mathsf{cd}_0)$

//$\mathsf{KEM}^{\not\perp}_{\mathsf{C}}.\mathsf{Decaps}^{\mathsf{F}}(\mathsf{sk}, c)$
$m' \leftarrow \mathsf{PKE_C}.\mathsf{Dec}(\mathsf{sk}, c)$
$(c', \mathsf{cd}_0') \leftarrow \mathsf{PKE_C}.\mathsf{Enc}^{\mathsf{F}}_{\mathsf{C}}(\mathsf{pk}, m')$
if $m' = \perp$ or $c' \neq c$
    $K_0' \leftarrow \mathsf{J}(z, c)$
else
    $K_0' \leftarrow \mathsf{G}(m', \mathsf{H}(\mathsf{pk}), \mathsf{cd}_0')$

if $K_0' \neq K_0$ //correctness error
    return 1 // is a trivial win
$K_1 \leftarrow \mathcal{A}^{\bar{\mathsf{F}}}(\mathsf{pk}, \mathsf{sk}, c)$
return $[\![K_0 = K_1]\!]$
```

Fig. 8: Left: Faulty implementation correctness experiment fCOR for a generic KEM KEM using random oracle(s) O and F. Right: Faulty implementation correctness experiment fCOR for a $\mathsf{UC}^{\not\perp}_{m}$-FO-transformed, confirmation-code augmented KEM $\mathsf{KEM}^{\not\perp}_{\mathsf{C}}$. $\mathsf{KEM}^{\not\perp}_{\mathsf{C}}$ uses a deterministic confirmation-code-augmented PKE scheme $\mathsf{PKE_C}$ (using random oracle F) and hash functions G, H, and J. (The encapsulation and decapsulation code of $\mathsf{KEM_C}$ are inlined). The version for a UC_m-FO-transformed, confirmation-code augmented KEM $\mathsf{KEM}^{\perp}_{\mathsf{C}}$ can be derived analogously. The adversary $\mathcal{A}$ has unlimited access to O but is allowed at most max queries to F (via the $\bar{\mathsf{F}}$ oracle).

4.1 fCOR: Formalizing that Faulty Implementations Are Noticed

We start by capturing the notion that an implementation cannot accidentally skip the re-encryption step without triggering functionality issues. To that end, we define generic **F**aulty implementation **COR**rectness (fCOR) for KEMs below, see the left of Fig. 8. In the fCOR game, we model the faulty-but-benign decapsulation implementation as an adversary $\mathcal{A}$, whose goal is to output the correct shared secret matching that of an honest encapsulation (c, K_0) sampled according to KEM.Encaps(pk). The adversary $\mathcal{A}$ is given pk, c, and also sk—clearly, without any restrictions, $\mathcal{A}$ can trivially compute K_0 by computing KEM.Decaps(sk, c). As for unpredictability, we restrict $\mathcal{A}$'s access to a function F used within the KEM algorithms and modeled as a random oracle, to at most max queries. The intuition here is that F is essential for some part of the KEM's decapsulation algorithm, and limiting access to F means that the faulty-but-benign implementation does not fully (or, correctly) implement decapsulation.

Our fCOR notion is generic and applies to any confirmation-code-augmented KEM. Here, we are of course interested in FO-transformed, confirmation-code-augmented KEMs resulting from the $\mathsf{UC}^{\not\perp}_{m}$ and $\mathsf{UC}^{\perp}_{m}$ transforms. On the right-hand side of Fig. 8, we thus inline a $\mathsf{UC}^{\not\perp}_{m}$-FO-transformed KEM $\mathsf{KEM}^{\not\perp}_{\mathsf{C}}$ for illustration (the case of $\mathsf{UC}^{\perp}_{m}$ is analogous). Inspecting the code corresponding to decapsulation, $\mathsf{KEM}^{\not\perp}_{\mathsf{C}}.\mathsf{Decaps}^{\mathsf{F}}(\mathsf{sk}, c)$, shows that limiting $\mathcal{A}$'s access to F translates to $\mathcal{A}$—representing the faulty-but-benign implementation—not being able to fully implement the re-encryption step. Put differently: for a $\mathsf{UC}_m^{\not\perp}$-FO-transformed KEM, our fCOR game models that re-encryption is faultily implemented. (Recall again that we want to catch faulty (but benign) implementations; a malicious implementation might of course implement re-encryption and its code computation correctly and still accept any ciphertext.)

The probability of $\mathcal{A}$ winning the fCOR game measures the chance of a faulty-but-benign implementation passing the most basic correctness (e.g., interoperability) test: testing whether a generated ciphertext encapsulating a shared secret decapsulates to the same secret. In contrast to "classical" security advantage notions in cryptography, we are okay with "high" faulty correctness advantages of $\mathcal{A}$, e.g., $1/2$ or $2/3$: as long as there is no adversary that has "very high" advantage close to 1. Assuming basic correctness and interoperability testing runs at least a few iterations, the faulty implementation will exhibit *noticeable* correctness issues in practice.

Definition 12 (Faulty implementation correctness (fCOR)). *Let* $\mathsf{PKE_C}$ *be a deterministic confirmation-code-augmented PKE scheme involving some function* F*, let* G*,* H*, and* J *be hash functions, and let* $\mathsf{KEM_C}$ *be* $\mathsf{KEM}_\mathsf{C}^{\perp} = \mathsf{UC}_m^{\perp}[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}]$ *or* $\mathsf{KEM}_\mathsf{C}^{\not\perp} = \mathsf{UC}_m^{\not\perp}[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}]$ *from Definition 13.*

We define the fault implementation correctness (fCOR*) experiment for* $\mathsf{KEM_C}$ *using* F *and parameterized by a number* $\mathsf{max} \in \mathbb{N}$ *in Fig. 8 and the* fCOR *advantage function of an adversary* $\mathcal{A}$ *against* $\mathsf{KEM_C}$ *as*

$$\mathrm{Adv}^{\mathsf{fCOR}}_{\mathsf{KEM_C},\mathsf{F},\mathsf{max}}(\mathcal{A}) = \Pr[\mathrm{Exp}^{\mathsf{fCOR}}_{\mathsf{KEM_C},\mathsf{F},\mathsf{max}}(\mathcal{A}) \Rightarrow 1] \ .$$

4.2 $\mathsf{UC}_m^{\not\perp}$ and $\mathsf{UC}_m^{\perp}$: Confirmation-code-augmented FO Constructions for *Deterministic* PKE Schemes

We start by presenting our modified FO-like transform for *deterministic* schemes: we augment the original U_m-transforms for deterministic schemes from [12] by

- including a confirmation code into the key derivation procedure to achieve faulty implementation correctness; and additionally,
- incorporating public-key identifiers, which reflects recent designs that target multi-user security;
- incorporating a re-encryption check, to prevent issues connected to rigidity [4].

We first define our augmented transformations $\mathsf{UC}_m^{\perp}$ (rejecting explicitly) and $\mathsf{UC}_m^{\not\perp}$ (rejecting implicitly).

Definition 13 (Confirmation-augmented U_m-transforms $\mathsf{UC}_m^{\not\perp}$ and $\mathsf{UC}_m^{\perp}$). *To a deterministic confirmation-code-augmented public-key encryption scheme* $\mathsf{PKE_C} = (\mathsf{KeyGen}, \mathsf{Enc_C}, \mathsf{Dec})$ *and hash functions* G*,* H*, and* J*, we associate*

$$\mathsf{KEM}_\mathsf{C}^{\not\perp} = \mathsf{UC}_m^{\not\perp}[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}] = (\mathsf{KeyGen}^{\not\perp}, \mathsf{Encaps}, \mathsf{Decaps}^{\not\perp}) \qquad \textit{and}$$
$$\mathsf{KEM}_\mathsf{C}^{\perp} = \mathsf{UC}_m^{\perp}[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}] = (\mathsf{KeyGen}^{\perp}, \mathsf{Encaps}, \mathsf{Decaps}^{\perp}) \ ,$$

where the respective algorithms are defined in Fig. 9 and Fig. 10.

Motivation of construction. The purpose of including cd in G's input is to prevent an implementation which accidentally skips the re-encryption check from still agreeing with the sender on the shared secret K, assuming that cd was suitably chosen.

We now show that the confirmation-code-augmented U_m-transform indeed achieves our goal: assuming that the confirmation codes are sufficiently unpredictable, we show that if the corresponding transformed KEM $\mathsf{KEM}_\mathsf{C}^{\perp} = \mathsf{UC}_m^{\perp}[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}]$ or $\mathsf{KEM}_\mathsf{C}^{\not\perp} = \mathsf{UC}_m^{\not\perp}[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}]$ is faultily implemented and does not perform all F calls necessary for the re-encryption check, it will have noticeable

```
$\underline{\mathsf{KeyGen}^{\not\perp}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KeyGen}()$
$h \leftarrow \mathsf{H}(\mathsf{pk})$ ; $z \leftarrow\!\!\$\ \mathcal{M}$
$\mathsf{sk}' \leftarrow (\mathsf{sk}, h, z)$
return $(\mathsf{pk}, \mathsf{sk}')$

$\underline{\mathsf{Encaps}_{\mathsf{C}}(\mathsf{pk})}$
$m \leftarrow\!\!\$\ \mathcal{M}$
$(c, \mathsf{cd}) \leftarrow \mathsf{Enc}_{\mathsf{C}}(\mathsf{pk}, m)$
$K \leftarrow \mathsf{G}(m, \mathsf{H}(\mathsf{pk}), \mathsf{cd})$
return $(K, c)$

$\underline{\mathsf{Decaps}^{\not\perp}_{\mathsf{C}}(\mathsf{sk}' = (\mathsf{sk}, h, z), c)}$
$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$(c', \mathsf{cd}') \leftarrow \mathsf{Enc}_{\mathsf{C}}(\mathsf{pk}, m')$
$K' \leftarrow \mathsf{G}(m', h, \mathsf{cd}')$
$\bar{K} \leftarrow \mathsf{J}(z, c)$
if $m' = \perp$ or $c' \neq c$
    return $\bar{K}$
else return $K'$
```

Fig. 9: Algorithms of the *implicitly rejecting* $\mathsf{KEM}^{\not\perp}_{\mathsf{C}} = (\mathsf{KeyGen}^{\not\perp}, \mathsf{Encaps}_{\mathsf{C}}, \mathsf{Decaps}^{\not\perp}_{\mathsf{C}})$, built via the confirmation-augmented counterpart $\mathsf{UC}^{\not\perp}_m$ to the transform used by ML-KEM. We highlight the differences between ML-KEM's transform and $\mathsf{UC}^{\not\perp}_m$ in violet boxes. As discussed in the text, we decided to call our transformation $\mathsf{UC}^{\not\perp}_m$ because it is similar in spirit to transformation $\mathsf{U}^{\not\perp}_m$.

```
$\underline{\mathsf{KeyGen}^{\perp}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KeyGen}()$
$h \leftarrow \mathsf{H}(\mathsf{pk})$
$\mathsf{sk}' \leftarrow (\mathsf{sk}, h)$
return $(\mathsf{pk}, \mathsf{sk}')$

$\underline{\mathsf{Encaps}_{\mathsf{C}}(\mathsf{pk})}$
$m \leftarrow\!\!\$\ \mathcal{M}$
$(c, \mathsf{cd}) \leftarrow \mathsf{Enc}_{\mathsf{C}}(\mathsf{pk}, m)$
$K \leftarrow \mathsf{G}(m, \mathsf{H}(\mathsf{pk}), \mathsf{cd})$
return $(K, c)$

$\underline{\mathsf{Decaps}^{\perp}_{\mathsf{C}}(\mathsf{sk}' = (\mathsf{sk}, h), c)}$
$m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
$(c', \mathsf{cd}') \leftarrow \mathsf{Enc}_{\mathsf{C}}(\mathsf{pk}, m')$
$K' \leftarrow \mathsf{G}(m', h, \mathsf{cd}')$
if $m' = \perp$ or $c' \neq c$
    return $\perp$
else return $K'$
```

Fig. 10: Algorithms of the *explicitly rejecting* $\mathsf{KEM}^{\perp}_{\mathsf{C}} = (\mathsf{KeyGen}^{\perp}, \mathsf{Encaps}_{\mathsf{C}}, \mathsf{Decaps}^{\perp}_{\mathsf{C}})$, built via our confirmation-code-augmented $\mathsf{UC}^{\perp}_m$ transform. Again, as discussed, we call our transformation $\mathsf{UC}^{\perp}_m$ because it is similar in spirit to transformation $\mathsf{U}^{\perp}_m$. We highlight the the code augmentation in violet boxes.

correctness errors. Intuitively, such an implementation will not be able to produce the proper code and thus not manage to produce the right session key, unless it finds a suitable collision in G.

Theorem 14 **($\mathsf{UC}^{\perp}_m$ and $\mathsf{UC}^{\not\perp}_m$: $\mathsf{cUP} \implies \mathsf{fCOR}$).** *A UC_m-transformed KEM from a PKE scheme with confirmation-code unpredictability is noticeably incorrect under faulty implementation, when modeling the key derivation function G as random oracle.*

Concretely, let $\mathsf{PKE}_{\mathsf{C}}$ be an arbitrary confirmation-augmented PKE scheme that is δ-correct. For any adversary $\mathcal{A}$ against the faulty implementation correctness game for $\mathsf{KEM}_{\mathsf{C}}$ being $\mathsf{KEM}^{\perp}_{\mathsf{C}} = \mathsf{UC}^{\perp}_m[\mathsf{PKE}_{\mathsf{C}}, \mathsf{G}, \mathsf{H}, \mathsf{J}]$ or $\mathsf{KEM}^{\not\perp}_{\mathsf{C}} = \mathsf{UC}^{\not\perp}_m[\mathsf{PKE}_{\mathsf{C}}, \mathsf{G}, \mathsf{H}, \mathsf{J}]$ where we model G as an (observable, non-programmable) random oracle with output length ℓ_{G}, function F, and parameter max, we construct an adversary $\mathcal{B}$ against the unpredictability (cUP) of $\mathsf{PKE}_{\mathsf{C}}$, such that

$$\mathrm{Adv}^{\mathsf{fCOR}}_{\mathsf{KEM}_{\mathsf{C}},\mathsf{F},\mathsf{max}}(\mathcal{A}) \leq \delta + \mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{PKE}_{\mathsf{C}},\mathsf{F},\mathsf{max}}(\mathcal{B}) + 2^{-\ell_{\mathsf{G}}} \ .$$

For the short confirmation codes we propose for ML-KEM and HQC (cf. Sect. 5), the cUP advantage is essentially also the fCOR advantage for the resulting KEMs, as the former dominates the bound above.

Proof. The proof applies identically to $\mathsf{KEM}^{\not\perp}_{\mathsf{C}}$ and $\mathsf{KEM}^{\perp}_{\mathsf{C}}$.

Let $(c, \mathsf{cd}_0) = \mathsf{PKE_C}.\mathsf{Enc}_\mathsf{C}^\mathsf{F}(\mathsf{pk}, m)$, $K_0 = \mathsf{G}(m, \mathsf{cd}_0, \mathsf{H}(\mathsf{pk}))$, $m' = \mathsf{PKE_C}.\mathsf{Dec}(\mathsf{sk}, c)$, and $K_0' = \mathsf{J}(z, c)$ or $K_0' = \mathsf{G}(m', \mathsf{cd}_0', \mathsf{H}(\mathsf{pk}))$ denote the values that were computed by the fCOR game (expanded for $\mathsf{KEM}_\mathsf{C}^{\not\perp}$) in lines 01–18 of Fig. 8, and let K_1 be the shared secret that was output by $\mathcal{A}$. For $\mathcal{A}$ to win, we must have that (a) $K_0' \neq K_0$ or (b) $K_0 = \mathsf{G}(m, \mathsf{cd}_0, \mathsf{H}(\mathsf{pk})) = K_1$. We note that (a) constitutes a correctness error, so we can upper-bound the probability of (a) happening by δ. For (b), we distinguish two cases:

- Case 1: $\mathcal{A}$ queries $(m, \mathsf{cd}_0, \mathsf{H}(\mathsf{pk}))$ to the random oracle G. This case induces an adversary $\mathcal{B}$ against $\mathrm{Exp}^{\mathsf{cUP}}_{\mathsf{PKE_C},\mathsf{F},\mathsf{max}}$ that simulates $\bar{\mathsf{F}}$ by relaying to its own oracle, observes the queries $\mathcal{A}$ makes to G, and upon the matching query $(m, \mathsf{cd}_0, \mathsf{H}(\mathsf{pk}))$ outputs cd_0.
- Case 2: $\mathcal{A}$ does not query $(m, \mathsf{cd}_0, \mathsf{H}(\mathsf{pk}))$ to the random oracle G. In that case, $K_0 = \mathsf{G}(m, \mathsf{cd}_0, \mathsf{H}(\mathsf{pk}))$ to $\mathcal{A}$ is an unknown random value of length ℓ_G. So $\mathcal{A}$'s chance of guessing it with its output K_1 is $1/2^{\ell_\mathsf{G}}$. $\square$

Confirmation-code-augmentation does not degrade security. We conclude this subsection by discussing that our augmentation approach does not degrade security: we show that the augmented transform $\mathsf{UC}_m^{\not\perp}$ still achieves $\mathsf{IND\text{-}CCA}$ security. We analyze transformation $\mathsf{UC}_m^{\not\perp}$ in 'standalone' mode, i.e., when $\mathsf{UC}_m^{\not\perp}$ is applied to a generic *deterministic* PKE scheme that is confirmation-augmented with *deterministic* code generation. We also discuss the security of other variants of these transformations (Remark 15).

Intuition why $\mathsf{UC}_m^{\not\perp}$ is secure when applied to deterministic schemes. Our transform resembles transformation $\mathsf{U}_m^{\not\perp}$ that was first analyzed in [12, Thm. 3.6]. The only differences between $\mathsf{UC}_m^{\not\perp}$ and $\mathsf{U}_m^{\not\perp}$ are that when deriving session keys, $\mathsf{UC}_m^{\not\perp}$ takes two additional inputs to G – $\mathsf{H}(\mathsf{pk})$ and cd – and that $\mathsf{UC}_m^{\not\perp}$ performs a re-encryption check. The re-encryption check prevents certain CCA attacks (and thus only makes the transform stronger). The additional input $\mathsf{H}(\mathsf{pk})$ is a value that everybody can compute, it thus does not weaken the resulting session key. While the additional input cd does depend on its originating message m, this dependence is deterministic by our requirement on $\mathsf{Enc_C}$. It thus creates no additional information/attack surfaces when being hashed together with m.

We make this intuition formal in Appendix D (for the ROM, see Thm. 26 in Appendix D.1, for the QROM, see Thm. 27 in Appendix D.2). We defer to the appendix because these proofs essential redo previous proofs for non-augmented U-transformations, in the ROM [13, Thm. 2.1.7] and the QROM [5, Thm. 2].

$$\begin{tikzcd} \boxed{\mathsf{PKE}'} \arrow[r, "\mathsf{UC}_m^{\not\perp}", "\text{ROM Thm. 26/QROM Thm. 27}"'] & \boxed{\mathsf{KEM_C}} \\ \text{det} + \mathsf{OW\text{-}CPA} & \mathsf{IND\text{-}CCA} \end{tikzcd}$$

Remark 15 ($\mathsf{IND\text{-}CCA}$ *security for other code-augmented* U*-variants).* One can obtain a proof for the explicitly rejecting variant $\mathsf{UC}_m^{\perp}$ that recovers the ROM bound [12, Thm. 3.5] for $\mathsf{U}_m^{\perp}$, using the same reasoning as in the ROM proof for $\mathsf{UC}_m^{\not\perp}$. The QROM situation of variants with explicit rejection mode so far is undetermined; we are not aware of any QROM result for standalone U-transforms in explicit mode (without key confirmation) at the time of writing.

We also remark that the results for $\mathsf{UC}_m^{\not\perp}$ can be recovered in a straightforward manner for variants of $\mathsf{UC}_m^{\not\perp}$ that

- omit $\mathsf{H}(\mathsf{pk})$ from the hash input since the hashing of pk has absolutely no influence on (single-instance) $\mathsf{IND\text{-}CCA}$ security;
- include the ciphertext c into the hash input [5, Thm. 5].

4.3 $\mathsf{FOC}_m^{\not\perp}/\mathsf{FOC}_m^{\perp}$: Confirmation-code-augmented FO Constructions for *Probabilistic* PKE Schemes

We now present our transformations for *probabilistic* schemes.

Definition 16 (Confirmation-augmented transforms $\mathsf{FOC}_m^{\not\perp}/\mathsf{FOC}_m^{\perp}$). *Transformations $\mathsf{FOC}_m^{\not\perp}/\mathsf{FOC}_m^{\perp}$ are the respective transformation $\mathsf{UC}_m^{\not\perp}/\mathsf{UC}_m^{\perp}$, used in conjunction with TC. In other words, the construction is exactly like the construction of $\mathsf{UC}_m^{\not\perp}/\mathsf{UC}_m^{\perp}$ (see Fig. 10), except that it derandomizes encryption (in lines 06 and 10). Encapsulation and Decapsulation do not sample the randomness for encryption at random, but instead derive it as $\mathsf{G}'(m)$ resp. $\mathsf{G}'(m')$.*

Next, we show that the confirmation-code-augmented FO-transforms indeed achieve our goal, fCOR. This result is a corollary, as it immediately follows from combining the fCOR result for the 'standalone' transformation $\mathsf{UC}_m^{\not\perp}/\mathsf{UC}_m^{\perp}$ (Thm. 14) with the result that TC maintains code unpredictability (Thm. 11).

Corollary 17. *A $\mathsf{FOC}_m^{\not\perp}$-transformed KEM from a probabilistic PKE scheme with confirmation-code unpredictability is noticeably incorrect under faulty implementation, when modeling the key derivation function G as random oracle.*

Concretely, let PKE_C be an arbitrary confirmation-augmented PKE scheme that is δ-correct. For any adversary $\mathcal{A}$ against the faulty implementation correctness game for KEM_C being $\mathsf{KEM}_\mathsf{C}^{\not\perp} = \mathsf{FOC}_m^{\not\perp}[\mathsf{PKE}_\mathsf{C}, \mathsf{G}, \mathsf{H}, \mathsf{J}, \mathsf{G}']$ or $\mathsf{KEM}_\mathsf{C}^{\perp} = \mathsf{FOC}_m^{\perp}[\mathsf{PKE}_\mathsf{C}, \mathsf{G}, \mathsf{H}, \mathsf{J}, \mathsf{G}']$, where we model G as an (observable, non-programmable) random oracle with output length ℓ_G, function F, and parameter max, we construct an adversary $\mathcal{B}$ against the unpredictability (cUP) of PKE_C, such that

$$\mathrm{Adv}^{\mathsf{fCOR}}_{\mathsf{KEM}_\mathsf{C},\mathsf{F},\mathsf{max}}(\mathcal{A}) \leq \delta + \mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{PKE}_\mathsf{C}}(\mathcal{A}) + \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{TC}[\mathsf{PKE}_\mathsf{C},\mathsf{G}']}(\mathcal{B}) + 2^{-\ell_\mathsf{G}} \ ,$$

and according to [12], there exists a $\mathsf{OW\text{-}CPA}$ adversary $\mathcal{C}_O$ and an $\mathsf{IND\text{-}CPA}$ adversary $\mathcal{C}_I$ against PKE such that

$$\mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{TC}[\mathsf{PKE}_\mathsf{C},\mathsf{G}']}(\mathcal{B}) \leq \begin{cases} (q_\mathsf{G}+1)\cdot \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{C}_O) \\ 3\cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{C}_I) + \frac{2q_\mathsf{G}+1}{|\mathcal{M}|} \end{cases} ,$$

where q_G denotes the number of queries made by $\mathcal{A}$ to the random oracle modeling G'. The running time of $\mathcal{B}$, $\mathcal{C}_O$ and $\mathcal{C}_I$ is about that of $\mathcal{A}$.

Confirmation-code-augmentation does not degrade security. We again conclude this subsection by discussing that our augmentation approach does not degrade security: we show that the augmented transform $\mathsf{FOC}_m^{\not\perp}$ still achieves $\mathsf{IND\text{-}CCA}$ security. We also discuss the security of other variants of these transformations (Remark 18).

Intuition why $\mathsf{FOC}_m^{\not\perp}$ is secure. Our transform resembles the $\mathsf{FO}_m^{\not\perp}$ transform that was first analyzed in [12]. The only differences are that when deriving session keys, $\mathsf{FOC}_m^{\not\perp}$ takes two additional inputs to G – $\mathsf{H}(\mathsf{pk})$ and cd. Again, the additional input $\mathsf{H}(\mathsf{pk})$ is a value that everybody can compute and thus does not weaken the resulting session key. While the additional input cd does depend on its originating message m, this dependence is deterministic since $\mathsf{FOC}_m^{\not\perp}$ derandomizes Enc_C. It thus creates no additional information/attack surfaces when being hashed together with m.

We make this intuition formal in Appendix D (for the ROM, see Cor. 29 in Appendix D.3, for the QROM, see Thm. 30 in Appendix D.4). We again defer to the appendix because Cor. 29 is obtained by combining the $\mathsf{UC}_m^{\not\perp}$ result, Thm. 26, with previous results about T that were given in [12], and QROM Thm. 30 essential redoes a previous proof for the non-augmented FO_m-transformations [14].

$$\begin{tikzcd} \begin{array}{c}\boxed{\mathsf{PKE}}\\ \text{OW-CPA/}\\ \text{IND-CPA}\end{array} \arrow[r, "\mathsf{FOC}^{\not\perp}_m = \mathsf{UC}^{\not\perp}_m \circ \mathsf{TC}", "\text{ROM Cor. 29, QROM Thm. 30}"'] & \begin{array}{c}\boxed{\mathsf{KEM_C}}\\ \text{IND-CCA}\end{array} \end{tikzcd}$$

*Remark 18 (*IND-CCA *security for other code-augmented* FO*-variants).* In the previous subsection, we also discussed (see Remark 15) the security of other variants of our construction for deterministic schemes, UC. Unlike for $\mathsf{UC}^{\perp}_m$, however, transformation $\mathsf{FO}^{\perp}_m$ is known to have a QROM security proof [14] that achieves the same bound as $\mathsf{FO}^{\not\perp}_m$. They essentially have the same proof, one can thus also obtain a QROM bound for $\mathsf{FOC}^{\perp}_m$.

The remarks about alternative hash inputs also apply for our construction for probabilistic schemes, FOC.

5 Application to Post-quantum KEMs

We now turn to augmenting real-world post-quantum KEM schemes with confirmation codes that are unpredictable and enable catching faulty implementations of decapsulation through noticeable correctness errors. We discuss and analyze proposals for both ML-KEM, standardized by NIST in FIPS 203 [21], as well as HQC [1], a Round 4 submission to NIST's PQC standardization process.

5.1 ML-KEM

Recall that ML-KEM operates on elements of $R_q = \mathbb{Z}_{3329}[X]/(X^{256}+1)$. In the PKE scheme underlying ML-KEM, the encryption process pseudorandomly samples error terms $\mathbf{e}_1 \in R_q^k$ and $e_2 \in R_q$ (for $k \in \{2,3,4\}$ depending on the security level). These error terms are used to compute a vector $\mathbf{u} \in R_q^k$ of polynomials and a polynomial $v \in R_q$. The polynomials in $\mathbf{u}$ and v are then compressed into the final ciphertext (c_1, c_2). This compression entails a loss of information by rounding coefficients modulo $q = 3329$ to a smaller range, namely modulo 2^{10} or 2^{11} (for $\mathbf{u}$) or 2^4 or 2^5 (for v) (depending on the security level). We leverage this information loss for our confirmation-code-augmented version of ML-KEM, denoted $\mathsf{ML\text{-}KEM_C}$. In doing so, we will model the PRF used to derive the error terms as the constrained function F in the cUP security notion.

Adding confirmation codes to ML-KEM's PKE. First, we specify how to augment the PKE encryption algorithm of ML-KEM with a confirmation code; the resulting encryption algorithm $\mathsf{ML\text{-}KEM_C.PKE_C.Enc_C}$ is given in Fig. 11 (the other PKE algorithms remain unchanged). More specifically, parameterized by $S \subseteq \{1, \ldots, 256\}$, $\mathsf{ML\text{-}KEM_C.PKE_C}[S]\mathsf{.Enc_C}$ outputs as confirmation code cd the coefficients at positions in S of each term of the uncompressed ciphertext values $\mathbf{u} \in R_q^k$ and $v \in R_q$ (see Fig. 11, line 24).

Using a small set S helps reduce the overhead of including cd in the KEM key derivation. We show in the following theorem that a very small set S—just a few coefficients from each uncompressed ciphertext term—already suffices for $\mathsf{ML\text{-}KEM_C.PKE_C}[S]$ to achieve meaningful confirmation-code unpredictability (cUP). Recall that we do not need cryptographically small advantages for confirmation-code unpredictability: a cUP advantage bound noticeably below 1 already suffices to ensure that a KEM built via the confirmation-code-augmented FO-transforms in Sect. 4 has noticeable correctness errors upon faulty implementation. For $\mathsf{ML\text{-}KEM_C.PKE_C}[S]$, we already obtain a cUP advantage upper bound of $\approx 2/3$ for $|S| = 2$ and $\mathsf{max} = k$ (i.e., missing only one of the $k+1$ error polynomials in $\mathbf{e}_1 \in R_q^k, e_2 \in R_q$). Note that we allow parameterizing $\mathsf{ML\text{-}KEM_C.PKE_C}[S]$ with a set of coefficient positions for generality; in practice, slicing off a consecutive sequence of coefficients at the start or end of each polynomial is sufficiently efficient, as our performance evaluation in Sect. 5.3 shows.

```
$\mathsf{ML\text{-}KEM_C.PKE_C}[S].\mathsf{Enc_C}(pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}, m \in \mathcal{B}^{32}; r \in \mathcal{B}^{32})$
01 $N \leftarrow 0$
02 parse $\hat{\mathbf{t}}$ from $pk$
03 parse $\rho$ from $pk$
04 $\hat{\mathbf{A}} \leftarrow$ (pseudorandomly generate matrix $\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k \times k}$ from $\rho$)
09 $\mathbf{y} \leftarrow$ (pseudorandomly generate $\mathbf{y} \in (\mathbb{Z}_q^{256})^k$ from $r$; set $N \leftarrow k$)
13 for $i$ from 0 to $k-1$:                                   //generate $\mathbf{e}_1 \in \left(\mathbb{Z}_q^{256}\right)^k$
14     $\mathbf{e}_1[i] \leftarrow \mathsf{SamplePolyCBD}_{\eta_2}(\mathsf{PRF}_{\eta_2}(r, N))$
15     $N \leftarrow N + 1$
16 endfor
17 $e_2 \leftarrow \mathsf{SamplePolyCBD}_{\eta_2}(\mathsf{PRF}_{\eta_2}(r, N))$              //generate $e_2 \in \mathbb{Z}_q^{256}$
18 $\hat{\mathbf{y}} \leftarrow \mathsf{NTT}(\mathbf{y})$
19 $\mathbf{u} \leftarrow \mathsf{NTT}^{-1}\left(\hat{\mathbf{A}}^T \circ \hat{\mathbf{y}}\right) + \mathbf{e}_1$
20 $\mu \leftarrow \mathsf{Decompress}_1(\mathsf{ByteDecode}_1(m))$
21 $v \leftarrow \mathsf{NTT}^{-1}\left(\hat{\mathbf{t}}^T \circ \hat{\mathbf{y}}\right) + e_2 + \mu$
22 $c_1 \leftarrow \mathsf{ByteEncode}_{d_u}(\mathsf{Compress}_{d_u}(\mathbf{u}))$
23 $c_2 \leftarrow \mathsf{ByteEncode}_{d_v}(\mathsf{Compress}_{d_v}(v))$
   //confirmation code:
   //uncompressed ciphertext coefficients at positions $S \subseteq \{1, \ldots, 256\}$ of each term
24 $\mathsf{cd} \leftarrow (\mathbf{u}[1][S], \ldots, \mathbf{u}[k][S], v[S])$
25 return $\left(c = c_1 \| c_2, \mathsf{cd}\right)$
```

Fig. 11: Confirmation-code-augmented version $\mathsf{ML\text{-}KEM_C.PKE_C}[S]$ of ML-KEM's PKE encryption algorithm, with the length of the confirmation code parameterized by $S \subseteq \{1, \ldots, 256\}$. Changes introducing confirmation codes are higlighted in violet boxes. In our analysis, we model PRF as (extendable-output) random oracle F, where PRF_η denotes a call producing a $(64 \cdot \eta)$-byte output; for all parameter sets, $\eta_2 = 2$. The line numbering follows the ML-KEM specification in NIST FIPS 203 [21, Algorithm 14, p. 30]. Some details not relevant to this paper are omitted; see the FIPS 203 specification for details.

Using confirmation codes in ML-KEM's encapsulation/decapsulation. We now explain how to integrate the confirmation-code-augmented PKE scheme into the encapsulation and decapsulation algorithms of ML-KEM to obtain our confirmation-code-augmented version $\mathsf{ML\text{-}KEM_C}$.

As shown in Fig. 12, ML-KEM uses a variant of the $\mathsf{FO}_m^{\not\perp} = \mathsf{U}_m^{\not\perp} \circ \mathsf{T}$ transform, deriving randomness and shared secret jointly as $(K, r) \leftarrow \mathsf{SHA3\text{-}512}(m \| \mathsf{SHA3\text{-}256}(pk))$ before re-encryption. Applying our confirmation-code-augmented transform $\mathsf{FOC}_m^{\not\perp} = \mathsf{UC}_m^{\not\perp} \circ \mathsf{TC}$ from Sect. 4, we separate the derivation of randomness and shared secret, to include the confirmation code in the latter (obtained through re-encryption). We thus replace the single SHA3-512 call with two calls of SHA3-256; see Fig. 12 for the result. Note that it is important to ensure domain separation between these calls of SHA3-256 and other SHA3-256 calls in ML-KEM, such as the hashing of pk, for example by using labels or ensuring inputs are different length [3]; this requires carefully checking the size of the confirmation code relative to the pk size. When the confirmation code is chosen to be shorter than 736 bytes (and non-empty), domain separation is achieved due to distinct input lengths for each call, namely 800–1568 bytes for the call hashing pk, 64 bytes for the call deriving r, and 65–799 bytes for the call deriving K.

```
ML-KEM.Encaps_internal($pk, m \in \mathcal{B}^{32}$)
$(K, r) \leftarrow$ SHA3-512($m \| $SHA3-256($pk$))
$c \leftarrow$ ML-KEM.PKE.Enc($pk, m; r$)
return $(K, c)$
```

```
ML-KEM$_\mathsf{C}$.Encaps_internal($pk, m \in \mathcal{B}^{32}$)
$r \leftarrow$ SHA3-256 ($m \| $SHA3-256($pk$))
$(c$, cd $) \leftarrow$ ML-KEM$_\mathsf{C}$.PKE$_\mathsf{C}$.Enc$_\mathsf{C}$ ($pk, m; r$)
$K \leftarrow$ SHA3-256($m \| $SHA3-256($pk$)$\|$cd)
return $(K, c)$
```

```
ML-KEM.Decaps_internal($sk' = (sk, h, z), c$)
$m' \leftarrow$ ML-KEM.PKE.Dec($sk, c$)
$(K', r') \leftarrow$ SHA3-512($m' \| h$)
$\bar{K} \leftarrow$ SHAKE256($z \| c$)
$c' \leftarrow$ ML-KEM.PKE.Enc($pk, m'; r'$)
if $c \neq c'$
    return $\bar{K}$
return $K'$
```

```
ML-KEM$_\mathsf{C}$.Decaps_internal($sk' = (sk, h, z), c$)
$m' \leftarrow$ ML-KEM.PKE.Dec($sk, c$)
$r' \leftarrow$ SHA3-256 ($m' \| h$)
$(c'$, cd$'$ $) \leftarrow$ ML-KEM$_\mathsf{C}$.PKE$_\mathsf{C}$.Enc$_\mathsf{C}$ ($pk, m'; r'$)
$K' \leftarrow$ SHA3-256($m' \| h \|$cd$'$)
$\bar{K} \leftarrow$ SHAKE256($z \| c$)
if $c \neq c'$
    return $\bar{K}$
return $K'$
```

Fig. 12: Internal encapsulation (top) and decapsulation (bottom) algorithms of ML-KEM [21, Algorithm 17, p. 33] (left) and of our confirmation-code-augmented version ML-KEM$_\mathsf{C}$ following the $\mathsf{FOC}_m^{\not\perp} = \mathsf{UC}_m^{\not\perp} \circ \mathsf{TC}$ transform (right).We highlight the differences between ML-KEM and ML-KEM$_\mathsf{C}$ in violet boxes .

Security evaluation. The following theorem shows that ML-KEM$_\mathsf{C}$.PKE$_\mathsf{C}[S]$ achieves confirmation-code unpredictability (cUP), when ML-KEM's PRF is modeled as a random oracle to which calls by the faulty implementation are constrained.

Theorem 19 (ML-KEM$_\mathsf{C}$.PKE$_\mathsf{C}$ is cUP). *Let* ML-KEM$_\mathsf{C}$.PKE$_\mathsf{C}[S]$ *be the confirmation-code-augmented PKE scheme of* ML-KEM$_\mathsf{C}$ *from Fig. 11 involving function* $\mathsf{F} = \mathsf{PRF}$*, using uncompressed ciphertext coefficients from the positions* $S \subseteq \{1, \ldots, 256\}$ *as confirmation code. Let* $\mathcal{A}$ *be an adversary in the* cUP *experiment making at most* max *queries to its* $\bar{\mathsf{F}}$ *oracle. If we make an additional uniformity assumption on certain values in* ML-KEM *(stated in the proof), then*

$$\mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{ML\text{-}KEM_C.PKE_C}[S],\mathsf{PRF},\mathsf{max}}(\mathcal{A}) \leq 2^{-h_{e_1} \cdot |S| \cdot (k+1-\mathsf{max})},$$

where h_{e_1} *is as shown in Table 1. Without the additional uniformity assumption, for* ML-KEM-512 *and* ML-KEM-768*, the advantage of* $\mathcal{A}$ *is upper-bounded by*

$$p_{|S|} = \sum_{i=0}^{|S|(k+1-\mathsf{max})} \binom{|S|(k+1-\mathsf{max})}{i} \left(\frac{2}{16}\right)^{|S|(k+1-\mathsf{max})-i} \left(\frac{14}{16}\right)^{i} \cdot 2^{-h \cdot i}, \quad (1)$$

where $h \approx 0.322$*. Calculated values for the bound in each case for short* ($|S| = 2$) *and full* ($|S| = n = 256$) *confirmation codes are shown in Table 1.*

Proof. We observe that PRF is used $k+1$ times in ML-KEM$_\mathsf{C}$.PKE$_\mathsf{C}[S]$.Enc$_\mathsf{C}$, to compute $\mathbf{e}_1[1], \ldots, \mathbf{e}[k]$ as well as e_2.[7] This means for any $\mathsf{max} \leq k$, $\mathcal{A}$ is missing at least one of the $k+1$ error terms.

[7] Strictly speaking, PRF is also used to generate $\mathbf{y}$. But since in the subsequent analysis we will conservatively assume $\mathcal{A}$ knows $\mathbf{y}$, we can also assume w.l.o.g. that $\mathcal{A}$ does not spend its limited $\bar{\mathsf{F}}$ queries on computing $\mathbf{y}$.

Table 1: Calculated error coefficient entropy and confirmation code guessing probabilities for $\mathsf{ML\text{-}KEM}$. Conservatively assumes the adversary is allowed $\mathsf{max} = k$ (i.e., all but one) evaluations of PRF. For short ($|S| = 2$) codes, the confirmation code guessing probability is essentially also the fCOR advantage upper bound, as the other terms in the bound of Thm. 14 are negligible in comparison.

	Conditional entropy		**Conf. code guessing probability**	
Scheme	**of $\mathbf{e}_1$ coeff.**	**of e_2 coeff.**	**Short** ($\|S\| = 2$)	**Full** ($\|S\| = 256$)
	(h_{e_1})	(h_{e_2})	(p_2)	(p_{256})
		Assuming uniform z		
ML-KEM-512	1.2733	2.0188	0.1712	2^{-325}
ML-KEM-768	1.2733	2.0188	0.1712	2^{-325}
ML-KEM-1024	0.6407	2.0069	0.4114	2^{-164}
		Worst-case analysis		
ML-KEM-512	0.3219	0.3219	0.6806	2^{-71}
ML-KEM-768	0.3219	0.3219	0.6806	2^{-71}

Let $\mathbf{u} = \mathsf{NTT}^{-1}\left(\hat{\mathbf{A}}^T \circ \hat{\mathbf{y}}\right) + \mathbf{e}_1$ and $v = \mathsf{NTT}^{-1}\left(\hat{\mathbf{t}}^T \circ \hat{\mathbf{y}}\right) + e_2 + \mu$ be the uncompressed ciphertexts (as computed during challenge encryption) and $c = c_1 \| c_2$ the challenge ciphertext. Since $\mathcal{A}$ knows the secret key sk, we conservatively assume it can compute $\mathsf{NTT}^{-1}\left(\hat{\mathbf{A}}^T \circ \hat{\mathbf{y}}\right)$, $\mathsf{NTT}^{-1}\left(\hat{\mathbf{t}}^T \circ \hat{\mathbf{y}}\right)$, and μ. Using its $\bar{\mathsf{F}}$ oracle, $\mathcal{A}$ can furthermore compute $\mathsf{max} \leq k$ many of the $k+1$ error vectors $\mathbf{e}_1[1], \ldots, \mathbf{e}[k], e_2$. That is, $\mathcal{A}$ knows the compressed ciphertext as well as all components of the uncompressed ciphertext except $k + 1 - \mathsf{max}$ error vectors. For each of the $k + 1 - \mathsf{max}$ missing error vectors, $\mathcal{A}$ needs $|S|$ coefficients to construct the complete confirmation code.

Each coefficient of either $\mathbf{u}$ or v is of the form $z + e$, where z is some element $\mathbb{Z}_q$ and e is sampled according to the $\mathsf{ML\text{-}KEM}$ error distribution, which is the following centered binomial distribution: 0 with probability $6/16$, ± 1 each with probability $4/16$, ± 2 each with probability $1/16$.

The core of the problem is to compute the conditional entropy of a single error coefficient e given $z \in \mathbb{Z}_q$ and the compressed value $c = \mathsf{Compress}_d(z + e \mod q)$, where $\mathsf{Compress}_d(x) = \lceil (2^d/3329) \cdot x \rfloor \bmod 2^d$, with $d = d_u = 10$ for compressing $\mathbf{u}$ and $d = d_v = 4$ for compressing v in both $\mathsf{ML\text{-}KEM\text{-}512}$ and $\mathsf{ML\text{-}KEM\text{-}768}$, and $d_u = 11$ and $d_v = 5$ correspondingly for $\mathsf{ML\text{-}KEM\text{-}1024}$.

Since z takes on values in $\{0, \ldots, 3328\}$ and e takes on values in $\{0, \pm 1, \pm 2\}$, the space of all possibilities can be efficiently explored numerically. We report bounds for two different analyses: an average-case analysis assuming uniform z, and a worst-case analysis not assuming uniform z.

Assuming uniform z. If we were willing to make the assumption that the distribution of z is uniform on $\mathbb{Z}_{3329}$, then it would suffice to compute the entropy of entries of $\mathbf{e}_1$ or e_2 given $(z, c = \mathsf{Compress}_d(z + e \bmod q))$, namely:

$$H(Y|X) = - \sum_{(z,c) \in X, e \in Y} \Pr[X = (z,c), Y = e] \log_2 \frac{\Pr[X = (z,c), Y = e]}{\Pr[X = (z,c)]}.$$

As observed in Table 1, for all $\mathsf{ML\text{-}KEM}$ parameters, the conditional entropy h_{e_1} in a coefficient of $\mathbf{e}_1$ is less than or equal to the conditional entropy h_{e_2} in a coefficient of e_2, so the optimal adversary would use its limited $\bar{\mathsf{F}}$ queries to learn e_2 first, and then any more of $\mathbf{e}_1$ it can. Since a confirmation code using subset S involves $|S|$ coefficients from each error polynomial, of which there are k in $\mathbf{e}_1$ and 1 more in e_2, and these are all independent of each other, the adversary's success probability is at most

$$p_{|S|} = \left(2^{-h_{e_1}}\right)^{|S|\cdot(k+1-\mathsf{max})} = 2^{-h_{e_1}\cdot|S|\cdot(k+1-\mathsf{max})}.$$

The results of calculating this numerically for ML-KEM are shown in the "Assuming uniform z" section of Table 1, for the conservative case that the adversary is allowed $\mathsf{max} = k$ (i.e., all but one) queries to $\bar{\mathsf{F}}$.

Worst-case analysis, not assuming uniform z. Since the cUP security notion assumes that the secret key is known to the adversary, we cannot employ the decisional LWE assumption to justify z being uniform. In practice, z appears quite close to uniform, but we can also obtain looser but still quite acceptable bounds for ML-KEM-512 and ML-KEM-768 without making any assumptions on the distribution of z.

There are certainly some coefficients for which the conditional entropy of the error, and hence the uncompressed ciphertext coefficient, is 0: these are (z, c) combinations for which there is a single way of building that combination. We observe (through exhaustively searching the combination space) that, for ML-KEM-512 and ML-KEM-768, the combinations for which this can occur are only combinations in which the error takes on values ± 2, and a simplification would be to consider solely combinations build from error terms in $\{-1, 0, 1\}$. Unfortunately, for ML-KEM-1024, these degenerate combinations can occur with any error value, so we do not proceed with this analysis for ML-KEM-1024.

The conditional min-entropy for cases when the error is in $\{-1, 0, 1\}$ is

$$H_{min}(Y|X) = -\log_2 \max_{(z,c)\in X, e\in\{-1,0,1\}} \Pr[Y = e|X = (z, c)].$$

As seen in Table 1, for ML-KEM-512 and ML-KEM-768, the conditional min-entropy is the same for coefficients of both $\mathbf{e}_1$ and e_2: $h = h_{e_1} = h_{e_2} \approx 0.322$. Hence it does not matter whether an adversary uses their limited $\bar{\mathsf{F}}$ queries to learn $\mathbf{e}_1$ or e_2.

Noting that the degenerate case of error ± 2 occurs only 2/16 of the time, we can compute the probability of adversary guessing the confirmation code for the short confirmation code and the full confirmation code. For a confirmation code with subset S, the adversary's success probability is a weighted sum of the success probability given that i coefficients involve errors in $\{-1, 0, 1\}$ times the probability that i coefficients take on such error values, which is given in Equation (1). The results of calculating this numerically for ML-KEM-512 and ML-KEM-768 are shown in the "Worst-case analysis" section of Table 1, again conservatively assuming that the adversary is allowed $\mathsf{max} = k$ (i.e., all but one) queries to $\bar{\mathsf{F}}$. □

5.2 HQC

We briefly recall some necessary preliminaries. HQC operates on elements that can be interchangeably represented as bit strings (row vectors) or polynomials in $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$. We denote $\mathcal{R}_w$ as the space of binary strings with length n and Hamming weight w that is, strings with w non-zero entries. For $\mathbf{u}, \mathbf{v} \in \mathcal{R}$, we define the product $\mathbf{z} = \mathbf{u} \cdot \mathbf{v}$ as $\mathbf{z}_k = \sum_{i+j\equiv k \mod n} u_i v_j \quad \text{for } k \in \{0, 1, ..., n-1\}$.

We define three helper functions that each take as input a string x and an integer l: $\mathsf{truncate}(x, l)$ returns the truncation of x by l *bits*, $\mathsf{firstBytes}(x, l)$ returns the first l *bytes* of x, and $\mathsf{lastBytes}(x, l)$ returns the last l *bytes* of x.

We make use of Proposition 3.1.1 from [2].

```
$\underline{\mathsf{HQC_C.PKE_C.Enc_C}(\mathsf{pk}, \mathbf{m} \in \mathbb{F}_2^{128}; r)}$
$\mathbf{r}_1 \leftarrow \mathsf{PRF}(r, w_r, n, 1)$
$\mathbf{r}_2 \leftarrow \mathsf{PRF}(r, w_r, n, 2)$
$\mathbf{e} \leftarrow \mathsf{PRF}(r, w_e, n, 3)$
$\mathbf{u} \leftarrow \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$
$\mathbf{v} \leftarrow \mathsf{truncate}(\mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, \ell)$
$\mathsf{cd} \leftarrow \mathsf{lastBytes}(\mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, 1)$
$\mathbf{c} \leftarrow (\mathbf{u}, \mathbf{v})$
$\mathbf{return}\ (\mathbf{c}, \mathsf{cd})$

$\underline{\mathsf{HQC_C.Encaps}(\mathsf{pk})}$
$salt \leftarrow\!\!\$\ \mathbb{F}_2^{128}$
$\mathbf{m} \leftarrow\!\!\$\ \mathbb{F}_2^{128}$
$\theta \leftarrow \mathcal{G}(\mathbf{m} \| \mathsf{firstBytes}(\mathsf{pk}, 32) \| salt)$
$(\mathbf{c}, \mathsf{cd}) \leftarrow \mathsf{HQC_C.PKE_C.Enc_C}(\mathsf{pk}, \mathbf{m}; \theta)$
$K \leftarrow \mathsf{G}(\mathbf{m}, \mathbf{c}, \mathsf{cd})$
$\mathbf{return}\ (K, c = \mathbf{c} \| salt)$

$\underline{\mathsf{HQC_C.Decaps}(\mathsf{sk}, c = \mathbf{c} \| salt)}$
$\mathbf{m}' \leftarrow \mathsf{HQC.Dec}(\mathsf{sk}, \mathbf{c})$
$\theta' \leftarrow \mathcal{G}(\mathbf{m}' \| \mathsf{firstBytes}(\mathsf{pk}, 32) \| salt)$
$(\mathbf{c}', \mathsf{cd}') \leftarrow \mathsf{HQC_C.PKE_C.Enc_C}(\mathsf{pk}, \mathbf{m}'; \theta')$
$K \leftarrow \mathsf{G}(\mathbf{m}, \mathbf{c}, \mathsf{cd}')$
$\bar{K} \leftarrow \mathsf{G}(z, \mathbf{c})$
$\mathbf{if}\ \mathbf{m}' = \bot\ \mathbf{or}\ \mathbf{c} \neq \mathbf{c}'$ :
    $\mathbf{return}\ \bar{K}$
$\mathbf{return}\ K$
```

Fig. 13: Confirmation-code-augmented version $\mathsf{HQC_C}$ of HQC, with the underlying confirmation-code-augmented encryption algorithm $\mathsf{HQC_C.PKE_C.Enc_C}$ using function PRF. Changes introducing confirmation codes are highlighted in violet boxes.

Proposition 20. *Let $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$ be a random vector chosen uniformly among all binary vectors of length n and weight w and let $\boldsymbol{r} = (r_0, \ldots, r_{n-1})$ be a random vector chosen uniformly among all binary vectors of length n and weight w_r and independently of $\boldsymbol{x}$. Then denoting $\boldsymbol{z} = \boldsymbol{x} \cdot \boldsymbol{r}$ we have that for every $k \in \{0, \ldots, n-1\}$ the kth coordinate $\boldsymbol{z}_k$ of $\boldsymbol{z}$ is Bernoulli-distributed with parameter $\tilde{p} = \Pr[\boldsymbol{z}_k = 1] = \frac{1}{\binom{n}{w}\binom{n}{w_r}} \sum_{\substack{1 \leq \ell \leq \min(w, w_r) \\ \ell\ odd}} \binom{n}{\ell}\binom{n-\ell}{w-\ell}\binom{n-w}{w_r-\ell}$.*

Adding confirmation codes to HQC's PKE. We augment the PKE encryption algorithm of HQC with a confirmation code and, in encapsulation and decapsulation, embed it into the KEM key derivation as illustrated in Fig. 13 (the other algorithms remain unchanged). The randomness for encryption is derived from the message $\mathbf{m}$ and further inputs as $\theta \leftarrow \mathcal{G}(\mathbf{m} \| \mathsf{firstBytes}(\mathsf{pk}, 32) \| salt)$ using SHAKE. Encryption then pseudorandomly generates vectors $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{e}$ of appropriate weight from θ'; this is internally achieved by pseudorandomly generating uniform byte strings using SHAKE and then deriving vectors of the required weights. These vectors are used along with $\mathsf{pk} = (\mathbf{h}, \mathbf{s})$ to compute ciphertext $\mathbf{c} = (\mathbf{u}, \mathbf{v})$ with $\mathbf{u} = (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2)$ and $\mathbf{v} = \mathsf{truncate}(\mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, \ell)$. We model skipping re-encryption as skipping all lines where random values are generated from θ—in other words, the generation of $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{e}$. We use the last byte of $\mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$ (i.e., the $\mathbf{v}$ part of the ciphertext *before* truncation) as confirmation code, leveraging that the truncation by ℓ bits for the ciphertext, where ℓ depends on the HQC parameter set. We show that these truncated bits cannot be guessed with high probability without knowing $\mathbf{r}_1$, $\mathbf{r}_2$, or $\mathbf{e}$.

Theorem 21. *Let $\mathsf{HQC_C.PKE_C}$ be the confirmation-code-augmented PKE scheme of HQC using the $\mathsf{Enc_C}$ algorithm from Fig. 13 involving function $\mathsf{F} = \mathsf{PRF}$. Let $\mathcal{A}$ be an adversary in the cUP experiment making no ($\mathsf{max} = 0$) queries to its $\bar{\mathsf{F}}$ oracle. Then, using the heuristic assumption that* *the coordinates of* $\mathbf{e}$ *are independent, as was done in [1, 2], we obtain*

$$\mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{HQC_C.PKE_C,PRF},0}(\mathcal{A}) \le 2^{-\min(\ell,8)}.$$

For our proposed HQC confirmation code, the unpredictability advantage above is essentially also the fCOR advantage upper bound, as the other terms in the bound of Thm. 14 are negligible in comparison.

Proof. Consider $\mathbf{v} = \mathsf{truncate}(\mathbf{mG} + \mathbf{s}\cdot\mathbf{r}_2 + \mathbf{e}, \ell)$. The length of $\mathbf{s}\cdot\mathbf{r}_2$ is n bits, which is larger by ℓ bits than the length of $\mathbf{mG}$. Thus, the truncation of $\mathbf{v}$ involves truncating ℓ bits of $\mathbf{s}\cdot\mathbf{r}_2+\mathbf{e}$, where $\mathbf{s} = \mathbf{x} + \mathbf{h}\cdot\mathbf{y}$, for secret values $\mathbf{x}, \mathbf{y} \in \mathcal{R}_w$. We show that the distribution of the last bits coefficients of $\mathbf{v}$ is uniform. Consider

$$\mathbf{s}\cdot\mathbf{r}_2 + \mathbf{e} = \mathbf{x}\cdot\mathbf{r}_2 + \mathbf{h}\cdot\mathbf{y}\cdot\mathbf{r}_2 + \mathbf{e}.$$

Note that $\mathbf{h}$ is uniformly distributed, and $\mathbf{e}$ is uniformly distributed over strings of weight w_e. As was done in [1,2], we make the simplifying assumption that the coordinates of $\mathbf{e}$ are independent, and thus $\Pr[\mathbf{e}_k = 1]$ follows a Bernoulli distribution with parameter $p = \frac{w}{n}$. [8] Let $\bar{\mathbf{x}} = \mathbf{x}\cdot\mathbf{r}_2$ and $\bar{\mathbf{y}} = \mathbf{y}\cdot\mathbf{r}_2$. From Proposition 20 we have that each coordinate of $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ follows a Bernoulli distribution with the following parameter $\tilde{p}$ describing the distribution of coefficients:

$$\tilde{p} = \frac{1}{\binom{n}{w}\binom{n}{w_r}} \sum_{\substack{1\le\ell\le\min(w,w_r)\\ \ell \text{ odd}}} \binom{n}{\ell}\binom{n-\ell}{w-\ell}\binom{n-w}{w_r-\ell}.$$

Now, consider the output distribution of $\mathbf{h}\cdot\bar{\mathbf{y}}$. We have

$$(\mathbf{h}\cdot\bar{\mathbf{y}})_k = \sum_{i+j\equiv k \mod n} \mathbf{h}_i\bar{\mathbf{y}}_j.$$

Since $\mathbf{h}$ follows a uniform distribution, and $\bar{\mathbf{y}}$ follows a Bernoulli distribution with parameter $\tilde{p}$ we have that $\Pr[\mathbf{h}_i\bar{\mathbf{y}}_j = 1] = \frac{\tilde{p}}{2}$. Then the probability that $\sum_{i+j\equiv k \mod n}\mathbf{h}_i\bar{\mathbf{y}}_j = 1$, is described by the probability that after $k+1$ independent Bernoulli trials, the number of successes is odd. A 'success' in this case occurs with probability $\frac{\tilde{p}}{2}$. We have

$$\Pr[(\mathbf{h}\cdot\bar{\mathbf{y}})_k = 1] = \frac{1}{2} - \frac{1}{2}(1-\tilde{p})^{k+1}.$$

Now we consider the distribution of $\mathbf{z} = \mathbf{x}\cdot\mathbf{r}'_2 + \mathbf{h}\cdot\mathbf{y}\cdot\mathbf{r}'_2 + \mathbf{e}'$. We consider $\Pr[\mathbf{z}_k = 1]$. Here, $\mathbf{z}_k$ is the sum modulo 2 of 3 terms, each distributed according to an independent Bernoulli distribution. Thus we have

$$\begin{aligned}\Pr[\mathbf{z}_k = 1] = &\Pr[\mathbf{x}\cdot\mathbf{r}'_2 = 1, \mathbf{h}\cdot\mathbf{y}\cdot\mathbf{r}'_2 = 1, \mathbf{e}'_k = 1]\\ &+\Pr[\mathbf{x}\cdot\mathbf{r}'_2 = 1, \mathbf{h}\cdot\mathbf{y}\cdot\mathbf{r}'_2 = 0, \mathbf{e}'_k = 0]\\ &+\Pr[\mathbf{x}\cdot\mathbf{r}'_2 = 0, \mathbf{h}\cdot\mathbf{y}\cdot\mathbf{r}'_2 = 1, \mathbf{e}'_k = 0]\\ &+\Pr[\mathbf{x}\cdot\mathbf{r}'_2 = 0, \mathbf{h}\cdot\mathbf{y}\cdot\mathbf{r}'_2 = 0, \mathbf{e}'_k = 1].\end{aligned}$$

We observe computationally $\Pr[\mathbf{z}_k = 1]$ converges to 0.5. Concretely, we have

– For HQC-128, $n = 17,699$, $w = 66$, $w_e = w_r = 75$, $\tilde{p} = 0.2157612$, and $\ell = 5$ and $\Pr[z_k = 1] = 0.5$ for $k > 140$.

[8] In reality, since $\mathbf{e}$ is of a fixed low weight, $\Pr[\mathbf{e}_k = 1]$ depends on the weight of $(\mathbf{e}_0, ..., \mathbf{e}_{k-1})$.

Table 2: Performance impact of confirmation code on ML-KEM and HQC.
Measured with `liboqs` v0.12.1-dev using `mlkem-native` [25] `aarch64` on an Apple Silicon M2 Max processor.

Algorithm		**Short conf. code**		**Full-length conf. code**	
		Size (bytes)	**Slowdown**	**Size (bytes)**	**Slowdown**
ML-KEM-512	Encaps	12	3.4%	1152	23.6%
	Decaps		3.1%		23.6%
ML-KEM-768	Encaps	16	2.1%	1536	20.5%
	Decaps		2.1%		15.0%
ML-KEM-1024	Encaps	20	1.2%	1920	14.1%
	Decaps		1.6%		12.2%
HQC-128 with bug	Encaps	1	0.09%	—	—
	Decaps		0.25%		
HQC-128 bug fixed	Encaps	1	0.13%	—	—
	Decaps		0.19%		

- For HQC-192, $n = 35,851$, $w = 100$, $w_e = w_r = 114$, $\tilde{p} = 0.2362947$, and $\ell = 11$, and $\Pr[\mathbf{z}_k = 1] = 0.5$ for $k > 125$.
- For HQC-256, $n = 57,637$, $w = 131$, $w_e = w_r = 149$, $\tilde{p} = 0.2468457$, and $\ell = 37$, and $\Pr[\mathbf{z}_k = 1] = 0.5$ for $k > 119$.

We conclude that the last ℓ bits of $\mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$ used as confirmation code are uniformly random, and independent of the view of the $\mathcal{A}$ and so

$$\mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{HQC}_{\mathsf{C}}.\mathsf{PKE}_{\mathsf{C}},\mathsf{PRF},0}(\mathcal{A}) \leq \frac{1}{2^{\min(\ell,8)}}. \qquad \square$$

5.3 Performance Evaluation

ML-KEM. We adapted the ML-KEM implementation in `liboqs` [22, 27] (based on `mlkem-native` [25]) to include confirmation codes as specified in Figs. 11 and 12. We evaluated two options: a short confirmation code, using the first and last coefficient of uncompressed ciphertext vector for the confirmation code (i.e., $S = \{1, 256\}$), and a full-length confirmation code (i.e., $S = \{1, \ldots, 256\}$). Our performance evaluation (see Table 2) shows that using the short confirmation code leads to a slow-down in the encapsulation and decapsulation algorithms of at most 3.4% across ML-KEM-512, ML-KEM-768, and ML-KEM-1024. Using the entire uncompressed ciphertext for the confirmation code has a much more significant performance impact due to the cost of hashing such a long input into key derivation, relative to the otherwise low cost of ML-KEM encapsulation/decapsulation. Table 2 shows the performance impact in more detail. Fortunately, as we have shown in Thm. 19, the short confirmation code ($|S| = 2$) already suffices to obtain a cUP advantage upper bound that ensures faulty implementations will fail noticeably (with probability about 1/3, see Table 1) in each test run.

HQC. We likewise augmented the HQC implementation in `liboqs` [22, 27] with confirmation codes as specified in Fig. 13 and evaluated it for the HQC-128 parameter set. Using the last byte of as confirmation code (for efficiency, given all the APIs are byte-oriented), we measured a slow-down in encapsulation and decapsulation of at most 0.25%. We applied confirmation codes to both the version of code *with* the re-encryption bug [23] and the patched version with the bug fixed, and experimentally confirmed that our approach works: the 1-byte confirmation code indeed detected the flaw in the buggy implementation, reliably causing the basic correctness tests of `liboqs` to fail.

6 Conclusion and Discussion

In this paper, we bring the methodology of cryptographically tying security to functional correctness through confirmation codes, proposed by Fischlin and Günther [7] in the form of "verifiable verification" for digital signatures, to the realm of key encapsulation mechanisms (KEMs). We developed a modified Fujisaki–Okamoto (FO) transform that binds re-encryption to functionality, making the correct implementation of the FO re-encryption verifiable for implementers. The core idea is to export an unpredictable confirmation code from the underlying PKE scheme and include it in the key derivation. We showed that this approach ensures that a faulty implementation which skips re-encryption will be noticeably incorrect, and hence exposed through basic correctness tests. We applied this technique to ML-KEM and HQC to present confirmation-code-augmented variants that leverage the entropy lost through ciphertext compression or truncation with negligible overhead, and experimentally confirmed that this approach successfully surfaces the re-encryption implementation flaw in HQC's reference implementation.

A PERMANENT SAFETY NET OR A TEMPORARY BUG TESTING TECHNIQUE? One could see verifiable decapsulation as a property that should be permanently integrated into KEM designs. In this way, one not only captures local implementation errors before deployment, but can also detect flaws in a remote KEM implementation when used, e.g., within a larger protocol. Also, any future code modifications would automatically benefit from the technique, providing a permanent safety net.

On the other hand, one could see verifiable decapsulation as a bug-testing technique. In this view, one would use confirmation codes in pre-deployment version of code for enhanced correctness tests. The deployed code version would then remove the confirmation codes again, reverting to the standard KEM scheme. This approach avoids the overhead introduced by confirmation codes. However, maintaining two implementations that differ in lower-level cryptographic instructions may itself be prone to errors, and skipping such a step could be a development "optimization" taken by a busy implementer.

We tend to view verifiable decapsulation as a design principle permanently embedded in KEMs, and see our experimental results on ML-KEM and HQC as supporting this, demonstrating that the overhead introduced by confirmation codes is very small.

FUTURE WORK. In order to model how a faulty implementation might behave, we presented a notion of confirmation code unpredictability that gave the adversary access to all inputs and instead limited its access to some internal function F. While this approach avoids trivial confirmation codes possible in [7] (like the message which is PKE-encrypted) and captures an incomplete re-encryption implementation step well in the KEMs we study, not every cryptographic scheme admits identifying such function F, limiting access to which would match intuition. Not all post-quantum KEMs we looked at admit natural confirmation codes in our paradigm. For example, FrodoKEM [20] does not use ciphertext compression or truncation like ML-KEM or HQC which we leveraged for obtain unpredictable confirmation codes. It would be interesting to explore different formalisms for restricting which computations a faulty implementation may perform, through limiting input access, computation time, etc.

References

1. Aguilar-Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G., Bos, J., Dion, A., Lacan, J., Robert, J.M., Veron, P.: HQC. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions
2. Aragon, N., Gaborit, P., Zémor, G.: HQC-RMRS, an instantiation of the HQC encryption framework with a more efficient auxiliary error-correcting code (2020), https://arxiv.org/abs/2005.10741
3. Bellare, M., Davis, H., Günther, F.: Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 3–32. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45724-2_1
4. Bernstein, D.J., Persichetti, E.: Towards KEM unification. Cryptology ePrint Archive, Report 2018/526 (2018), https://eprint.iacr.org/2018/526
5. Bindel, N., Hamburg, M., Hövelmanns, K., Hülsing, A., Persichetti, E.: Tighter proofs of CCA security in the quantum random oracle model. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 61–90. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-36033-7_3
6. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Online-extractability in the quantum random-oracle model. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 677–706. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_24
7. Fischlin, M., Günther, F.: Verifiable verification in cryptographic protocols. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 3239–3253. ACM Press (Nov 2023). https://doi.org/10.1145/3576915.3623151
8. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_34
9. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of Cryptology **26**(1), 80–101 (Jan 2013). https://doi.org/10.1007/s00145-011-9114-1
10. Heninger, N.: Biased Nonce Sense: Lattice attacks against weak ECDSA signatures in the wild. Talk at the Workshop on Attacks in Cryptography 2 (WAC2), Crypto 2019 (Aug 2019), https://crypto.iacr.org/2019/affevents/wac/medias/Heninger-BiasedNonceSense.pdf
11. Heninger, N.: Implementation footguns for post-quantum cryptography. Talk at the Real World Post Quantum Workshop (RWPQC 2024) (Mar 2024), https://na-admin.eventscloud.com/docs/9769/414552
12. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 341–371. Springer, Cham (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2_12
13. Hövelmanns, K.: Generic constructions of quantum-resistant cryptosystems. doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek (2021). https://doi.org/10.13154/294-7758
14. Hövelmanns, K., Hülsing, A., Majenz, C.: Failing gracefully: Decryption failures and the Fujisaki-Okamoto transform. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part IV. LNCS, vol. 13794, pp. 414–443. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22972-5_15
15. Hövelmanns, K., Kiltz, E., Schäge, S., Unruh, D.: Generic authenticated key exchange in the quantum random oracle model. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 389–422. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45388-6_14
16. Hövelmanns, K., Majenz, C.: A note on failing gracefully: Completing the picture for explicitly rejecting fujisaki-okamoto transforms using worst-case correctness. In: Saarinen, M.J., Smith-Tone, D. (eds.) Post-Quantum Cryptography - 15th International Workshop, PQCrypto 2024, Part II. pp. 245–265. Springer, Cham (Jun 2024). https://doi.org/10.1007/978-3-031-62746-0_11
17. Jiang, H., Zhang, Z., Chen, L., Wang, H., Ma, Z.: IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 96–125. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_4

18. Jiang, H., Zhang, Z., Ma, Z.: Key encapsulation mechanism with explicit rejection in the quantum random oracle model. In: Lin, D., Sako, K. (eds.) PKC 2019, Part II. LNCS, vol. 11443, pp. 618–645. Springer, Cham (Apr 2019). https://doi.org/10.1007/978-3-030-17259-6_21
19. Kuchta, V., Sakzad, A., Stehlé, D., Steinfeld, R., Sun, S.: Measure-rewind-measure: Tighter quantum random oracle model proofs for one-way to hiding and CCA security. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 703–728. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45727-3_24
20. Naehrig, M., Alkim, E., Bos, J., Ducas, L., Easterbrook, K., LaMacchia, B., Longa, P., Mironov, I., Nikolaenko, V., Peikert, C., Raghunathan, A., Stebila, D.: FrodoKEM. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
21. National Institute of Standards and Technology: Module-lattice-based key-encapsulation mechanism standard. Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 203, U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.203
22. Open Quantum Safe project: liboqs, https://openquantumsafe.org/liboqs/
23. Open Quantum Safe project: Correctness error in HQC decapsulation. Reported by Célian Glénaz and Dahmun Goudarzi. CVE-2024-54137 (Dec 2024), https://github.com/open-quantum-safe/liboqs/security/advisories/GHSA-gpf4-vrrw-r8v7
24. Poulsen, K.: Behind iPhone's critical security bug, a single bad 'goto'. https://www.wired.com/2014/02/gotofail/ (Feb 2014)
25. PQ Code Package project: mlkem-native, https://github.com/pq-code-package/mlkem-native/
26. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 520–551. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_17
27. Stebila, D., Mosca, M.: Post-quantum key exchange for the internet and the open quantum safe project. In: Avanzi, R., Heys, H.M. (eds.) SAC 2016. LNCS, vol. 10532, pp. 14–37. Springer, Cham (Aug 2016). https://doi.org/10.1007/978-3-319-69453-5_2

A Basic Definitions

A.1 Security Notions for PKE Schemes

We now recall the formal definitions of *One-Wayness* (OW-CPA) and *Indistinguishability under Chosen Plaintext Attacks* (CPA) security for PKE schemes.

Definition 22 (OW, IND-CPA). *Given a PKE* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, we define the* OW-CPA *game as in Fig. 14, and the* OW-CPA *advantage of an adversary* $\mathcal{A}$ *as*

$$\mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{OW\text{-}CPA}}(\mathcal{A}) = \Pr\left[\mathsf{OW\text{-}CPA}_{\mathsf{PKE}}(\mathcal{A}) \to 1\right] \ ,$$

where the probability is taken over the randomness in the OW-CPA *game and the internal coins of* $\mathcal{A}$.

We also define the IND-CPA *game as in Fig. 14, and the* IND-CPA *advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *as*

$$\mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}) = \left|\Pr\left[\mathsf{IND\text{-}CPA}_{\mathsf{PKE}}(\mathcal{A}) \to 1\right] - \frac{1}{2}\right| \ ,$$

where the probability is taken over the randomness in the IND-CPA *game and the internal coins of* $\mathcal{A}$.

```
Game OW-CPA_PKE(A):
01 $(\mathsf{pk}, sk) \leftarrow \mathsf{KeyGen}$
02 $m^* \leftarrow_\$ \mathcal{M}$
03 $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m^*)$
04 $m' \leftarrow \mathcal{A}(\mathsf{pk}, c^*)$
05 return $[\![m' = m]\!]$

Game IND-CPA_PKE(A):
06 $(\mathsf{pk}, sk) \leftarrow \mathsf{KeyGen}$
07 $b \leftarrow_\$ \{0,1\}$
08 $(m_0, m_1, st) \leftarrow A_1(\mathsf{pk})$
09 $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
10 $b' \leftarrow \mathcal{A}(\mathsf{pk}, c^*, st)$
11 return $[\![b' = b]\!]$
```

Fig. 14: OW-CPA and CPA game for PKE scheme PKE.

A.2 Security Notions for Key Encapsulation Mechanisms

Here, we recall the standard security notions for key encapsulation: Indistinguishability under Chosen Ciphertext Attacks (IND-CCA).

Definition 23 (IND-CCA). *We define the* IND-CCA *game as in Fig. 15 and the* IND-CCA *advantage function of an adversary $\mathcal{A}$ (with binary output) against KEM as*

$$\mathrm{Adv}_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}) = |\Pr\left[\mathsf{IND\text{-}CCA}_{\mathsf{KEM}}^{\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2}| \ .$$

```
Game IND-CCA_KEM:
01 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$
02 $b \leftarrow_\$ \{0,1\}$
03 $(K_0, c^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
04 $K_1 \leftarrow_\$ \mathcal{K}$
05 $b' \leftarrow A^{\mathsf{Decaps}}(\mathsf{pk}, c^*, K_b)$
06 return $[\![b' = b^*]\!]$

Decaps($c \neq c^*$):
07 $K \leftarrow \mathsf{Decaps}(\mathsf{sk}, c)$
08 return $K$
```

Fig. 15: IND-CCA game for KEM = (KeyGen, Encaps, Decaps).

B Recalling the OW-CPA bounds for the T-transform given in [12]

We now restate [12, Thms. 3.1 and 3.2]. In the theorems, PKE_1 denotes the PKE scheme $\mathsf{T}[\mathsf{PKE}, \mathsf{G}]$, and OW-CPA-PCVA denotes OW-CPA security even in the presence of two additional oracles PCO and CVO. Simplifying the given bounds as indicated in Thm. 11 by dropping the additional oracles is straightforward, but we still detail this below the theorems for the sake of completeness.

Theorem 24 ([12, Theorem 3.1]). *If* PKE *is δ-correct, then* PKE_1 *is δ_1-correct in the random oracle model with $\delta_1(q_\mathsf{G}) = q_\mathsf{G} \cdot \delta$. Assume* PKE *to be γ-spread. Then, for any* OW-CPA-PCVA *adversary $\mathcal{B}$ that issues at most q_G queries to the random oracle* G*, q_P queries to a plaintext checking oracle* PCO*, and q_V queries to a validity checking oracle* CVO*, there exists an* OW-CPA *adversary $\mathcal{A}$ such that*

$$\mathrm{Adv}_{\mathsf{PKE}_1}^{\mathsf{OW\text{-}CPA\text{-}PCVA}}(\mathcal{B}) \leq (q_\mathsf{G} + q_P) \cdot \delta + q_V \cdot 2^{-\gamma} + (q_\mathsf{G} + q_P + 1) \cdot \mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{OW\text{-}CPA}}(\mathcal{A})$$

and the running time of $\mathcal{A}$ is about that of $\mathcal{B}$. Furthermore, PKE_1 *is rigid.*

Theorem 25 ([12, Theorem 3.2]). *Assume* PKE *to be* δ*-correct and* γ*-spread. Then, for any* $\mathsf{OW\text{-}CPA\text{-}PCVA}$ *adversary* $\mathcal{B}$ *that issues at most* q_{G} *queries to the random oracle* G*,* q_P *queries to a plaintext checking oracle* PCO*, and* q_V *queries to a validity checking oracle* CVO*, there exists an* INDCPA *adversary* $\mathcal{A}$ *such that*

$$\mathrm{Adv}^{\mathsf{OW\text{-}CPA\text{-}PCVA}}_{\mathsf{PKE}_1}(\mathcal{B}) \leq (q_{\mathsf{G}} + q_P) \cdot \delta + q_V \cdot 2^{-\gamma} + \frac{2q_{\mathsf{G}} + 1}{|\mathcal{M}|} + 3 \cdot \mathrm{Adv}^{\mathsf{INDCPA}}_{\mathsf{PKE}}(\mathcal{A})$$

and the running time of $\mathcal{A}$ *is about that of* $\mathcal{B}$*.*

In Thm. 11, we are only interested in plain $\mathsf{OW\text{-}CPA}$ security. By definition, $\mathsf{OW\text{-}CPA}$ security is $\mathsf{OW\text{-}CPA\text{-}PCVA}$ security with $q_P = 0$ queries to the additionally provided plaintext checking oracle PCO and $q_V = 0$ queries to the validity checking oracle CVO, thus the term $(q_{\mathsf{G}} + q_P + 1) \cdot \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A})$ in Theorem 3.1 simplifies to $(q_{\mathsf{G}} + 1) \cdot \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A})$.

The two terms involving δ and γ are shared by both bounds and account for how both proofs simulate the two additional oracles PCO and CVO. Since we are only interested in plain $\mathsf{OW\text{-}CPA}$ security, we can dismiss the two oracles and the respective terms vanish.

C Proof that the TC-transform maintains code unpredictability (Proofs of Thm. 11)

We briefly repeat the proof's main idea: to an cUP adversary trying to predict the code belonging to a ciphertext, there is not too much of a difference between attacking $\mathsf{PKE}_{\mathsf{C}}$ and its derandomized version $\mathsf{TC}[\mathsf{PKE}_{\mathsf{C}}, \mathsf{G}]$. Since the only additional attack surface is that it might be more easy to invert derandomized ciphertexts, thus breaking $\mathsf{OW\text{-}CPA}$ security of $\mathsf{T}[\mathsf{PKE}, \mathsf{G}]$, we bound the success via a $\mathsf{OW\text{-}CPA}$ reduction both in the ROM (Thm. 11).

C.1 Proof of ROM Thm. 11

Consider the two games given in Fig. 16.

```
GAMES $G_0 - G_1$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$
$m \leftarrow_\$ \mathcal{M}$
$r \leftarrow \mathsf{G}(m)$                         // $G_0$
$r \leftarrow_\$ \mathcal{R}$                         // $G_1$
$(\mathsf{cd}, c) \leftarrow \mathsf{Enc}^{\mathsf{PRF}}_{\mathsf{C}}(\mathsf{pk}, m; r)$
$\mathsf{cd}' \leftarrow \mathcal{A}^{\mathsf{G}}(\mathsf{pk}, c)$
return $[\![\mathsf{cd} = \mathsf{cd}']\!]$
```

```
$\mathcal{B}^{\mathsf{G}}(\mathsf{pk}, c^*)$
$\mathfrak{L}_{\mathsf{G}} \leftarrow \emptyset$
$\mathsf{cd}' \leftarrow \mathcal{A}_{\mathsf{TC}[\mathsf{PKE}^{\mathsf{PRF}}_{\mathsf{C}}\mathsf{G}],\max}(\mathsf{pk}, \mathsf{sk}, m, c, r)$
$m' \leftarrow_\$ \mathcal{M}$
for $m \in \mathfrak{L}_{\mathsf{G}}$
    if $\mathsf{Enc}(\mathsf{pk}, m; \mathsf{G}(m)) = c^*$
        $m' \leftarrow m$
return $m'$
```

Fig. 16: Games G_0 and G_1 and adversary $\mathcal{B}$ for the proof of Thm. 11. List $\mathfrak{L}_{\mathsf{G}}$ stores $\mathcal{A}$'s random oracle queries to G.

Game G_0 is the confirmation code unpredictability game against $\mathsf{TC}[\mathsf{PKE}^{\mathsf{PRF}}_{\mathsf{C}}\mathsf{G}]$.

$$|\Pr[G_0 \Rightarrow 1]| = \mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{TC}[\mathsf{PKE}_{\mathsf{C}},\mathsf{G}]}(\mathcal{A}) \ .$$

Game G_1 differs from G_0 by decoupling the involved randomness r from the message: when computing $(c, \mathsf{cd}) \leftarrow \mathsf{Enc_C}(\mathsf{pk}, m; r)$, we now use a uniformly random value r that is independent of m instead of using $r \leftarrow \mathsf{G}(m_0)$. In other words, we switch from the $\mathsf{Enc_C}$ algorithm of $\mathsf{TC}[\mathsf{PKE_C}, \mathsf{G}]$ to that of $\mathsf{PKE_C}$. Game G_1 thus is exactly the unpredictability game for the underlying augmented scheme $\mathsf{PKE_C}$.

$$|\Pr[G_1 \Rightarrow 1]| = \mathrm{Adv}^{\mathsf{cUP}}_{\mathsf{PKE_C}}(\mathcal{A}) \ .$$

Since we model G as a random oracle, the change is unnoticed by $\mathcal{A}$ unless it queries G on m, thereby breaking one-way security of $\mathsf{TC}[\mathsf{PKE}, \mathsf{G}]$. To formalize this, we start by calling the event QUERY.

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_0 \Rightarrow 1]| \leq \Pr[\mathrm{QUERY}] \ .$$

We now define a $\mathsf{OW\text{-}CPA}$ adversary $\mathcal{B}$ against $\mathsf{T}[\mathsf{PKE}, \mathsf{G}]$ in Fig. 16 that wins if QUERY occurs: $\mathcal{B}$ forwards its challenge ciphertext to $\mathcal{A}$. It forwards $\mathcal{A}$'s random oracle queries to G to its own random oracle, and also keeps track of these queries by storing them in a list $\mathfrak{L}_\mathsf{G}$. If QUERY occurs, the challenge plaintext appears in $\mathfrak{L}_\mathsf{G}$. Since $\mathsf{T}[\mathsf{PKE}, \mathsf{G}]$ encrypts deterministically, $\mathcal{B}$ can go through the list after $\mathcal{A}$ has finished and identify the right plaintext.

$$\Pr[\mathrm{QUERY}] \leq \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{T}[\mathsf{PKE},\mathsf{G}]}(\mathcal{B}) \ .$$

Adding the inequalities gives the desired bound.

D IND-CCA security of our transformations $\mathsf{UC}^{\not\perp}_m$ and $\mathsf{FOC}^{\not\perp}_m$

D.1 $\mathsf{UC}^{\not\perp}_m$ is IND-CCA secure in the ROM

Our ROM theorem below establishes a bound that resembles the bound $\mathsf{U}^{\not\perp}_m$ that was given in [13, Thm. 2.1.7], except that Thm.2.1.7 bounded correctness errors slightly differently.

Theorem 26 (ROM security of $\mathsf{UC}^{\not\perp}_m$). *Let $\mathsf{PKE_C}$ be a deterministic confirmation augmented PKE with code and ciphertext generation independent of G, and assume the underlying PKE to be δ-worst-case correct. If there are any random oracles used in the construction of $\mathsf{PKE_C}$, let $q_{\mathsf{Enc},\mathsf{O}}$ denote an upper bound on the number of random oracle queries that $\mathsf{Enc_C}$ trigger upon a single invocation. Let $\mathsf{H} : \mathcal{PK} \to \{0,1\}^\gamma$ be a fixed-output length function.*

Let $\mathcal{A}$ be an adversary against $\mathsf{IND\text{-}CCA}$ security of $\mathsf{KEM_C} = \mathsf{UC}^{\not\perp}_m[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}]$, issuing at most q_D many queries to the decapsulation oracle for $\mathsf{KEM_C}$, and at most q_O, q_G, q_J queries to the random oracle used in PKE and the random oracles G, J, respectively.

Then in the ROM, there exists a $\mathsf{OW\text{-}CPA}$ adversary $\mathcal{B}$ against PKE such that

$$\mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM_C}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{B}) + (q_\mathsf{O} + q_\mathsf{G}(q_{\mathsf{Enc},\mathsf{O}}) + 1) \cdot \delta + \frac{q_\mathsf{J}}{|\mathcal{M}|} \ ,$$

$\mathcal{B}$ issues at most $q_\mathsf{O} + q_\mathsf{G} \cdot (q_{\mathsf{Enc},\mathsf{O}})$ many queries to oracle O, and the running time of $\mathcal{B}$ is about that of $\mathcal{A}$.

Proof. Our proof is close to the proof for $\mathsf{U}^{\not\perp}_m$ that was given in [13]. Recall that we denote by $\mathsf{Enc}(\mathsf{pk}, m)$ the algorithm that runs $\mathsf{Enc_C}(\mathsf{pk}, m)$ and returns only the ciphertext. Likewise, let Code denote the algorithm that runs $\mathsf{Enc_C}$ and outputs only cd. We start by recalling its high-level idea and discussing how to adapt it to $\mathsf{UC}^{\not\perp}_m$: $\mathsf{IND\text{-}CCA}$ attacker $\mathcal{A}$ is given a challenge plaintext $c^* = \mathsf{Enc}(\mathsf{pk}, m^*)$ and a challenge key, which is either random or $\mathsf{G}(m^*)$. In the ROM, $\mathsf{G}(m^*)$ is indistinguishable from random unless $\mathcal{A}$ queries G on m^*, which can be leveraged by one-way reduction $\mathcal{B}$ that uses $\mathcal{A}$ to invert its challenge plaintext c^*. To help $\mathcal{B}$ simulate the oracle for Decaps

without sk, the proof utilizes that Enc is deterministic – $\mathcal{B}$ uses Enc to recognize when a random oracle query m to G is connected to a decapsulation query c, in which case it outputs the same key. The simulation thus switches the connection between the oracles from $\mathsf{Decaps}(\mathsf{sk}, c) = \mathsf{G}(\mathsf{Dec}(\mathsf{sk}, c))$ to $\mathsf{G}(m) = \mathsf{Decaps}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m))$. $\mathcal{A}$ only notices this when finding a message m for which $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) \neq m$, which explains the δ-term.

To adapt the proof to $\mathsf{UC}_m^{\not\perp}$, we note that we can easily integrate random oracle input $\mathsf{H}(\mathsf{pk})$ into the reasoning above since pk is a public value. To integrate cd, we leverage that $\mathsf{Enc_C}$ is assumed to be deterministic and one can thus easily identify the code that belongs to a plaintext. To enable the simulation of Decaps for $\mathsf{UC}_m^{\not\perp}$, our proof essentially redefines the two oracles such that their connection switches from $\mathsf{Decaps}(\mathsf{sk}, c) = \mathsf{G}(m', \mathsf{H}(\mathsf{pk}), \mathsf{Enc}(\mathsf{pk}, m'))$, where $m' = \mathsf{Dec}(\mathsf{sk}, c)$, to $\mathsf{G}(m, \mathsf{H}(\mathsf{pk}), \mathsf{Enc}(\mathsf{pk}, m)) = \mathsf{Decaps}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m))$.

We now recall the ROM proof of $\mathsf{U}_m^{\perp}$ in more detail and discuss in the relevant places how we adapt it to accommodate the additional hash inputs.

Game 1. To prepare the simulation of the oracle for Decaps, the proof first replaces 'implicit rejection' keys with uniformly random values. This goes unnoticed unless $\mathcal{A}$ queries J on the rejection seed z, which happens with probability $q_\mathsf{J}/|\mathcal{M}|$. This game needs no adapting.

Game 2. To prepare the simulation of the oracle for Decaps, the proof now 'patches encryption into' the lazily sampled random oracle G: upon a query m to G, the game computes the encryption c of m and stores c together with the randomly sampled key in an additional book-keeping list $\mathfrak{L}_D$. Upon a query c to the decapsulation oracle, the game consults list $\mathfrak{L}_D$ to keep $\mathcal{A}$'s view consistent, i.e., to return the key that was already associated to that ciphertext (if it already exists). Effectively, this switches the association from $\mathsf{Decaps}(\mathsf{sk}, c) = \mathsf{G}(\mathsf{Dec}(sk, c))$ to $\mathsf{G}(m) = \mathsf{Decaps}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m))$. Thus, the switch of association could only create an inconsistency in one of two cases:

- $\mathcal{A}$ queries G on a message that exhibits decryption failure – in that case, the proper input to G, m', would differ from the chosen input m. This is reflected in the δ-term.
- $\mathcal{A}$ could request two distinct ciphertexts $c_1 \neq c_2$ that both decrypt to the same message m. In game 1, the decapsulation oracle would respond to both queries with the same value $\mathsf{G}(m)$. This would not necessarily be the case in game 2. This inconsistency is prevented by requiring PKE to be rigid, i.e., by requiring that for any key pair and any ciphertext c, it always holds that $m' = \mathsf{Dec}(\mathsf{sk}, c) = \perp$ or $\mathsf{Enc}(\mathsf{pk}, m') = c$.

We now **adapt Game 2** to accommodate the additional hashing of confirmation code and public-key identifier $\mathsf{H}(\mathsf{pk})$. Intuitively, this works since pk is a public value and since we assume code generation to be deterministic: upon a query (m, h, cd) to G, the game checks if $h = \mathsf{H}(\mathsf{pk})$. If yes, it computes the code $\mathsf{cd}' \leftarrow \mathsf{Code}(\mathsf{pk}, m)$ and checks if $\mathsf{cd} = \mathsf{cd}'$. If yes, it computes the encryption c of m and stores c together with the randomly sampled key in book-keeping list $\mathfrak{L}_D$. Upon a decapsulation query, the game then again consults list $\mathfrak{L}_D$ to keep $\mathcal{A}$'s view consistent. This switches the association from $\mathsf{Decaps}(\mathsf{sk}, c) = \mathsf{G}(\mathsf{Dec}(\mathsf{sk}, c), \mathsf{H}(\mathsf{pk}), \mathsf{Code}(\mathsf{Dec}(\mathsf{sk}, c)))$ to $\mathsf{G}(m, \mathsf{H}(\mathsf{pk}), \mathsf{Code}(\mathsf{pk}, m)) = \mathsf{Decaps}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m))$. With the same reasoning as in the original proof, we note that this goes unnoticed unless $\mathcal{A}$ queries G on a message that exhibits decryption failure, which is upper-bounded by the same δ-term as before. (We note that if no decryption failure occurred, then the confirmation codes $\mathsf{Code}(\mathsf{pk}, m)$ and $\mathsf{Code}(\mathsf{pk}, m')$ for $m' \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m))$ will not differ for any queried message, thus not leading to any inconsistency.)

Note that we do not require PKE to be rigid because unlike $\mathsf{U}_m^{\not\perp}$, our transformation $\mathsf{UC}_m^{\not\perp}$ performs re-encryption – if $\mathcal{A}$ requests two distinct ciphertexts $c_1 \neq c_2$ that both decrypt to the same message m, at least one of them will land in the implicit rejection branch, meaning their decapsulations would already have differed in game 1.

Game 3. The last game argues indistinguishability of the challenge session key via one-wayness of PKE: Game 3 raises flag CHAL and immediately aborts on the event that $\mathcal{A}$ queries G on the input that would generate the honest challenge key K_0, so on the input m^* with m^* being the challenge plaintext. (In our case, this changes to the input $(m^*, \mathsf{H}(\mathsf{pk}), \mathsf{Code}(\mathsf{pk}, m^*))$ with m^* being the challenge plaintext.) If this query never occurs, $\mathcal{A}$ never sees the honest challenge key, thus has no chance distinguishing it from random beyond random guessing, $\mathcal{A}$' success probability in game 3 thus is $1/2$. The change in $\mathcal{A}$'s success probability between games 2 and 3 is upper-bounded by the probability of CHAL, and event CHAL can be used to break the one-wayness of PKE. Our adapted adversary $\mathcal{B}$ simulates game 3 for $\mathcal{A}$. Since Enc and code generation are assumed to be deterministic, $\mathcal{B}$ is able to recognize CHAL and thus m^*: upon each of $\mathcal{A}$'s G queries $(m, \mathsf{H}(\mathsf{pk}), \mathsf{cd})$, $\mathcal{B}$ checks if $\mathsf{Code}(\mathsf{pk}, m) = \mathsf{cd}$ and, if yes, if $\mathsf{Enc}(\mathsf{pk}, m)$ equals its challenge ciphertext c^*. If yes, it immediately aborts $\mathcal{A}$ and returns m to its $\mathsf{OW\text{-}CPA}$ game. Since Enc is assumed to be deterministic, we have $\mathsf{Enc}(\mathsf{pk}, m) = c$ and $\mathcal{B}$ thus returns m^* unless m^* exhibits decryption failure, which happens with probability at most δ. □

D.2 $\mathsf{UC}_m^{\not\perp}$ is also $\mathsf{IND\text{-}CCA}$ secure in the QROM

Our second theorem adapts the ROM result, Thm. 26, to the QROM. It essentially recovers the QROM security bound for $\mathsf{U}^{\not\perp}$ that was given in [5, Thm. 2] for our augmented variant.

Theorem 27 (QROM security of $\mathsf{UC}_m^{\not\perp}$). *Let G be a quantum-accessible random oracle, let $\mathsf{H} : \mathcal{PK} \to \{0,1\}^\gamma$ be a fixed-output length function, and let $\mathsf{J} : \mathcal{K}_\mathsf{J} \times \mathcal{C} \to \mathcal{K}$ be a PRF. Let PKE_C be a deterministic confirmation augmented PKE with code and ciphertext generation both are independent of G, and let PKE denote the PKE scheme associated with PKE_C (i.e. it uses as Enc the algorithm that runs Enc_C and only outputs c). Assume PKE to be ϵ-injective, i.e., assume that*

$$\Pr[\mathsf{Enc}(\mathsf{pk}, -) \text{ is not injective}] \leq \epsilon \ ,$$

where the probability is being taken over the internal coins of KeyGen and the choice of the random oracles involved in the construction of PKE_C (if any).

Let $\mathcal{A}$ be an $\mathsf{IND\text{-}CCA}$ adversary against $\mathsf{KEM}_\mathsf{C} = \mathsf{UC}_m^{\not\perp}[\mathsf{PKE}_\mathsf{C}, \mathsf{G}, \mathsf{H}, \mathsf{J}]$ that issues at most q_D decapsulation queries. Then there exist three adversaries with about the same time/resources as $\mathcal{A}$:

- *an $\mathsf{OW\text{-}CPA}$ adversary B_1 against PKE;*
- *an FFC adversary B_2 against PKE, returning a list of at most q_D many ciphertexts; and*
- *a PRF adversary B_3 against J*

such that

$$\mathrm{Adv}_{\mathsf{KEM}_\mathsf{C}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}) \leq 2\sqrt{\mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{OW\text{-}CPA}}(\mathcal{B}_1)} + \mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{FFC}}(\mathcal{B}_2) + 2\mathrm{Adv}_{\mathsf{J}}^{\mathsf{PRF}}(\mathcal{B}_3) + \epsilon \ .$$

Remark 28. Before discussing the proof, we note that for perfectly correct schemes, $\mathsf{Enc}(\mathsf{pk}, -)$ will be non-injective with probability 0; the ϵ-term thus vanishes in this case. We also note that for PKE schemes that are deterministic, perfectly correct and that satisfy a stronger security property called *disjoint simulatability*, it may be possible to achieve a tighter QROM security bound by re-doing/adapting the proof of [26, Thm. 4.2].

Proof. The proof closely follows a proof [5, Thm. 2] for transformation $\mathsf{U}^{\not\perp}$. We will thus first recap that proof and then discuss how to adapt it.

On a high level, the proof for $\mathsf{U}^{\not\perp}$ resembles the ROM reasoning above: base indistinguishability of the challenge session key on $\mathsf{OW\text{-}CPA}$ security, and to do so, replace $\mathcal{A}$'s oracles with sk-independent simulations. For the transformation $\mathsf{U}^{\not\perp}$, this means redefining the oracles such that $\mathsf{G}(m, c) = \mathsf{Decaps}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m))$ instead of $\mathsf{Decaps}(\mathsf{sk}, c) = \mathsf{G}(\mathsf{Dec}(\mathsf{sk}, c), c)$. Since the random oracle queries now are in superposition, the proof needs to adapt how to

1. simulate G and Decaps without sk in a consistent way (no bookkeeping);
2. reason about the cases in which the simulation fails; and
3. find the challenge message m^* within the random oracle queries.

To deal with #1, the proof for $\mathsf{U}^{\not\perp}$ 'patches encryption into' the random oracle by redefining $\mathsf{G}(m, c) = \mathsf{R}(c)$ whenever $c = \mathsf{Enc}(\mathsf{pk}, m)$, where R is a random function. With this change, the proof can simulate Decaps by setting $\mathsf{Decaps}(\mathsf{sk}, c) = \mathsf{R}(c)$.

The proof for $\mathsf{U}^{\not\perp}$ notes that the simulation fails (#2) in three cases: a) the redefined G no longer produces independent responses for distinct messages, which is prevented by requiring that $\mathsf{Enc}(\mathsf{pk}, -)$ is ϵ-injective. b) $\mathcal{A}$ requests a c that goes into the rejection branch and is able to distinguish $\mathsf{R}(c)$ from $\mathsf{J}(z, c)$, which is captured via PRF. c) $\mathcal{A}$ requests a decapsulation of a ciphertext that doesn't decrypt to its originating plaintext, which is captured via FFC.

To deal with #3, the proof for $\mathsf{U}^{\not\perp}$ utilizes double-sided OwtH (Lem. 6), which incurs a quadratic loss. In the context of this proof, Lem. 6 states that any attacker $\mathcal{A}$ who can distinguish the session key $\mathsf{G}(m^*, c^*)$ from uniform can be turned into an algorithm $\mathcal{B}$ that outputs m^*.

We now adapt this proof to our transformation $\mathsf{UC}_m^{\not\perp}$ by making the following changes: We simulate G via $\mathsf{G}(m, h, \mathsf{cd}) \leftarrow \mathsf{R}(\mathsf{Enc}(\mathsf{pk}, m))$ whenever $h = \mathsf{H}(\mathsf{pk})$ and $\mathsf{cd} = \mathsf{Enc}(\mathsf{pk}, m)$. This simulation fails in exactly the same cases as in the non-adapted proof and thus leads to the same analysis. After switching to the simulation, we can again apply double-sided One-way-to-Hiding (with our adapted function G) to find the challenge message m^* within $\mathcal{A}$'s random oracle queries. □

D.3 $\mathsf{FOC}_m^{\not\perp}$ is IND-CCA secure in the ROM

Our third theorem, Cor. 29 below, essentially recovers the ROM security bound for $\mathsf{FOC}_m^{\not\perp}$ that was given in [13, Sect. 2.1.4] for our augmented variant. Since $\mathsf{FOC}_m^{\not\perp} = \mathsf{UC}_m^{\not\perp} \circ \mathsf{TC}$, Cor. 29 is obtained by combining Thm. 26 with previous results about T that were given in [12].

Corollary 29 (ROM security of $\mathsf{FOC}_m^{\not\perp}$). *Let $\mathsf{PKE_C}$ be a probabilistic confirmation augmented PKE scheme, and assume PKE to be δ-worst-case correct. Let $\mathsf{H} : \mathcal{PK} \to \{0,1\}^\gamma$ be a fixed-output length function. Let $\mathsf{KEM_C} = \mathsf{FOC}_m^{\not\perp}[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}, \mathsf{G}']$.*

Let $\mathcal{A}$ be an adversary against $\mathsf{IND\text{-}CCA}$ security of $\mathsf{KEM_C}$, issuing at most q_D many queries to the decapsulation oracle for $\mathsf{KEM_C}$, and at most $q_\mathsf{G}/q_{\mathsf{G}'}/q_\mathsf{J}$ queries to the random oracles $\mathsf{G}/\mathsf{G}'/\mathsf{J}$.

According to Thm. 26, in the ROM there exists a $\mathsf{OW\text{-}CPA}$ adversary $\mathcal{B}$ against PKE such that

$$\mathrm{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM_C}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{T}[\mathsf{PKE},\mathsf{G}']}(\mathcal{B}) + (q_{\mathsf{G}'} + q_\mathsf{G} + 1) \cdot \delta + \frac{q_\mathsf{J}}{|\mathcal{M}|} \quad ,$$

$\mathcal{B}$ issues at most $q_{\mathsf{G}'} + q_\mathsf{G}$ many queries to oracle G'. According to [12], there thus exists a $\mathsf{OW\text{-}CPA}$ adversary $\mathcal{C}_O$ and an $\mathsf{IND\text{-}CPA}$ adversary $\mathcal{C}_I$ against $\mathsf{T}[\mathsf{PKE}, \mathsf{G}']$ such that

$$\mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{T}[\mathsf{PKE},\mathsf{G}']}(\mathcal{B}) \leq \begin{cases} (q_{\mathsf{G}'} + q_\mathsf{G} + 1) \cdot \mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{C}_O) \\ 3 \cdot \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{C}_I) + \frac{2q_{\mathsf{G}'} + q_\mathsf{G} + 1}{|\mathcal{M}|} \end{cases} \quad ,$$

The running time of $\mathcal{B}$, $\mathcal{C}_O$ and $\mathcal{C}_I$ is about that of $\mathcal{A}$.

D.4 $\mathsf{FOC}_m^{\not\perp}$ is IND-CCA secure in the QROM

Thm. 30 below essentially recovers the QROM security bound for $\mathsf{FO}_m^{\not\perp}$ that was implicitly given in [14] for our augmented variant.

Theorem 30. *Let $\mathsf{PKE_C}$ be a (randomized) augmented PKE scheme that is γ-spread, let $\mathsf{H} : \mathcal{PK} \to \{0,1\}^\gamma$ be a fixed-output length function, and let $\mathsf{KEM_C} = \mathsf{FOC}_m^{\not\perp}[\mathsf{PKE_C}, \mathsf{G}, \mathsf{H}, \mathsf{J}, \mathsf{G}']$, and let PKE denote the PKE scheme associated to $\mathsf{PKE_C}$. Let $\mathcal{A}$ be an $\mathsf{IND\text{-}CCA}$ adversary against $\mathsf{KEM_C}$, making at most q_D many queries to its decapsulation oracle, and making $q_{\mathsf{G}'}$, q_{G} queries to its respective random oracles. Furthermore, let $q = q_{\mathsf{G}'} + q_{\mathsf{G}}$, and let d and w be the query depth and query width of the combined queries to G' and G.*

Then there exist an $\mathsf{IND\text{-}CPA}$ adversary $\mathcal{B}_{\mathsf{IND}}$, a $\mathsf{OW\text{-}CPA}$ adversary $\mathcal{B}_{\mathsf{OW}}\text{-}\mathsf{CPA}$ against PKE and an $\mathsf{FFP\text{-}CPA}$ adversary $\mathcal{C}$ against $\mathsf{T}[\mathsf{PKE}, \mathsf{G}']$ in the extractable QROM with extractor function Enc such that

$$\mathrm{Adv}_{\mathsf{KEM_C}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}) \leq \mathrm{Adv}_{\mathsf{PKE}} + (q_D + 1) \cdot \mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{FFP\text{-}CCA}}(\mathcal{C}) + \varepsilon_\gamma, \text{ with} \tag{2}$$

$$\mathrm{Adv}_{\mathsf{PKE}} = \begin{cases} 4 \cdot \sqrt{(d + q_D) \cdot \mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(\mathcal{B}_{\mathsf{IND}})} + \frac{8(q+q_D)}{\sqrt{|\mathcal{M}|}} \\ 8\,(d + q_D) \cdot \sqrt{w \cdot \mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{OW\text{-}CPA}}(\mathcal{B}_{\mathsf{OW}}\text{-}\mathsf{CPA})}. \end{cases} \tag{3}$$

The additive error term ε_γ is given by

$$\varepsilon_\gamma = 24 q_D (q_{\mathsf{G}} + 4 q_D) 2^{-\gamma/2} \ .$$

Remark 31. Before discussing the proof, we note that [14] show how to bound $\mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{FFP\text{-}CCA}}(\mathcal{C})$ with a fine-grained analysis. Alternatively, [16] gives a simpler, albeit heuristic-prone bound by bounding

$$\mathrm{Adv}_{\mathsf{PKE}}^{\mathsf{FFP\text{-}CCA}}(\mathcal{C}) \leq 10(q + q_D + 1)^2 \cdot \delta \ ,$$

where δ is the correctness term of PKE as defined in Definition 1.

Proof. To summarize the proof for $\mathsf{FO}_m^{\not\perp}$ in [14], the proof proceeds in two steps with their own separate theorems:

1. The first step mutes the decapsulation oracle, i.e., it bounds the difference between $\mathsf{IND\text{-}CCA}$ and $\mathsf{IND\text{-}CPA}$ security of $\mathsf{FO}_m^{\not\perp}$.
2. The second step bases $\mathsf{IND\text{-}CPA}$ security of $\mathsf{FO}_m^{\not\perp}$ on passive security of PKE.

For the first step, [14] showed how to simulate the decapsulation oracle without the secret key. To that end, the randomness-generating random oracle G' is modeled as an extractable QRO. The extractable QRO provides not only the random oracle interface, but additionally an 'extraction' interface that (essentially) finds the right message for any queried ciphertext: the extraction interface can be queried on any ciphertext c and returns either $\perp$ or a originating message, i.e., a message m such that $\mathsf{Enc}(\mathsf{pk}, m; \mathsf{G}'(m)) = c$. Using the extraction interface, the simulated decapsulation oracle can (essentially) decrypt the ciphertext without the secret key, and then derive the corresponding session key like the original decapsulation oracle. This simulation fails in two cases: a) the attacker submits a ciphertext such that the extraction interface does not find its preimage, which is captured via γ-spreadness. b) the attacker submits a ciphertext for which the originating message and the decrypted message do not match, which is captured via the 'find failing plaintext' property $\mathsf{FFP\text{-}CCA}$.

The only differences in our setting are: instead of having an encryption algorithm Enc, we have a code-augmented encryption algorithm $\mathsf{Enc_C}$, and when deriving the key, we additionally hash the obtained confirmation code and the public-key identifier $\mathsf{H}(\mathsf{pk})$. We can easily integrate random oracle input $\mathsf{H}(\mathsf{pk})$ into that reasoning since pk is a public value. To integrate cd, we will use the same extraction interface as the proof for $\mathsf{FO}_m^{\not\perp}$. (We can define the 'pure encryption' algorithm Enc of our augmented scheme $\mathsf{PKE_C}$ as the algorithm that runs $\mathsf{Enc_C}$ and drops the confirmation code.) With that, our simulation of the decapsulation oracle can again obtain the originating plaintext

m of any queried ciphertext c. The simulation can then also acquire the associated confirmation code since $\mathsf{FOC}_m^{\not\perp}$ derandomizes the algorithm $\mathsf{Enc_C}$: the simulation can simply compute $(c', \mathsf{cd}) := \mathsf{Enc_C}(\mathsf{pk}, m; \mathsf{G}'(m))$. (One might think that this costs one more call to G', but this call is anyways being done to be able to perform the re-encryption check.)

The second step simply argues that the resulting key is indistinguishable from random due to One-Way To Hiding (in the extractable QROM). Since we did not change the extractor function, this part can remain unchanged, except that we append $\mathsf{H}(\mathsf{pk})$ and cd to the inputs of the random oracle G. $\square$