# Lattice-Based Updatable Public-Key Encryption for Group Messaging

Joël Alwen[1], Georg Fuchsbauer[2], Marta Mularczyk[1], and Doreen Riepel[3]

[1] AWS Wickr, {alwenjo,mulmarta}@amazon.com
[2] TU Wien, first.last@tuwien.ac.at
[3] CISPA Helmholtz Center for Information Security, riepel@cispa.de

**Abstract.** *Updatable Public-Key Encryption* (UPKE) augments the security of PKE with Forward Secrecy properties. While requiring more coordination between parties, UPKE enables much more efficient constructions than full-fledged *Forward-Secret PKE*. Alwen, Fuchsbauer and Mularczyk (AFM, Eurocrypt'24) presented the strongest security notion to date. It is the first to meet the needs of UPKE's most important applications: Secure Group Messaging and Continuous Group Key Agreement. The authors provide a very efficient construction meeting their notion with classic security based on the Computational Diffie-Hellman (CDH) assumption in the Random Oracle Model (ROM).

In this work we present the first post-quantum secure UPKE construction meeting (a slight relaxation of) the AFM security notion. Based on the Module LWE assumption, our construction is practically efficient. Moreover, public key sizes are about 1/2 and ciphertext sizes around 2/3 of those of the state-of-the-art lattice-based UPKE scheme in the ROM by Abou Haidar, Passelègue and Stehlé – despite only being shown to satisfy a significantly weaker security notion. As the AFM proofs relies on random self-reducibility of CDH, which has no analogue for lattices, we develop a new proof technique for strong UPKE, identifying the core properties required from the underlying (lattice-based) encryption scheme.

# Table of Contents

## 1 Introduction

In light of the never ending parade of real-world security breaches, Forward Secrecy (FS) has emerged as a critical security property. In the context of secure communication, FS limits the security loss future corruptions can have on past communication. FS is especially important for long-lasting cryptographic applications such as secure (group) messaging, where a single protocol session can last for many years and involve a large number of participating devices. For example, FS is a core design goal for protocols such as the Double Ratchet [PM16] and the Messaging Layer Security (MLS) protocol [BBR+23], both used in practice today.

For Public-Key Encryption (PKE), FS boils down to maintaining confidentiality (e.g. IND-CCA security) for previously decrypted ciphertexts in the event that a decryption key is leaked. As demonstrated in [ACDT20, ACJM20, ACDT21], the FS of the MLS and related multi-party protocols can be substantially improved by replacing their use of PKE with a forward secret variant.

Classic PKE uses a fixed decryption key. Leaking it at any point in time allows an adversary to decrypt all past and future ciphertexts. To avoid this, [CHK03] introduced *Forward-Secure Public-Key Encryption* (FS-PKE), which augments traditional PKE security by adding FS guarantees. Intuitively, with FS-PKE, decrypting a ciphertext $C$ also punctures the decryption key removing its ability to ever decrypt $C$ again.

Unfortunately, to date, the efficiency of FS-PKE constructions remains untenable for most practical applications. Thus, [JMM19] introduced *Updatable Public-Key Encryption (UPKE)*, which provides similar FS guarantees as FS-PKE (also using evolving decryption keys). However, to allow for improved efficiency, UPKE requires more coordination between sending and receiving parties than needed for PKE and FS-PKE. Specifically, in addition to decryption evolving a secret key $sk_i$ into some $sk_{i+1}$, a UPKE encryption to public key $pk_i$ also produces a correspondingly updated public key $pk_{i+1}$. With this in mind, correctness for UPKE (i.e. the guarantee that honest parties can decrypt each other's ciphertexts) is only guaranteed when encryption and decryption keys remain in sync with each other. For example, Alice encrypting to Bob's UPKE public key $pk_i$ has the side effect of updating $pk_i$ to $pk_{i+1}$. So, once Bob decrypts the ciphertext from Alice, he will update his secret key from $sk_i$ to $sk_{i+1}$, which means that Charlie will first have to obtain $pk_{i+1}$ before he can encrypt to Bob.

Fortunately, for multi-party protocols such as MLS and its relatives, the coordination required to guarantee correctness is relatively easy to provide when using a reliable network. This makes UPKE an attractive primitive for improving the FS of those applications.

**Updatable Key Encapsulation Mechanisms.** As in prior work [APS23, AFM24], the technical focus of our work is actually *Updatable Key Encapsulation Mechanisms* (UKEM) rather than UPKE. A standard KEM/DEM construction transforms any UKEM into a UPKE with essentially the same security and efficiency. (To port a UKEM definition to UPKE, we simply replace the real/random key challenge oracle with a chosen-message challenge oracle.)

In fact, real-world UPKE applications (such as MLS) are already formulated as using a KEM (not PKE). Of course, under the hood, they then use the KEM in a standard KEM/DEM construction to realize PKE. Thus, although [ACDT20] speak of replacing PKE with UPKE, to improve MLS's FS it would be more in line with the original protocol description to speak of replacing the KEM with a UKEM. Ultimately, the effect is exactly the same. So, for the remainder of this work, we opt to stick with the slightly simpler UKEM primitive.

**UKEM security.** In this work, with an eye towards the multi-party protocol uses of UKEM, we target the UKEM security notion of [AFM24]. A key observation motivating that notion is that an adversary controlling the network used by a multi-party protocol can actually (quite easily) desynchronize honest parties to the point where UKEM correctness might fail. For example, by simply selectively forwarding or re-ordering protocol messages for different group members. Fortunately, subject to such network behavior the multi-party protocols usually forgo availability requirements, so possible correctness failures for the underlying UKEM scheme do not automatically present a problem.

However, even for the more challenging case of fully adversarial networks, the protocols *do* still require strong (especially FS) security properties. Yet, no prior UKEM (nor UPKE) security notion had adequately captured the types of desynchronized UKEM executions implicit in such attacks on the higher-level multi-party protocols. Thus, it remained unclear as to if/how UPKE (and UKEM) might be of use for such applications. This led the authors of [AFM24] to introduce a significantly stronger UKEM security notion than prior works capturing the full generality of attacks UKEM must resist for such use cases.

*Our UKEM security vs [AFM24].* The UKEM security notion used in this work is identical to the one in [AFM24] (e.g. capturing the full generality of desychnronization for use in multi-party protocols) except for two caveats. First, the notion of [AFM24] implies two types of UKEM security called "joiner" and "member" security. As all prior works except [AFM24], we only consider the latter and leave constructing UKEM schemes with PQ joiner security as an open problem. We stress that for most applications, member security is crucial while joiner security provides additional guarantees in edge cases (to parties joining a fake group created by the adversary). It is also difficult to achieve; the work [AFM24] uses special (efficient) malleable non-interactive zero knowledge proofs (NIZKs) with a security proof directly in the Algebraic Group Model [FKL18]. Second, our notion is slightly weaker in that it allows to challenge fewer generations of a key pair. However, we argue that this is irrelevant for applications; see Section 4.3 for a detailed discussion.

**UPKE/UKEM constructions.** The first explicit UPKE construction was introduced in [JMM19], although prior implicit constructions can already be found in [JS18, PR18]. These were soon followed by many more (explicit and implicit) constructions including [ACDT20, ALP22, AFM24], as well as constructions [JS18, PR18, EJKM22, DKW21, AW23, APS23, ACJM20, AMT23] satisfying post-quantum (PQ) security. In particular, [EJKM22] is isogeny-based while both [DKW21, APS23] are lattice-based. The remaining PQ constructions rely exclusively on generic primitives with known post-quantum secure instantiations.

Of all these constructions, only [AFM24] is shown to provide sufficient security for high-level multi-party protocol applications while the rest are analyzed w.r.t. significantly weaker notions. In this work we propose a construction with appropriate security, which is most closely related to the one in [APS23] (albeit with significant differences described below). Like [APS23], we rely on the NIZK techniques in [LNP22].

**Practical efficiency.** From the perspective of *applied* cryptography, an overarching goal in the study of UPKE is to realize the promised FS gains of replacing PKE with UPKE in MLS *in the wild*.[4] We view the results in this work as making substantial progress towards this goal. Indeed, demonstrating a practically efficient PQ UPKE scheme sufficiently strong

[4] MLS is already in use in several applications including Cisco's Webex and Discord, with many more to come. For example, Google intends to deploy MLS to over a billion mobile devices as the E2EE layer for securing all RCS text messages.

for MLS was probably the single biggest hurdle till today on the path to more FS for MLS, even just for classic security!

To really use UPKE in MLS widely in practice, we will need a UPKE (or UKEM) standard, usually via the IETF. Indeed, MLS and every primitive in every one of its standardized ciphersuites are all IETF standards. Nor would hardcoding a concrete UPKE/UKEM construction in MLS directly be a viable approach as MLS (like TLS, SSH, IPSSEC, Wireguard, etc.) is designed to work with a generic ciphersuite to be negotiated at runtime by particpants. Fortunately, for the classic case, the practical efficiency of the UKEM in [AFM24] makes real progress towards laying the foundations required to build such a standard.

However, with advent of the post-quantum era, it's problematic (at best) to establish standards for new primitives (e.g. UPKE) or extend existing standards (e.g. switching PKE to UPKE in MLS) without a clear path to PQ security in mind. Thus, the lack of a plausible PQ alternative to UPKE for MLS presents a barrier preventing not just better PQ FS for MLS, but even just improving classic FS for MLS. In light of this, although we nominally consider only PQ security, this work takes a big step forward for the wider goals of work on UPKE.

### 1.1 Technical Overview

We propose a new PQ-secure UKEM scheme, which improves upon the state-of-the-art PQ-secure UKEM of [APS23] in two ways:

1. It enjoys a proof for a stronger security needed for real-world applications.
2. It is more efficient in terms of ciphertext and key size and computation cost.

We next outline our construction and then explain how our design and modularization allows us to achieve the above contributions.

**Our construction.** At a high level, we construct our UKEM scheme generically from a primitive called *key-homomorphic (kh-)PKE* (and other standard primitives such as PKE). In kh-PKE, secret and public key spaces are (abelian) groups with a homomorphism $\mathsf{pk} : sk \mapsto pk$. The concept can be efficiently instantiated by adapting standard IND-CPA secure PKE schemes, such as ElGamal and most importantly the IND-CPA secure PKE of Kyber [BDK+18].

We introduce a security notion for kh-PKE that we call Shifty-IND-CPA security. It guarantees confidentiality of messages encrypted to a public key $pk + \overline{pk}$ where $pk$ is honestly generated and $\overline{pk}$, the *shift*, corresponds to an $\overline{sk}$ chosen by the adversary.[5]

A public key in our UKEM scheme contains a public key $pk_i$ for a kh-PKE scheme. The encapsulation algorithm encrypts a random message $m$ to $pk_i$. To get IND-CCA security, we use the Fujisaki-Okamoto [FO99] transform and derive the encryption randomness and the encapsulated key from a hash of $m$. In order to update the public key with no additional overhead, we also derive a shift $\overline{pk}$ and the corresponding $\overline{sk}$ from another hash of $m$. The updated public key is then $pk_{i+1} := pk_i + \overline{pk}$.

The above construction does not yet have *public verifiability*. This means that there is no way for parties not knowing $sk_i$ to verify that $pk_{i+1}$ can be trusted. (An adversary could have freshly generated $pk_{i+1}$ and would thus know $sk_{i+1}$.) To achieve this, we add to

[5] Note that e.g. for ElGamal, standard security implies shifty security: given an (additively denoted) ciphertext $\mathbf{c} = (\rho \cdot g, \rho \cdot pk + m)$ and a secret key $\overline{sk}$, $\mathbf{c}$ shifted to $pk + \overline{pk}$ is $(c_1, c_2 + \overline{sk} \cdot c_1)$; for LWE-based schemes, this is in general not the case; see below.

Table 1: Comparison of public key, ciphertexts and member-tag sizes between this work and [APS23].

(a) Comparison for concrete parameters, for at most $2^{20}$ updates and 137-bit security. See details and more parameter sets in Section 7.

| | public key | ciphertext with member-tag |
|---|---|---|
| [this work] | 6.9 KB | 86.3 KB |
| [APS23] | 13.9 KB | 129.6 KB |

(b) Generic comparison depending on the following: the security parameter $\kappa$, and the sizes of a public key $pk$ and ciphertext $c$ (encrypting $\kappa$ bits) for the kh-PKE scheme, a public key $pk_{crs}$ and a ciphertext $c_{crs}$ (encrypting a kh-PKE secret key) for the PKE used in the NIZKPoK construction and the proof $\pi$ for the NIZK used in the NIZKPoK.

| | public key[6] | ciphertext with member-tag |
|---|---|---|
| [this work] | $\lvert pk\rvert + \kappa$ | $\lvert c\rvert + \lvert c_{crs}\rvert + \lvert\pi\rvert$ |
| [APS23] | $\lvert pk\rvert + \lvert pk_{crs}\rvert$ | $\lvert c\rvert + 2\cdot\lvert c_{crs}\rvert + \lvert\pi_{\text{APS}}\rvert$ |

(c) Statement proved in $\pi$ and $\pi_{\text{APS}}$ as a proxy for its size.

| | NIZK statement for $\pi$ and $\pi_{\text{APS}}$ |
|---|---|
| $\pi$ [this work] | $(pk_i, pk_{i+1}, pk_{crs}, c_{crs})$ s.t. $c_{crs}$ encrypts $sk$ to $pk_{crs}$ and $\mathsf{pk}(sk) = pk_{i+1} - pk_i$ |
| $\pi_{\text{APS}}$ [APS23] | $(pk_i, pk_{i+1}, pk_{crs}, c_{crs}, c'_{crs})$ s.t. "$c_{crs}$ encrypts $sk$ to $pk_{crs}$ and $c'_{crs}$ encrypts $sk$ to $pk_{crs}$ and $\mathsf{pk}(sk) = pk_{i+1} - pk_i$ |

a ciphertext a non-interactive zero-knowledge proof of knowledge (NIZKPoK) of $\overline{sk}$ such that

$$\mathsf{pk}(\overline{sk}) = pk_{i+1} - pk_i \ . \tag{1}$$

In the UKEM syntax of [AFM24], such a proof is called a *member-tag*. To achieve the strong security we aim for, the NIZKPoK needs to satisfy *straightline simulation extractability*. We construct such a NIZKPoK generically from a (simulation-sound) NIZK (not PoK) and a (IND-CPA secure) PKE. The NIZKPoK's common reference string CRS (which will be part of the UKEM public key, sampled fresh together with the key pair) is a PKE public key $pk_{crs}$ and potentially a CRS for the used NIZK. A proof of knowledge of a shift $\overline{sk}$ consists of (i) an encryption $c$ of $\overline{sk}$ to $pk_{crs}$ and (ii) a NIZK proof that $c$ encrypts an $\overline{sk}$ satisfying (1). We prove that one can define a straight-line extractor that extracts $\overline{sk}$ by decrypting $c$. We then optimize the above NIZKPoK construction so that $pk_{crs}$ is replaced by a *random* string; see below.

**More efficient instantiation from LWE.** We build on the construction of Abou Haidar, Passelègue and Stehlé (APS) [APS23]. In particular, we use their basic (IND-CPA-only) lattice-based UPKE (without the updating functionality) as an instantiation of kh-PKE. Their scheme is based on [LPS10] and Kyber [BDK+18]. Compared to Kyber, it requires larger parameters in order to guarantee correctness after several key updates. (Every time a public key $pk_i$ is updated to $pk_{i+1} := pk_i + \overline{pk}$, the error increases, which for standard Kyber parameters would make decryption impossible after a number of updates.)

[6] Public key contains a Kyber public key $(\mathbf{A}, \mathbf{b})$ and seed $s$. In practice, $\mathbf{A}$ is derived by hashing a random seed, which can also be used to derive $s$, reducing size to $|pk|$.

We instantiate the standard PKE used within the NIZKPoK with the *standard* IND-CPA secure scheme of Kyber and use the efficient lattice-based NIZK from [LNP22] that has been argued to be (adaptively) simulation-sound in [BBP23, Lemma 2].

We next sketch how we improve upon the efficiency of APS, as outlined in Table 1.

Our first insight allows us to cut the public key size roughly in half. Since the secret key $sk_{crs}$ is not used in the protocol, the corresponding $pk_{crs}$ can be "fake"; that is, our key generation samples a short seed $s$ and derives $pk_{crs}$ by hashing $s$. Thus, $pk_{crs}$ is a random matrix not necessarily in the support of Kyber key generation; however, modeling the hash as a random oracle allows us to embed a real key in $pk_{crs}$. This can be done since, under LWE, Kyber public keys look like random matrices.

Further, in APS the shift $\overline{sk}$ is freshly sampled and must be encrypted under $pk_i$, so the holder of the corresponding $sk_i$ can update $sk_i$ to $sk_{i+1} := sk_i + \overline{sk}$. We avoid this by deriving $\overline{sk}$ from the hash of the encrypted random message. Thus, a ciphertext in our scheme consists of two Kyber ciphertexts: $c$ encrypting the random message $m$ to $pk_i$, and $c_{crs}$ encrypting $\overline{sk}$ to $pk_{crs}$ (implicit in our proof of knowledge). In contrast, a ciphertext in APS contains an additional Kyber ciphertext $c_{up}$ encrypting $\overline{sk}$ to $pk_i$.

Another source of improvement efficiency is the simpler NIZK statement compared to APS. The NIZK is by far the largest (and most expensive to compute) part of a ciphertext-tag pair, and a simpler statement results in smaller NIZK proofs. In particular, our statement involves only one encryption of the secret key shift instead of two as in APS; see Table 1. Note that an encryption of a secret key is larger than an encryption of a short random message.

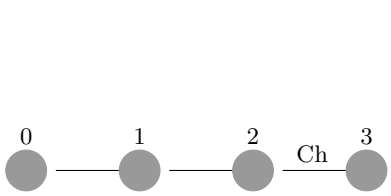
Finally, in contrast to APS, for the CRS we use a PKE scheme different from the kh-PKE used for updates. Like in APS the modulus $q$ for the kh-PKE needs to be increase so correct decryption is still guaranteed for (many-times) updated keys (even with adversarial shifts). This increase of key/ciphertext sizes and is not needed for the PKE. Thus, we can use a more efficient PKE as long as it is compatible with the NIZK.

**Strong security.** At a high level, we can visualize the evolution of a UKEM key pair as a path of nodes containing evolutions of a key pair, $(pk_0, sk_0)$, $(pk_1, sk_1)$, $(pk_2, sk_2), \ldots$ Here $(pk_0, sk_0)$ is a fresh key pair, $(pk_1, sk_1)$ is a result of updating $(pk_0, sk_0)$ and $(pk_2, sk_2)$ is a result of updating $(pk_1, sk_1)$, etc.

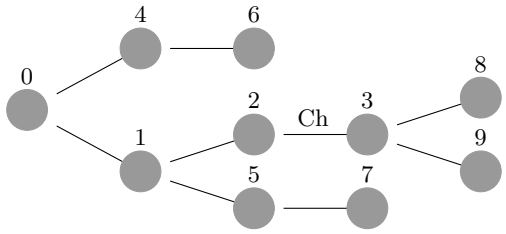
Following [AFM24], we observe that to model *asynchronous* executions of the application using UKEM (e.g. MLS), we need to consider not paths but more general (directed) *trees*, see Fig. 1. The root of such a tree contains a fresh key pair $(pk_0, sk_0)$. A node with two children represents the result of a network split in the execution of an application using the UKEM. For example, due to a network split, Alice may update $pk_0$ to $pk_1$ and Bob may update $pk_0$ to $pk'_1$. This is represented as the root node 0 having two children, one with $pk_1$ and the other with $pk'_1$.

All prior works on Kyber-based UKEMs [DKW21, APS23] restrict UKEM executions to paths. Proving security of a Kyber-based UKEM in the setting with arbitrary trees requires new techniques both for protocol design and the security proof.

*UKEM security game.* We use (a small variation of) the UKEM security notion of [AFM24] which builds upon the above tree intuition. At a high level, a security experiment is defined with the challenger and the adversary. The challenger generates a fresh root key pair $(pk_0, sk_0)$ and the adversary, given $pk_0$, drives the key pair's evolution. That is, the challenger creates a tree with "honest" and "adversarial" edges. The adversary decides to create an honest or adversarial edge using Enc and Dec oracles, respectively. For an honest edge, the key pair of the child node is created by the challenger. For an adversarial edge,




(a) A UKEM execution from the notion of [DHW23, APS23]. The adversary corrupts only node 3. After the challenge query, it can no longer query the Dec oracle for nodes 0-2.




(b) A UKEM execution from the notion of [AFM24]. Adversary defines the structure of the tree and can corrupt any nodes that do not result in a trivial win, e.g. 6, 3, 5. It can also access the Dec oracle for any nodes at any time.

Fig. 1: Illustrations of UKEM executions in the security notion of [DHW23, APS23] and the stronger one of [AFM24]. "Ch" marks the edge created by challenge query. The adversary's goal is to distinguish the key outputted by the encapsulation call creating this edge from a random key.

the key pair is created by the adversary: it provides an arbitrary updated public key, a (valid) member tag and a ciphertext (which updates the secret key or not, if invalid). The adversary can also corrupt arbitrary nodes to get updated secret keys (modelling FS) and see the keys decapsulated by the challenger when creating adversarial edges (modelling CCA). At some point, the adversary challenges a node, i.e., it gets a ciphertext and either the real or a random key. It wins by guessing which is the case. (The challenge also creates an honest edge.)

*Reduction to Shifty-IND-CPA.* Consider the line-execution in Fig. 1a as an example for an execution of the weaker security game in [DKW21, APS23]. The adversary makes two updates to the root key $pk_0$, leading to $pk_2$, which it then *challenges*, that is, it is given an encapsulation and either the encapsulated or a random key and has to decide which. $pk_2$ is then updated by the experiment to $pk_3$ and the adversary receives ("corrupts") the corresponding $sk_3$.

We want to show that Shifty-IND-CPA of the kh-PKE scheme implies confidentiality of the key $K$ encapsulated to the public key $pk_2$ in node 2. The reduction embeds the challenge public key from its Shifty challenger as $pk_0$. It can then request a challenge ciphertext encrypted to $pk_2$, as long as it provides the "shift" secret key $sk_2 - sk_0$ (which we can extract from the proofs the adversary must provide). The Shifty-IND-CPA notion moreover provides a key $sk'$, which is an honest update of the challenged secret key, that is, $sk_2$, in the example. The reduction can thus give the adversary $sk_3 := sk'$.

This proof strategy is similar to [APS23], which instead of using the abstraction of Shifty-IND-CPA, reduces security directly to their variant of LWE. However, since their scheme (unlike ours) includes an encryption of the update secret key, they need to argue that this does not leak information.

Unfortunately, the sketched strategy does not extend to the security notion of [AFM24], where the adversary can corrupt other keys (which would not lead to a trivial break) in the scheme. Considering Fig. 1b, the adversary can corrupt any nodes outside the challenge path 0-1-2, for example node 4. The reduction to Shifty security thus has to provide $sk_4$. So, when the adversary asks to create node 4, the reduction could ask for another challenge (under $pk_0$) to get a key $sk_4$ which is $sk_0$ shifted by an unknown $\overline{sk}_4$. However, what if the

adversary then *challenges* node 4 instead of node 2 (after creating nodes 1-4 via updates)? Then the reduction fail as it can not longer embed a Shifty challenge ciphertext in $pk_4$ (for which it knows the secret key). So to adjust the previous reduction strategy would require guessing the challenge path, which incurs an exponential loss.

To deal with this, [AFM24] uses a more clever strategy. In their scheme $sk_2$ is derived from the hash $H(m_2)$, where $m_2$ is the random message encrypted in $c_2$. In addition to improving efficiency, this also enables a different proof strategy that avoids a sequence of hybrids overall, when modeling $H$ as a random oracle. The reduction to (Shifty-)IND-CPA embeds the challenge public key directly as $pk_2$ (instead of $pk_0$). In the random oracle model, this is fine since $c_2$ is independent of $sk_2$ as long as the adversary does not query $H$ on $m_2$. On the other hand, if it makes this query, then it has "broken" $c_2$, and the reduction could break Shifty security for $pk_1$ (if it had embedded the challenge key there). The reduction thus guesses which public key will be broken first and embeds its challenge there. This only incurs a linear security loss, while allowing the adversary to adaptively corrupt keys.

Unfortunately, this strategy does not work with LWE-based encryption schemes such as Kyber. The reason is that, unlike with ElGamal used in [AFM24], $pk_2$ is not distributed like a fresh PKE key. In particular, say the edge 0-1 is created when the adversary injects an update $\overline{pk}_1$, and the edge 1-2 is created by the game that generates a fresh update $\overline{pk}_2$. Then $pk_2 = pk_0 + \overline{pk}_1 + \overline{pk}_2$. With ElGamal, $+$ is an isomorphism so this is the same as sampling $pk_2$ at random and setting $\overline{pk}_2 = pk_2 - pk_0 - \overline{pk}_1$. This is not the case for Kyber, where $pk_2$ comes from the LWE distribution shifted by an honest $\overline{pk}_2$ and an *adversarial* $\overline{pk}_1$.

We therefore devise a different proof strategy: instead of embedding the challenge in $pk_2$, we embed it in the *update* $\overline{pk}_2$. We then use our new "Shifty" variant of IND-CPA to "shift" (challenge) ciphertexts encrypted to $\overline{pk}_2$ to ones encrypted to $pk_2$. This way our reduction can embed the challenge.

*Security of NIZKPoK.* Recall that when updating a key $pk$ to $pk' = pk + \overline{pk}$, a *member tag* $mt$ is generated, which guarantees that if $pk$ was "secure" then so is $pk'$ – even to users that do not know the secret keys. The tag $mt$ proves knowledge of $\overline{sk}$ corresponding to $\overline{pk}$ by encrypting $\overline{sk}$ to $pk_{crs}$ and including a NIZK for correctness.

Proving the stronger security notion abstracted by trees (cf. Fig. 1b) for LWE-based schemes requires a stronger security for the proof of knowledge (PoK) than what was necessary for previously considered UKEM security notions. For these, regular soundness suffices, as the graph constructed by the adversary can only be a line (cf. Fig. 1a) and thus the adversary can only corrupt the key (node 3 in the example) following the challenged node. The security reduction extracts the $\overline{sk}$'s corresponding to edges created by the adversary (e.g. 0-1 and 1-2), so it can *shift* a challenge ciphertext for $pk_0$ to $pk_2$. Only *after* the extraction, it needs to simulate the proof for this last edge (2-3 in the example).[7]

To prove that our scheme satisfies the stronger security from [AFM24], we require $mt$ to be *straight-line simulation-extractable*, which is achieved by using a simulation-sound NIZK (while for the weaker security notion in [APS23], soundness would suffice). This is necessary because our reduction to Shifty-IND-CPA has to extract from $mt$'s continuously

[7] This is similar to the Naor-Yung construction [NY90] for IND-CCA1 security, for which a sound NIZK is sufficient. We note that the security notion in [APS23] also only guarantees (and their scheme only satisfies) IND-CCA1 security (for what in [APS23] is called the update ciphertext) whereas the stronger security [AFM24] we consider allows further decryption queries, thus corresponding to IND-CCA2 security.

during the simulation. Specifically, consider the reduction for the scenario in Fig. 1b, where a challenge query for node 2 creates node 3.

Further, say node 1 is created honestly and node 2 is created by the adversary injecting $pk_2$ and $mt_2$. This means that the reduction embeds its challenge key as the shift $\overline{pk}_1$ corresponding to the edge 0-1. When node 1 was created, the reduction, not knowing $\overline{sk}_1$, had to simulate the PoK $mt_1$. Then, when the adversary asks for the challenge encrypted to $pk_2$, the reduction has to get a Shifty-IND-CPA challenge $c^*$ encrypted to $\overline{pk}_1$ shifted to $pk_2 = \overline{pk}_1 + (\overline{pk}_2 + pk_0)$. Getting such a challenge requires extracting $\overline{sk}_2$ from $mt_2$. Moreover, the proof for the tag $mt_3$ attached to $c^*$ needs to again be simulated.

Finally, we note that NIZK used in the scheme of [AFM24] only requires simulation extractability, which need not be straight-line. Their reduction to computational Diffie-Hellman does not need to extract during the simulation. Instead, it only extracts at the very end of the experiment in order to "shift" its CDH solution.

## 2 Preliminaries

**Notation.** For integers $n$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. If $X$ is a finite set, then $x \leftarrow\!\!\$\, X$ denotes picking $x$ uniformly at random from $X$. $y \leftarrow \mathsf{A}(x_1, x_2, \ldots)$ denotes that on input $x_1, x_2, \ldots$, the (possibly probabilistic) algorithm $\mathsf{A}$ returns $y$. $\mathsf{A}^{\mathsf{O}}$ denotes that $\mathsf{A}$ has access to an oracle $\mathsf{O}$.

**Failures.** We assume that an algorithm may "fail", meaning it outputs a special failure symbol $\perp$; if it has multiple outputs, all of them are set to $\perp$. The probability of this happening is bounded by the respective correctness property. Further, we assume that whenever an algorithm gets $\perp$ as any of its inputs, the output(s) will be $\perp$ as well. Some algorithms or game procedures output a bit determined by a Boolean statement B. Here, the notation (B) denotes the output 1 if B is true and 0 otherwise.

### 2.1 Public-Key Encryption

We generalize PKE slightly, as required by our constructions of key-homomorphic PKE and NIZKPoK. In addition to a secret key space $\overline{\mathcal{SK}}$ and a randomness space $\overline{\mathcal{RS}}$, PKE setup also defines subspaces thereof: $\mathcal{SK}$ are "good" secret keys for which decryption succeeds, and $\mathcal{RS}$ are "good" randomness values for which encryption succeeds. For example, for LWE-based schemes, $\overline{\mathcal{SK}}$ may be a ring $\mathbb{Z}_q^n$ and $\mathcal{SK}$ a set of short vectors in $\mathbb{Z}_q^n$. Formally, PKE is described the following PPT algorithms:

*Setup.* $pp \leftarrow \mathsf{PKE.Setup}$ outputs public parameters $pp$, which also define:
- secret and public key spaces, respectively, $\overline{\mathcal{SK}}$ and $\mathcal{PK}$,
- a set $\mathcal{SK} \subseteq \overline{\mathcal{SK}}$ of secret keys outputted by key generation,
- a message space $\mathcal{M}$ and a randomness space $\overline{\mathcal{RS}}$,
- a set $\mathcal{RS} \subseteq \overline{\mathcal{RS}}$ of "good" randomness values.

*Key Generation.* $(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(pp)$, on input the public parameters, outputs a public key $pk \in \mathcal{PK}$ and a secret key $sk \in \mathcal{SK}$.

*Encryption.* $c \leftarrow \mathsf{PKE.Encrypt}(pp, pk, m)$, on input a public key $pk \in \mathcal{PK}$ and a message, outputs a ciphertext $c$.

*Decryption.* $m \leftarrow \mathsf{PKE.Decrypt}(pp, sk, c)$, on input a secret key $sk \in \overline{\mathcal{SK}}$ and a ciphertext $c$, outputs a message $m$.

**Correctness.** Using PKE for our proof system requires correct decryption whenever encryption uses randomness from $r \in \mathcal{RS}$. A PKE is $\delta$-correct if for all $pp \in \mathsf{PKE.Setup}$ we have:

- $\Pr[(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(pp) : sk \notin \mathcal{SK}] \leq \delta$;
- for all $m \in \mathcal{M}$ and $pk \in \mathcal{PK}$: $\Pr[c \leftarrow \mathsf{PKE.Encrypt}(pp, pk, m) : c = \perp] \leq \delta$; and
- for all $m \in \mathcal{M}$, $r \in \mathcal{RS}$ and $(pk, sk)$ output by $\mathsf{PKE.KeyGen}(pp)$: if $sk \in \mathcal{SK}$ and $\mathsf{Encrypt}(pp, pk, m; r) \neq \perp$ then $\mathsf{Decrypt}(pp, sk, c) = m$.

### 2.2 Proof Systems

**Non-Interactive Proof System (NIPS).** A *Non-Interactive Proof System* (NIPS) for an NP-relation $\mathcal{R}$ in the random oracle model is a triple of algorithms, all with access to a random oracle $H$:

*Setup.* $crs \leftarrow \mathsf{Setup}^H$ outputs a common reference string $crs$.
*Prove.* $\pi \leftarrow \mathsf{P}^H(crs, x, w)$ on input a statement $x$ and a witness $w$, outputs a proof $\pi$ that $(x, w) \in \mathcal{R}$.
*Verification.* $0/1 \leftarrow \mathsf{V}^H(crs, x, \pi)$ verifies a proof $\pi$ of statement $x$, i.e., that $(x, w) \in \mathcal{R}$ for some $w$.

**Non-Interactive Zero-Knowledge proofs (NIZK).** We consider NIPS that are zero-knowledge and (adaptive) simulation-sound (formally defined in Appendix B). *Zero knowledge* means that there exists a simulator that can produce proofs of (true) statements without knowing the witness (but being able to program the random oracle) that are indistinguishable of outputs of $\mathsf{P}$. Adaptive Soundness means that no adversary, after being given the CRS, can produce a proof for a false statement. *Simulation soundness* means that soundness even holds against adversaries that can query simulated proofs of (possibly false) statements.

**NIZK Proof of Knowledge (NIZKPoK).** We define NIZKPoKs as NIPS that are zero-knowledge and straight-line-extractable (formally defined in Appendix B). *Straight-line simulation extractability* requires two additional algorithms: first, an *alternative setup* that outputs an "extraction trapdoor" together with a CRS that is indistinguishable from the output of $\mathsf{Setup}$; second, an *extractor* that, given the extraction trapdoor, can extract witnesses from proofs generated by an adversary with access to a simulation oracle.

## 3 Key-Homomorphic PKE

The main building block for our UKEM construction is a *key-homomorphic public-key encryption* (kh-PKE) scheme. A hk-PKE scheme is a PKE scheme (as defined in Section 2.1) that satisfies the following definition.

**Definition 1.** *A PKE is* key-homomorphic *if* $(\overline{\mathcal{SK}}, +)$ *and* $(\mathcal{PK}, +)$ *are abelian groups and there is a homomorphism* $\mathsf{pk} : \overline{\mathcal{SK}} \to \mathcal{PK}$.

**Correctness.** Let $\ell > 0$ be an integer and $\delta > 0$. A key-homomorphic $\mathsf{PKE}$ is $(\ell, \delta)$-correct if for all $pp \in \mathsf{Setup}$ the following hold:

- $\Pr[(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(pp) : sk \notin \mathcal{SK}] < \delta$.
- For all $sk_0, sk_1, \ldots, sk_\ell \in \mathcal{SK}$ and $m \in \mathcal{M}$:

$$\Pr\left[\mathsf{Decrypt}(pp, sk, \mathsf{Encrypt}(pp, \mathsf{pk}(sk_0 + sk_1 + \cdots + sk_\ell), m)) \neq m\right] < \delta \ ,$$

  where the probability is taken over the random coins of $\mathsf{Encrypt}$.

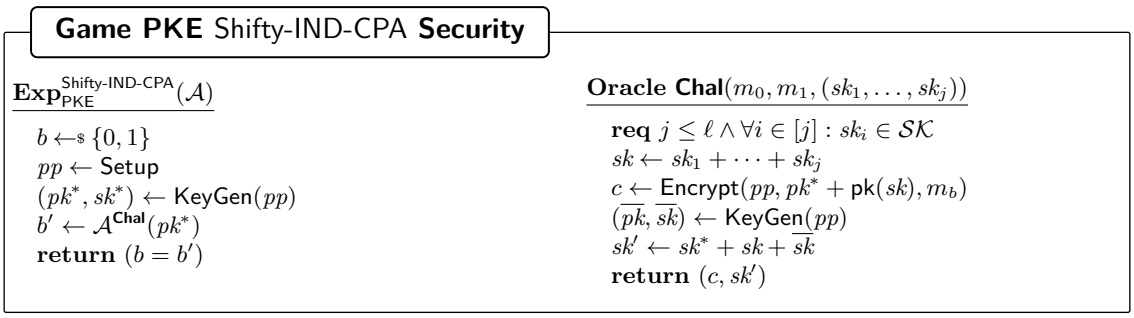



Fig. 2: The Shifty-IND-CPA security game for homomorphic PKE PKE = (Setup, KeyGen, Encrypt, Decrypt), an integer $\ell$ and adversary $\mathcal{A}$.

**Security.** We define a security notion for kh-PKE required for its use in our UKEM construction. In UKEM security, the adversary will be able to choose update secret keys, so the challenge public key might not be distributed according to the PKE key generation, but "shifted". This is not the case for ElGamal encryption, but it is for lattice-based schemes. Therefore, our notion Shifty-IND-CPA extends standard CPA security to reflect this.

**Definition 2** (PKE Shifty-IND-CPA)**.** *Let $\ell$ be a positive integer. Let game* Shifty-IND-CPA *for* PKE *be as defined in Fig. 2. The advantage of an adversary $\mathcal{A}$ in that game is defined as*

$$\mathsf{Adv}^{\text{Shifty-IND-CPA}}_{\text{PKE}}(\mathcal{A}) := 2\Pr\left[\mathrm{Exp}^{\text{Shifty-IND-CPA}}_{\text{PKE}}(\mathcal{A}) = 1\right] - 1.$$

## 4 Updatable KEM

### 4.1 Functionality

An *updatable key encapsulation mechanism* (UKEM) is a collection of the following PPT algorithms (where we use the "short syntax" from [AFM24]):

*Setup.* $pp \leftarrow \mathsf{Setup}$ outputs public parameters $pp$, which implicitly define a secret key space $\mathcal{SK}$ and $\mathcal{K}$, the space of (symmetric) keys.

*Key generation.* $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(pp)$, on input public parameters, outputs a key pair $(pk_0, sk_0)$ with $sk_0 \in \mathcal{SK}$.

*Encapsulation.* $(K, c, pk_{i+1}, mt_{i+1}) \leftarrow \mathsf{Encaps}(pp, pk_i)$, on input the current public key, outputs a symmetric key $K$, a ciphertext $c$, an updated public key $pk_{i+1}$ and a member tag $mt_{i+1}$.

*Member-tag verification.* $0/1 \leftarrow \mathsf{Verify}_{\mathsf{mt}}(pp, pk_i, pk_{i+1}, mt_{i+1})$ verifies the update from $pk_i$ to $pk_{i+1}$ using the tag $mt_{i+1}$.

*Decapsulation.* $(K, sk_{i+1}) \leftarrow \mathsf{Decaps}(pp, sk_i, c, pk_{i+1})$ outputs decapsulated key $K$ and the updated secret key $sk_{i+1}$, but only if it matches $pk_{i+1}$.

**Comparison to APS.** APS [APS23] use "long syntax". To update public and secret keys, they have separate algorithms UpdatePK, which returns an *update ciphertext*, and UpdateSK. Long syntax can be easily transformed to short syntax by running those algorithms inside Encaps and Decaps. An additional difference is that our syntax from [AFM24] makes member tags explicit. Hence, $\mathsf{Verify}_{\mathsf{mt}}$ can be viewed as the analogue of VerifyUpdate in APS, which was originally introduced in [DKW21].

**Correctness.** Let $\ell > 0$ be an integer and $\delta > 0$. Let $pp \leftarrow \mathsf{Setup}$ and $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(pp)$. For $t \in [\ell]$, let

$$(K_t, c_t, pk_t, mt_t) \leftarrow \mathsf{Encaps}(pp, pk_{t-1}) \quad \text{and} \quad (K'_t, sk_t) \leftarrow \mathsf{Decaps}(pp, sk_{t-1}, c_t, pk_t) \ .$$

We say that $\mathsf{UKEM}$ is $(\ell, \delta)$-correct if for all $t \in [\ell]$ we have

$$\Pr\left[K_t = K'_t \wedge \mathsf{Verify}_{\mathsf{mt}}(pp, pk_{t-1}, pk_t, mt_t)\right] \geq 1 - \delta \ ,$$

where the probability is taken over the random coins of the underlying algorithms.

### 4.2 Security

The $\mathsf{IND\text{-}CCA}$ security of UKEM schemes is defined by the experiment in Fig. 3, which we adapted from [AFM24]. As we focus on *member security*, we removed the parts related to "joiner-security", another notion they define.

**Security notion of [AFM24].** At a high level, the $\mathsf{IND\text{-}CCA}$ challenger creates a tree representing the evolution of a (single) UKEM key pair, driven by the adversary's queries. Each node in the tree has assigned a public key and possibly a matching secret key. We call nodes with secret keys *full nodes* and those without *half nodes*. See the illustration in Fig. 4.

The key pair of the root node is generated by the challenger using $\mathsf{KeyGen}$. Key pairs of other nodes are generated by updating the key pairs of their parents. New nodes can be created as follows:

*Enc edges.* When $\mathcal{A}$ calls the $\mathrm{Enc}(i)$, the challenger "honestly" generates a child node of $i$ with the public key generated using $\mathsf{Encaps}$. If $i$ is a full node, then the secret key of the new node is generated using $\mathsf{Decaps}$ (which may result in a half node if $\mathsf{Decaps}$ returns $(\perp, \perp)$ in case of correctness failure). The adversary receives the generated key $K$, the ciphertext, the updated public key and the member tag.

*Dec edges.* When $\mathcal{A}$ calls $\mathrm{Dec}(i, c', pk', mt')$, the challenger first checks if $\mathsf{Verify}_{\mathsf{mt}}$ accepts $pk'$ and $mt'$. If so, it generates a child node of $i$ with public key $pk'$ (chosen by $\mathcal{A}$). If $i$ is a full node, then the secret key of the new node is generated using $\mathsf{Decaps}$ with $c'$ (which could return $\perp$).

*MChal edge.* The adversary can make one call to the MChal oracle for a node of $\mathcal{A}$'s choice, for which a child node is generated, as with Enc. The oracle either returns the generated key $K^{(1)}$ or a random and independent key $K^{(0)}$ and the adversary's goal is to distinguish the two cases.

Further, $\mathcal{A}$ can corrupt nodes by calling the oracle $\mathrm{Rev}(i)$, which reveals the secret key of node $i$. $\mathsf{IND\text{-}CCA}$ thus implies forward-secrecy. Not all nodes can be corrupted, e.g. corrupting the root would allow $\mathcal{A}$ to trivially win. More precisely, $\mathcal{A}$ can corrupt all nodes outside the *challenge set* of the node $i^*$ queried to MChal.

*Challenge set.* The challenge set $S$ of a node $i^*$ contains the nodes on the challenge path from the root to $i^*$ (corrupting any of these allows for a trivial win for any correct scheme). Moreover, $S$ includes *duplicates* (i.e., nodes that have the same public key) of nodes on the challenge path. Finally, $S$ includes *branches*, i.e., all nodes reachable from the challenge path and its duplicates via Dec edges. (As argued in [AFM24], allowing corruption of branches would require computationally heavy tools, such as hierarchical identity-based encryption.)

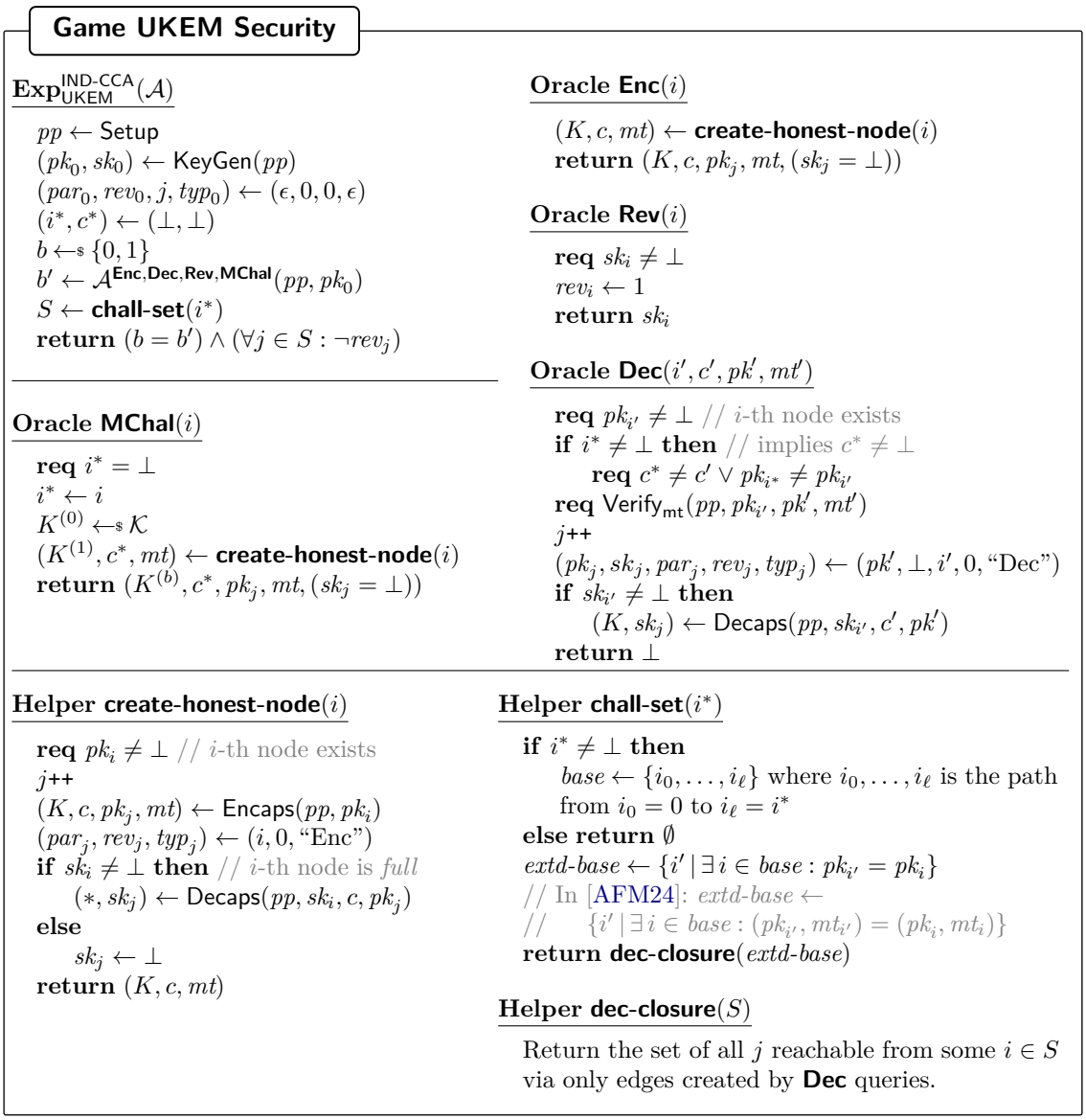



Fig. 3: IND-CCA security for $\mathsf{UKEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps}, \mathsf{Verify}_{\mathsf{mt}})$. By default, all variables are initialized to $\bot$.

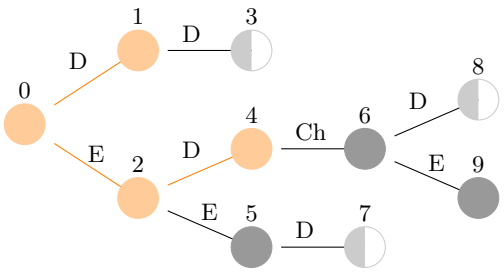
**Definition 3 (UKEM Security).** *Let* $\mathrm{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{UKEM}}(\mathcal{A})$ *be as defined in Fig. 3. The advantage of an adversary* $\mathcal{A}$ *against* IND-CCA *security of* UKEM *is defined as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{UKEM}}(\mathcal{A}) := 2\Pr\left[\mathrm{Exp}^{\mathsf{IND\text{-}CCA}}_{\mathsf{UKEM}}(\mathcal{A}) = 1\right] - 1 \ .$$

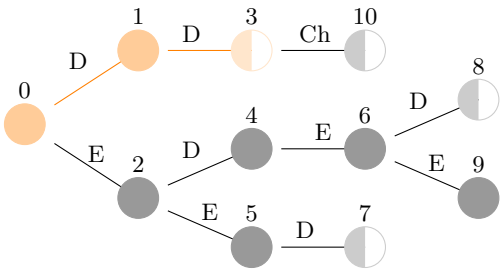
### 4.3 Comparison to AFM

Our security notion is weaker than the one in [AFM24] in three ways: we disallow slightly more corruption queries and we do not consider “joiner security”; moreover, by considering LWE- instead of DL-based schemes, we do not assume perfect correctness.

**Fewer allowed corruptions.** Our challenge set $S$ of node $i^*$ (defining the disallowed corruptions) is larger than in [AFM24] (cf. comment in chall-set in Fig. 3). There, a node is a




(a) $\mathcal{A}$ queries MChal($i^* = 4$), which creates node 6. The challenge set $\{0, 1, 2, 4\}$ includes the challenge path 0-2-4 and in addition 1 which is in its dec-closure.




(b) In a different execution, $\mathcal{A}$ can also challenge a half node i.e. it queries MChal($i^* = 3$).

Fig. 4: Example trees in executions of the IND-CCA experiment in Fig. 3. Full and half nodes are represented, resp., by full circles (●, ●) and half circles (◐, ◐). Edges are tagged whether they were created by an **E**nc, a **D**ec or the **Ch**allenge query. The challenge set is marked in orange. Any node outside of it can be corrupted.

"duplicate" if it has the same public key *and member tag*. (Unlike our definition, their notion implies that given (honestly generated) $pk_i$, $pk_j$ and $mt$, it is hard to produce a different $mt'$ for $pk_i$ and $pk_j$.) They achieve by using a NIZK that is *strongly* simulation-sound (which ours is not).

This difference in UKEM security is irrelevant for most applications: for a party receiving a key $pk_j$ it is important to receive *some* tag $mt$, as this already implies that $pk_j$ can be trusted. It does not matter if $mt$ is the exact tag outputted by encapsulation or different.[8]

**No joiner security.** Like all UPKE/UKEM notions before [AFM24], we too omit the property of *joiner security* defined in [AFM24]. Their construction achieves this notion using statement-malleable NIZK proofs of knowledge, which they instantiated directly in the algebraic group model [FKL18]. We are unaware of any analogous PQ constructions; so achieving efficient lattice-based UPKE joiner security seems to require either a significant breakthrough in lattice-based NIZK technology or a fundamentally new approach to UPKE/UKEM constructions.

Mitigating the lack of joiner security is the following observation. Joiner security only plays a role in the security of UPKE applications in a relatively specific corner case referred to in [AFM24] as "fake group" security, relevant in the following scenario: First, an honest party must join a "fake" group mid-session, which is using an adversarially generated public cryptographic session state. Second, this state must include corrupted public keys on behalf of one or more group members. Third, the joiner must perform the protocol operations to remove any such members. Now, UPKE joiner security is needed to ensure confidentiality for subsequent communication in the group for the joiner.

We note that, although practical protocols like MLS do include mechanisms aimed at providing fake group security (cf. "parent hash" [BBR+23]), many academic protocols do not [ACDT21, AAN+22, KPPW+21]. Similarly, formal security analyses of MLS and

[8] Somewhat related to this, we note that the notion of [AFM24] does not prevent the adversary from creating a $pk_j$ with multiple valid tags using some algorithm other than Encaps. Nor does it imply that it is hard to create a different ciphertext than the one outputted by Encaps that is also accepted as transitioning from $pk_i$ to $pk_j$.

**Construction** $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}[\mathsf{PKE}, H_1, H_2, H_3]$

Setup

$pp_{\mathsf{PKE}} \leftarrow \mathsf{PKE.Setup}$
**return** $pp = pp_{\mathsf{PKE}}$

Encaps$(pp = pp_{\mathsf{PKE}}, pk_i)$

$m \leftarrow\!\!\$\ \{0,1\}^\kappa$
// FO transform
$K \leftarrow H_1(pp_{\mathsf{PKE}}, pk_i, m)$
$r \leftarrow H_2(pp_{\mathsf{PKE}}, pk_i, m)$ // $r \in \{0,1\}^\kappa$
$c \leftarrow \mathsf{PKE.Encrypt}(pp_{\mathsf{PKE}}, pk_i, m; r)$
// Update the public key
$d \leftarrow H_3(pp_{\mathsf{PKE}}, pk_i, m)$ // $d \in \{0,1\}^\kappa$
$(\overline{pk}, \overline{sk}) \leftarrow \mathsf{PKE.KeyGen}(pp_{\mathsf{PKE}}; d)$
$pk_{i+1} \leftarrow pk_i + \overline{pk}$
**return** $(K, c, pk_{i+1}, mt_{i+1} = \epsilon)$

KeyGen$(pp = pp_{\mathsf{PKE}})$

$(pk, sk') \leftarrow \mathsf{PKE.KeyGen}(pp_{\mathsf{PKE}})$
**return** $(pk, sk = (sk', pk))$

Decaps$(pp = pp_{\mathsf{PKE}}, sk_i = (sk'_i, pk_i), c, pk_{i+1})$

$m \leftarrow \mathsf{PKE.Decrypt}(pp_{\mathsf{PKE}}, sk'_i, c)$
// FO transform
$K \leftarrow H_1(pp_{\mathsf{PKE}}, pk_i, m)$
$r \leftarrow H_2(pp_{\mathsf{PKE}}, pk_i, m)$ // $r \in \{0,1\}^\kappa$
**req** $c = \mathsf{PKE.Encrypt}(pp_{\mathsf{PKE}}, pk_i, m; r)$
// Update the key pair
$d \leftarrow H_3(pp_{\mathsf{PKE}}, pk_i, m)$ // $d \in \{0,1\}^\kappa$
$(\overline{pk}, \overline{sk}) \leftarrow \mathsf{PKE.KeyGen}(pp_{\mathsf{PKE}}; d)$
$sk'_{i+1} \leftarrow sk'_i + \overline{sk}$
**req** $pk_{i+1} = pk_i + \overline{pk}$
**return** $(K, sk_{i+1} = (sk'_{i+1}, pk_{i+1}))$

Fig. 5: Our (key-homomorphic) UKEM construction without public verifiability from kh-PKE and three random oracles. W.l.o.g. we define key space, PKE KeyGen and Encrypt randomness spaces and message space to be $\{0,1\}^\kappa$.

related protocols often omit the case for simplicity, focusing instead on (presumably more important) security properties.

**No perfect correctness.** The security notion of [AFM24] assumes perfect correctness. Thus, when the Enc or MChal oracle creates the secret key of a new node via Decaps, this is assumed to always succeed. In our notion, if Decaps outputs $\perp$ as the secret key, the challenger simply creates a half-node. $\mathcal{A}$ is also notified if there was a decryption error, i.e., if the new secret key $sk_j = \perp$. (Our syntax requires that in this case the decapsulated key $K'$ is $\perp$ as well.) Further, key generation can output $\perp$ as well, which means that the root is a half node.

## 5 Our UKEM Construction

We build our **L**attice-based **u**pdatable KEM in a modular and black-box way. From kh-PKE (and hashes), we construct $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$, a *key-homomorphic* UKEM (defined below), that does not yet provide public verifiability (and outputs "empty" member tags). This is added by combining a kh-UKEM (instantiated with $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$) with a NIZK proof of knowledge PoK, yielding LuKEM.

$\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$. The scheme is defined in Fig. 5 and uses a kh-PKE scheme PKE. Encapsulation samples a random message $m$ and inputs it to three hash functions (modeled as random oracles): The first two are used for the FO transform: $H_1$ outputs the encapsulated key $K$ and $H_2$ outputs the randomness for the encryption of $m$. The output of $H_3$ is used as randomness for kh-PKE key generation in order to derive the shift $(\overline{pk}, \overline{sk})$. Decapsulation is analogous. In Section 6.1, we give a lattice-based instantiation of PKE based on [APS23].

LuKEM. The scheme LuKEM is defined in Fig. 6. It assumes two building blocks: a NIZKPoK, and a *key-homomorphic UKEM*.

**Construction** LuKEM[UKEM, PoK]

Setup

$pp_{\mathsf{UKEM}} \leftarrow \mathsf{UKEM.Setup}$
$crs \leftarrow \mathsf{PoK.Setup}$
**return** $pp = (pp_{\mathsf{UKEM}}, crs)$

$\mathsf{KeyGen}(pp = (pp_{\mathsf{UKEM}}, *))$

**return** $\mathsf{UKEM.KeyGen}(pp_{\mathsf{UKEM}})$

$\mathsf{Encaps}(pp = (pp_{\mathsf{UKEM}}, pp_{\mathsf{PoK}}), pk_i)$

$(K, c, pk_{i+1}, *, \overline{sk}) \leftarrow \mathsf{UKEM.Encaps}(pp_{\mathsf{UKEM}}, pk_i)$
$mt_{i+1} \leftarrow \mathsf{PoK.Prove}(crs, (pp_{\mathsf{UKEM}}, pk_i, pk_{i+1}), \overline{sk})$
**return** $(K, c, pk_{i+1}, mt_{i+1})$

$\mathsf{Decaps}(pp = (pp_{\mathsf{UKEM}}, *), sk_i, c, pk_{i+1})$

**return** $\mathsf{UKEM.Decaps}(pp_{\mathsf{UKEM}}, sk_i, c, pk_{i+1})$

$\mathsf{Verify}_{\mathsf{mt}}(pp = (pp_{\mathsf{PKE}}, crs), pk_i, pk_{i+1}, mt_{i+1})$

**return** $\mathsf{PoK.Verify}(crs, (pp_{\mathsf{UKEM}}, pk_i, pk_{i+1}), mt_{i+1})$

Fig. 6: Our final construction, which achieves UKEM security (Definition 3) by extending a key-homomorphic UKEM (Definition 4) with a proof of knowledge PoK for the language in Eq. (2).

**Definition 4.** *A UKEM is key-homomorphic if it has the following two properties: (i) public and secret keys are kh-PKE public and secret keys, (ii)* Encaps *outputs the shift* $\overline{sk}$ *corresponding to* $pk_{i+1} - pk_i$.

$\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ can be easily made key-homomorphic by having Encaps also output $\overline{sk}$. The NIZKPoK PoK then proves knowledge of this shift: the proved statement is a pair of PKE public keys $pk_i, pk_{i+1}$ and the corresponding witness is a secret key $\overline{sk}$ for $pk_{i+1} - pk_i$. We define the relation $\mathcal{R}$:

$$(x = (pp_{\mathsf{PKE}}, pk_i, pk_{i+1}), w = \overline{sk}) \in \mathcal{R} \iff \mathsf{pk}(\overline{sk}) = pk_{i+1} - pk_i \wedge \overline{sk} \in \mathcal{SK}\,, \qquad (2)$$

where $pp_{\mathsf{PKE}}$ defines $\mathcal{SK}$. We let $\mathsf{LuKEM}[\mathsf{PKE}, \mathsf{PoK}]$ denote LuKEM instantiated with $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}[\mathsf{PKE}, H_1, H_2, H_3]$ and PoK using random oracles $H_1, H_2, H_3$. We present a lattice-based instantiation of PoK in Section 6.2. We also prove in Appendix E that $\mathsf{LuKEM}[\mathsf{PKE}, \mathsf{PoK}]$ is robust, i.e., correct in the presence of adversarial updates, if PKE is correct and PoK is complete.

### 5.1 Security

Our main theorem shows that our UKEM construction LuKEM is IND-CCA secure.

**Theorem 1.** *Let* PKE *be a key-homomorphic PKE with message space* $\{0,1\}^\kappa$ *that is* $(\ell, \delta)$*-correct,* $\gamma$*-spread and* Shifty-IND-CPA *secure, and let* PoK *be a straightline extractable zero-knowledge proof system for the relation in* (2)*. Then* $\mathsf{LuKEM}[\mathsf{PKE}, \mathsf{PoK}]$ *is* IND-CCA *secure in the ROM.*

*Specifically, for any adversary* $\mathcal{A}$ *against* IND-CCA *security of* LuKEM *that issues* $q_{\mathrm{ro}}$ *RO queries,* $q_{\mathrm{enc}}$ *Enc queries and* $q_{\mathrm{dec}}$ *Dec queries and creates a tree of depth at most* $\ell$*, there exist adversaries* $\mathcal{B}$*,* $\mathcal{B}_1$*,* $\mathcal{B}_2$ *and* $\mathcal{B}_3$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{LuKEM}}(\mathcal{A}) \;\le\; & (q_{\mathrm{enc}} + q_{\mathrm{dec}}) \cdot \mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{B}) \\ & + 2 \cdot \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{PoK}}(\mathcal{B}_2) + 2 \cdot \mathsf{Adv}^{\mathsf{SI}}_{\mathsf{PoK}}(\mathcal{B}_1) + 2 \cdot \mathsf{Adv}^{\mathsf{Ext}}_{\mathsf{PoK}}(\mathcal{B}_3) \\ & + q_{\mathrm{ro}} \cdot (q_{\mathrm{enc}} + 1) \cdot 2^{-\kappa} + q_{\mathrm{dec}} \cdot 2^{-\gamma} + (2q_{\mathrm{ro}} + q_{\mathrm{enc}} + 1) \cdot \delta\ .\end{aligned}$$

**Proof outline.** The full proof is given in Appendix F. For a modular treatment, we split the proof in three steps and define two intermediate security notions for *key-homomorphic* UKEM's that are "between" IND-CPA and IND-CCA. The first, IND-CPA$^+$, is similar to IND-CCA except that instead of a decryption oracle the adversary gets an oracle $\mathbf{Upd}(i, \overline{sk})$, which lets the adversary create a "Dec" node by shifting the keys of node $i$ by a (well-formed) update $\overline{sk}$ of its choice. We show that if PKE is Shifty-IND-CPA secure, then LuKEM$_{\mathsf{no\text{-}pv}}$ is IND-CPA$^+$ secure and give ample intuition below.

The second notion, IND-CCA$^-$, has a (restricted) decryption oracle, which does not yet capture public verifiability; the oracle only creates a new node if Decaps succeeds. LuKEM$_{\mathsf{no\text{-}pv}}$ achieves this notion by simulating decryption via the random oracles, using the technique from the FO transform. By adding member tags via the NIZKPoK we then obtain (in a black-box way) our final scheme LuKEM, for which we prove IND-CCA security assuming the underlying key-homomorphic UKEM (as in Definition 4) is IND-CCA$^-$ and PoK is a straightline simulation-extractable NIZK.

**From IND-CPA to IND-CPA$^+$.** We show that if PKE is Shifty-IND-CPA secure and correct, then LuKEM$_{\mathsf{no\text{-}pv}}$ is IND-CPA$^+$ secure.

Relying on $\delta$-correctness of PKE, we can assume that key generation does not fail and thus the updated secret keys computed by the challenger during Enc and MChal calls (using Decaps) are not $\perp$. This implies that there are no half-nodes in the experiment. This entails the last term in the security bound in Theorem 1.

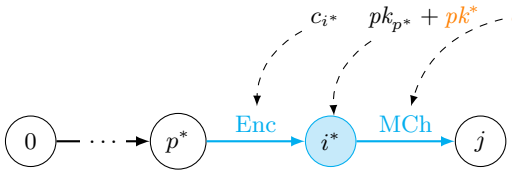
Let $\mathcal{A}$ be an adversary against the IND-CPA$^+$ security of LuKEM$_{\mathsf{no\text{-}pv}}$. Consider the event Brk that occurs when $\mathcal{A}$ queries (one of) the RO on the message $m^*$ encrypted in $c^*$ returned by MChal. Until Brk occurs, the "real" and "random" IND-CPA$^+$ experiments are identical; the challenge key $K$, as well as $r$ and $d$ chosen by Encaps during the MChal query are random and independent of $\mathcal{A}$'s view. $\mathcal{A}$'s challenge bit is thus perfectly hidden, and it remains to upper-bound the probability of Brk. To this end, we construct a reduction $\mathcal{B}$ against the Shifty-IND-CPA security of PKE.

*Assuming no Upd queries.* We first describe the reduction $\mathcal{B}$ assuming that $\mathcal{A}$ does not make Upd queries. $\mathcal{B}$ is given $pk^*$ and access to the Chal oracle in the Shifty game $\mathcal{B}$ guesses the index $i^*$, hoping that $\mathcal{A}$ queries MChal($i^*$). As illustrated in Fig. 7a, $\mathcal{B}$ embeds $pk^*$ as the update from the parent $p^*$ of $i^*$ to $i^*$, i.e., $pk_{i^*} = pk_{p^*} + pk^*$. This allows $\mathcal{B}$ to embed its challenge $c^*$ as the ciphertext returned by MChal($i^*$): $\mathcal{B}$ queries its own oracle Chal with two random messages $m^{(0)}, m^{(1)} \leftarrow\!\!\$\ \{0,1\}^\kappa$ and the key shift $sk_{p^*}$[9] (since there are no Upd queries, $\mathcal{B}$ chose $sk_{p^*}$ itself). Chal returns $c^*$ which encrypts $m^{(b)}$ to $pk_{p^*} + pk^*$, and $\mathcal{B}$ forwards $c^*$ to $\mathcal{A}$ as the output of MChal.

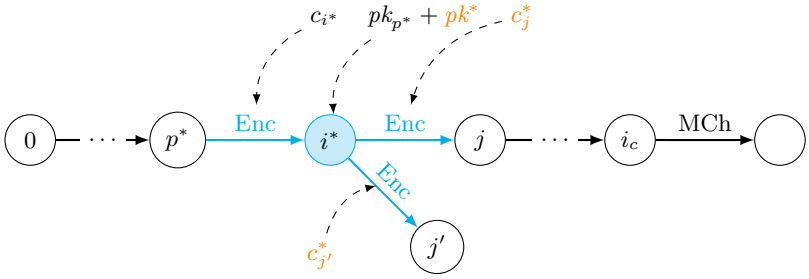
If $\mathcal{A}$ queries the RO on an input containing $m^{(b')}$ for some $b'$, then $\mathcal{B}$ stops and outputs $b'$. Observe that if $\mathcal{B}$'s challenge bit is $b$ then $m^{(1-b)}$ is random and independent of $\mathcal{A}$'s view. Thus, the probability that an RO query by $\mathcal{A}$ contains $m^{(1-b)}$ (and thus $\mathcal{B}$ loses) is at most $2^{-\kappa}$. So if $\mathcal{A}$ triggers Brk then $\mathcal{B}$ outputs the correct bit except with probability $q_{\mathrm{ro}} \cdot 2^{-\kappa}$ (by the union bound).

*Embedding $pk^*$ and challenges.* When embedding $pk^*$ as the update from node $p^*$ to $i^*$, $\mathcal{B}$ simulates the other outputs of the Enc oracle, $K_{i^*}$ and $c_{i^*}$, by running Encaps with a random message $m_{i^*}$. This means that, not knowing $sk^*$, $\mathcal{B}$ cannot consistently answer

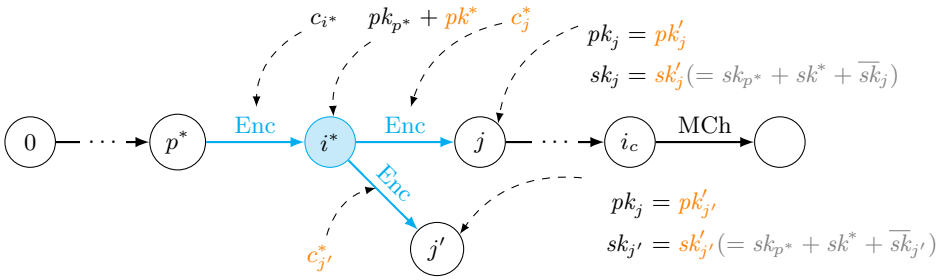
[9] The Chal oracle of Shifty-IND-CPA expects a vector of secret keys as input and sums them. For readability, we directly consider this sum. Since the Upd oracle requires the same condition on individual secret keys and the maximum number of keys $\ell$ is that for correctness, the reduction will never make an invalid query.




(a) "Simple case": $\mathcal{B}$ embeds $pk^*$ as the update from $p^*$ to $i^*$, queries Chal on two random messages and the shift $sk_{p^*}$ (assuming Enc edges between 0 and $p^*$). Chal returns $c^*$, which $\mathcal{B}$ forwards as the ciphertext returned by MChal.




(b) $\mathcal{B}$ embeds $pk^*$ as the first update on the path from 0 to $i_c$ for which $\mathcal{A}$ will break some ciphertext. $\mathcal{B}$ embeds one challenge in each Enc-edge going out of $i^*$.




(c) $\mathcal{A}$ can query the Rev oracle for any node outside the challenge path 0-$i_c$. To answer such queries for descendants of $i^*$, $\mathcal{B}$ uses its Chal oracle: Each time $\mathcal{A}$ queries Enc($i^*$) creating node $j$, $\mathcal{B}$ queries Chal which returns $c_j^*$, $pk'_{j'}$ and $sk'_{j'}$. As the output of Enc, $\mathcal{B}$ returns $c_j^*$ and $pk_j := pk'_{j'}$. As guaranteed by Shifty-IND-CPA, $sk'_{j'}$ is the key $sk_{j'}$.

Fig. 7: Simulations of the UKEM security experiment by reduction $\mathcal{B}$ to Shifty-IND-CPA security running the UKEM adversary $\mathcal{A}$, *assuming there are no Upd edges*. Orange marks values obtained from the Shifty-IND-CPA challenger. Cyan marks nodes for which $\mathcal{B}$ does not know the secret key and edges for which $\mathcal{B}$ doesn't know the secret update $\overline{sk}$.

if $\mathcal{A}$ queries $m_{i^*}$ to the RO $H_3$. But if $\mathcal{A}$ makes this query, then $\mathcal{B}$ should have guessed differently and embedded $pk^*$ as the update from the parent of $p^*$ to $p^*$, and $c^*$ as $c_{i^*}$.

More generally, $\mathcal{B}$'s strategy, as illustrated in Fig. 7b, is to guess the following index $i^*$: Let $i_c$ denote the node for which $\mathcal{A}$ calls MChal($i_c$). Now guess the *first* node $i^*$ on the *challenge path* (the nodes from the root 0 to $i_c$) that has a child $j$ created by an Enc or MChal query, which returned $c_j$, and $\mathcal{A}$ "breaks" $c_j$. "Breaking" $c_j$ means that $\mathcal{A}$ queries the message encrypted in $c_j$ to the RO. From another perspective, $\mathcal{A}$ breaking $c_j$ means that $\mathcal{B}$ wins if it embeds $c^*$ as $c_j$.

Observe that $\mathcal{B}$ does not know which child of $i^*$ will end up on the challenge path. Therefore, it calls its Chal oracle for each query Enc($i^*$) and MChal($i^*$) and embeds the challenge ciphertext as $c_j$ (where $j$ is the new node) returned by its oracle Chal on fresh random messages $m_j^{(0)}$ and $m_j^{(1)}$.

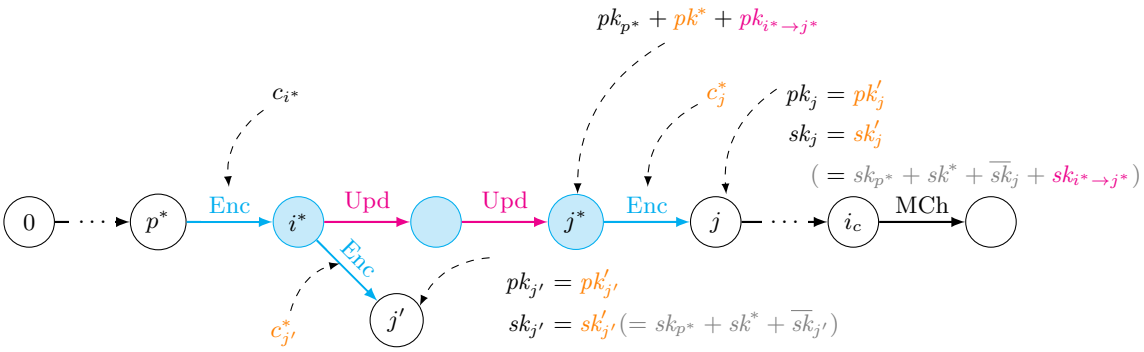



Fig. 8: Adjusting the strategy of the reduction $\mathcal{B}$ to deal with Upd queries. Magenta marks edges for which $\mathcal{A}$ (playing in the IND-CPA$^+$ game) provides secret keys $\overline{sk}$. Later, these edges will represent Dec queries for which the reduction will extract secret keys from ciphertexts and RO queries (for $\mathcal{A}$ in the IND-CCA$_{\text{no-pv}}$ game) or PoKs (for $\mathcal{A}$ in the IND-CCA game).

For this strategy we define a "break event" Brk that is more general than the one defined above; it occurs whenever $\mathcal{A}$ queries any RO on the message encrypted in $c_j$ returned by Enc($i^*$) or MChal($i^*$) for some $i^*$ on the challenge path. As soon as $\mathcal{A}$ makes a query that contains any of the messages $m_j^{(b')}$, $\mathcal{B}$ returns the bit $b'$. Again, if $b$ is $\mathcal{B}$'s challenge bit, all $m_j^{(1-b)}$ are independent of $\mathcal{A}$'s view and $\mathcal{B}$ wins the Shifty-IND-CPA game with overwhelming probability (if $\mathcal{B}$'s guess was correct).

*Answering Rev queries.* Consider a query Enc($i^*$) or MChal($i^*$) which creates a new node $j'$. If $j'$ does not end up on the challenge path, $\mathcal{A}$ can query Rev($j'$). Thus, $\mathcal{B}$ needs to generate an updated key pair $(pk_{j'}, sk_{j'})$ in a way that is consistent with $\mathcal{A}$'s view. Recall that in the game played by $\mathcal{A}$ the challenger sets $pk_{j'} := pk_{i^*} + \overline{pk}_{j'}$ and $sk_{j'} := sk_{i^*} + \overline{sk}_{j'}$ for a key pair $(\overline{pk}_{j'}, \overline{sk}_{j'})$ derived from $d$ outputted by the RO on input $m$ encrypted in $c_{j'}^*$.

$\mathcal{B}$ knows neither $m$ nor $sk_{i^*}$. However, unless Brk occurs (in which case $\mathcal{B}$ stops the simulation), $\mathcal{A}$ does not query the RO on $m$. Therefore, from $\mathcal{A}$'s perspective $(\overline{pk}_{j'}, \overline{sk}_{j'})$ is generated independently at random, and so $\mathcal{B}$ can set the key pair $(pk_{j'}, sk_{j'})$ of revealed node $j'$ to the value $(pk', sk')$ it got from its Chal oracle when it created node $j'$. (Recall that Chal computes $sk' := sk_{p^*} + sk^* + \overline{sk}$ for a fresh $\overline{sk}$, exactly as required.) See Fig. 7c.

*Adding Upd queries.* We modify $\mathcal{B}$'s strategy to deal with $\mathcal{A}$'s Upd queries. First, we look at ancestors of $p^*$ that are created by Upd queries. This would be a problem if $\mathcal{B}$ did not know $sk_{p^*}$, which is the sum of the key shifts it sends to the Chal oracle. Fortunately, since $\mathcal{B}$ has computed all nodes until then honestly, it knows all corresponding secret keys including $sk_{p^*}$.

Second, Upd queries mean that the parent of $j$ for which $\mathcal{A}$ breaks $c_j$ may be created by an Upd query, as illustrated in Fig. 8. Thus, we adjust $\mathcal{B}$'s guessing strategy as follows. Analogously to IND-CCA security (cf. Fig. 3), IND-CPA$^+$ defines for an (honest) node $i$ the set dec-closure($i$) of all its descendants reachable from $i$ via only Upd edges. $\mathcal{B}$'s strategy is to guess the index $i^*$ of the first node created by an Enc query with a descendant $j^* \in$ dec-closure($i^*$) such that $\mathcal{A}$ breaks $c_j$ for some child $j$ of $j^*$ created by an Enc query. Accordingly, $\mathcal{B}$ embeds challenges in all Enc edges going out of dec-closure($i^*$).

With the adjusted strategy, a challenge $c_j^*$ embedded by $\mathcal{B}$ is no longer encrypted to $pk_{i^*}$ but to $pk_{j^*} = pk_{i^*} + pk_{i^* \to j^*}$ where $pk_{i^* \to j^*}$ is the sum of updates chosen by $\mathcal{A}$ when

it created the Upd edges between $i^*$ and $j^*$. This means that $\mathcal{B}$ needs to send to Chal the key shifts that sum up to $sk_{p^*} + sk_{i^* \to j^*}$. As we explained before, $\mathcal{B}$ can compute $sk_{p^*}$ honestly. Later (in Theorems 4 and 5), we will explain how Dec queries can be translated to Upd queries.

## 6 Instantiation of Building Blocks

### 6.1 Key-Homomorphic PKE from Lattices

We start by recalling some background on lattices. The following definition is from [BF11]:

**Definition 5 (Gaussian distribution).** *Let $m \in \mathbb{N}$ and $\mathbf{c} \in \mathbb{R}^m$ and $\sigma > 0$. Define the Gaussian function $\rho_{\sigma,\mathbf{c}} \colon \mathbb{R}^m \to \mathbb{R}$*

$$\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp\left(-\pi\|\mathbf{x}-\mathbf{c}\|_2^2/\sigma^2\right)$$

*The Gaussian distribution on $\mathbb{Z}^m$ with center $\mathbf{c}$ and parameter $\sigma$ is defined as $\mathcal{D}_{\mathbb{Z}^m,\sigma,\mathbf{c}}(\mathbf{x}) = \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\sum_{\mathbf{y}\in\mathbb{Z}^m}\rho_{\sigma,\mathbf{c}}(\mathbf{y})$. We write $\mathcal{D}_{\mathbb{Z}^m,\sigma}$ for $\mathcal{D}_{\mathbb{Z}^m,\sigma,\mathbf{0}}$. We further write $\mathcal{D}_{\mathbb{Z}^{m\times n},\sigma}$ for the distribution obtained by sampling $n$ vectors from $\mathcal{D}_{\mathbb{Z}^m,\sigma}$ and viewing them as the columns of a matrix in $\mathbb{Z}^{m\times n}$.*

The following is from [BF11], adapted as [APS23, Lemma 2].

**Definition 6 (Convolution).** *Let $m \in \mathbb{N}$; let $\mathcal{D}_1, \mathcal{D}_2$ be two distributions on $\mathbb{Z}^m$. If two independent random variables are distributed as $X_1 \sim \mathcal{D}_1$ and $X_2 \sim \mathcal{D}_2$, then their sum is distributed as*

$$(\mathcal{D}_1 * \mathcal{D}_2)(\mathbf{x}) \coloneqq \sum_{\mathbf{y}\in\mathbb{Z}^m} \mathcal{D}_1(\mathbf{x}-\mathbf{y})\mathcal{D}_2(\mathbf{y}) \ .$$

**Lemma 1 (Gaussian convolution).** *Let $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{Z}^n$, let $X_1 \sim \mathcal{D}_{\mathbb{Z}^n,\sigma_1,\mathbf{c}_1}$, $X_2 \sim \mathcal{D}_{\mathbb{Z}^n,\sigma_2,\mathbf{c}_2}$, let $Y$ be the distribution of $X_1 + X_2$ and let $\varepsilon > 0$. If*

$$1/\sigma_1^2 + 1/\sigma_2^2 < \pi/\left(\ln(2n(1+1/\varepsilon))\right)$$

*then*

$$\Delta(Y, \mathcal{D}_{\mathbb{Z}^n,\sqrt{\sigma_1^2+\sigma_2^2},\mathbf{c}_1+\mathbf{c}_2}) < 2\varepsilon/(1-\varepsilon) \ .$$

**Definition 7 (LWE assumption).** *Let $q, n, m \geq 0$, $\mathcal{S}$ be a distribution on $\mathbb{Z}_q^n$ and $\chi$ be an error distribution on $\mathbb{Z}^m$. The goal of the adversary $\mathcal{A}$ in game $\mathsf{LWE}_{q,n,m,\chi,\mathcal{S}}$ is to distinguish between $(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$ and $(\mathbf{A}, \mathbf{u})$ for $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{m\times n}$, $\mathbf{s} \leftarrow \mathcal{S}$, $\mathbf{e} \leftarrow \chi^m$ and $\mathbf{u} \leftarrow\!\!\$\ \mathbb{Z}_q^m$. We define $\mathcal{A}$'s advantage as*

$$\mathsf{Adv}^{\mathsf{LWE}}_{q,n,m,\chi,\mathcal{S}}(\mathcal{A}) \coloneqq \left|\Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{A}\mathbf{b}+\mathbf{e}) \Rightarrow 1\right] - \Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{u}) \Rightarrow 1\right]\right| .$$

*For $\sigma > 0$, we write $\mathsf{LWE}_{q,n,m,\sigma}$ to denote $\mathsf{LWE}_{q,n,m,\chi,\mathcal{S}}$ when $\chi = \mathcal{D}_{\mathbb{Z}^m,\sigma}$ and $\mathcal{S}$ the uniform distribution over $\mathbb{Z}_q^n$.*

The next definition is adapted from [APS23, Def. 13]. We state the "multi-secret variant" in transposed form (which is how we use it in the proof of Theorem 2).

**Definition 8 (Multi-secret HNF adaptive extended LWE).** *Let $q, n, m, k, B \geq 0$ and $\chi$ be an error distribution on $\mathbb{Z}^m$. We define the following game for $\beta \in \{0,1\}$:*

$$
\begin{aligned}
&\mathbf{Exp}^{\mathsf{mHaeLWE},(\beta)}_{q,n,m,k,\chi,B}(\mathcal{A}) \\
&\quad \mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m} \\
&\quad (\mathbf{z}_0, \mathbf{z}_1, st) \leftarrow \mathcal{A}_1(\mathbf{A}) \\
&\quad \textbf{if } \|\mathbf{z}_0\|_\infty > B \textbf{ or } \|\mathbf{z}_1\|_\infty > B \\
&\quad\quad \textbf{return } 0 \\
&\quad \mathbf{X} \leftarrow \chi^{k\times n}; \mathbf{E} \leftarrow \chi^{k\times m}; \mathbf{g} \leftarrow \chi^k \\
&\quad \mathbf{h} \leftarrow \mathbf{X}\mathbf{z}_0 + \mathbf{E}\mathbf{z}_1 + \mathbf{g} \\
&\quad \textbf{if } \beta = 0 \textbf{ then } \mathbf{C} \leftarrow \mathbf{X}\mathbf{A} + \mathbf{E} \\
&\quad \textbf{if } \beta = 1 \textbf{ then } \mathbf{C} \leftarrow\!\!\$\ \mathbb{Z}_q^{k\times m} \\
&\quad \textbf{return } \mathcal{A}_2(st, \mathbf{C}, \mathbf{h})
\end{aligned}
$$

*We define $\mathcal{A}$'s advantage as*

$$
\mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,m,k,\chi,B}(\mathcal{A}) := \left|\Pr[\mathbf{Exp}^{\mathsf{mHaeLWE},(1)}_{q,n,m,k,\chi,B}(\mathcal{A}) \Rightarrow 1] - \Pr[\mathbf{Exp}^{\mathsf{mHaeLWE},(0)}_{q,n,m,k,\chi,B}(\mathcal{A}) \Rightarrow 1]\right| .
$$

As for LWE, we write $\mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,m,k,\sigma,B}$ for $\mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,m,k,\mathcal{D}_{\mathbb{Z}^m,\sigma},B}$. The (multi-) HaeLWE assumption was shown to be implied by LWE for appropriate choices of parameters [APS23, Lemma 5 and Theorem 1]. (We state the corollary for LWE with parameter $\sigma/2$, which is how we will apply it in Corollary 2.)

**Corollary 1.** *Let $q \geq 25$ be a prime, $n \geq 1$, $m \geq 16n + 4\log\log q$, $\gamma, \sigma, B \geq 0$ with*

$$
\sigma \geq \sqrt{8\ln(2(n+1)(1+1/\varepsilon))/\pi} \qquad \text{and} \qquad \gamma > \sigma\sqrt{(1+nB^2)/2} .
$$

*Define $m' := m - 16n - 4\log\log q$. For any adversary $\mathcal{A}$ against multi-HaeLWE there exists a reduction $\mathcal{B}$ with similar running time against LWE so that*

$$
\mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,m',k,\gamma,B}(\mathcal{A}) \leq 4k \cdot \left(\mathsf{Adv}^{\mathsf{LWE}}_{q,n,m,\sigma/2}(\mathcal{B}) + 4\varepsilon/(1-\varepsilon)\right) .
$$

**The APS key-homomorphic PKE.** In Fig. 9 we recall the PKE scheme from [APS23], which is derived from Kyber [BDK+18], which is based on [LPS10]. The scheme depends on several parameters.

For message space $\mathcal{M} := \mathbb{Z}_p^n$, to guarantee that, after $\ell$ (called $k$ in APS) subsequent key updates, the decryption failure probability is bounded by $\delta$, we require that for $y := \sqrt{-2\ln(\delta/(4n))}$ we have

$$
q > 2p\sigma_c \cdot (2y^2\sigma n\ell + y) . \tag{3}
$$

For security, in terms of a statistical security parameter $\varepsilon$, we require that[10]

$$
\sigma \geq \sqrt{8\ln(2n(1+1/\varepsilon))/\pi} \qquad \text{and} \qquad \sigma_c > 2\sigma\sqrt{1 + n((\ell+1)y\sigma)^2} ,
$$

and that LWE holds for certain parameters (see Corollary 2).

For correctness, we first bound the norms $\|\mathbf{s}\|_\infty$ and $\|\mathbf{e}\|_\infty$:

[10] The authors [APS23] claim to require $\sigma \geq \sqrt{2\ln(2n(1+1/\varepsilon))/\pi}$ (i.e, only half of our bound), which (by Lemma 1) guarantees that the sum of two $\mathcal{D}_{\mathbb{Z}^n,\sigma}$-distributed random variables is close to $\mathcal{D}_{\mathbb{Z}^n,\sqrt{2}\sigma}$; however, in their proof, they require this also for random variables with variance $\sigma/2$.

**Construction** $\mathsf{PKE}_{\mathsf{LWE}}$

**Parameters:** $n, q, p, \sigma, \sigma_c$

Setup

$\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times n}$ // can be derived from seed
**return** $pp = \mathbf{A}$

KeyGen($pp = \mathbf{A}$)

$\mathbf{s}, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$
$\mathbf{b} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$
**return** $(pk = \mathbf{b}, sk = (\mathbf{s}, \mathbf{e}))$

Encrypt($pp = \mathbf{A}, pk = \mathbf{b}, \mathbf{m} \in \mathbb{Z}_p^n$)

$\mathbf{X}, \mathbf{E} \leftarrow \mathcal{D}_{\mathbb{Z}^{n\times n},\sigma_c}$
$\mathbf{f} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_c}$
$\mathbf{C} \leftarrow \mathbf{X}\mathbf{A} + \mathbf{E}$
$\mathbf{c} \leftarrow \mathbf{X}\mathbf{b} + \mathbf{f} + \lfloor q/p \rfloor \cdot \mathbf{m} \bmod q$
**return** $ct = (\mathbf{C}, \mathbf{c})$

Decrypt($pp = \mathbf{A}, sk = (\mathbf{s}, *), ct = (\mathbf{C}, \mathbf{c})$)

$\mathbf{v} \leftarrow \mathbf{c} - \mathbf{C}\mathbf{s}$
**return** $\lfloor p/q \cdot \mathbf{v} \rceil_p$

Fig. 9: LWE-based (key-homomorphic) PKE scheme.

**Lemma 2.** $\Pr\left[\|\mathbf{s}\|_\infty \geq y\sigma \vee \|\mathbf{e}\|_\infty \geq y\sigma\right] < \delta/2$.

*Proof.* Since $\|\mathbf{x}\|_2 \leq \sqrt{n}\|\mathbf{x}\|_\infty$, we have

$$\Pr_{\mathbf{x}\leftarrow\mathcal{D}_{\mathbb{Z}^n,\sigma}}\left[\|\mathbf{x}\|_\infty \geq y\sigma\right] \leq \Pr_{\mathbf{x}\leftarrow\mathcal{D}_{\mathbb{Z}^n,\sigma}}\left[\|\mathbf{x}\|_2 \geq \sqrt{n}y\sigma\right] \leq e^{-(n/2)(\sqrt{2\pi}\cdot y-1)^2} ,$$

where the last inequality follows from a Gaussian tail bound ([APS23, Lemma 1] for $t := \sqrt{2\pi}\cdot y$). Since $y > 1$, we have $\sqrt{2\pi}\cdot y - 1 > y$, and thus the above is (strictly) upper-bounded by $e^{-(n/2)y^2} = (\delta/4n)^n \leq \delta/4$. The lemma now follows by a union bound. □

Motivated by the above, we define the extended secret key space (which encompasses keys that have been updated up to $\ell$ times) as

$$\overline{\mathcal{SK}}_\ell := \left\{(\mathbf{s}, \mathbf{e}) \in \mathbb{Z}^{n\times n} \,\middle|\, \|\mathbf{s}\|_\infty, \|\mathbf{e}\|_\infty \leq B\right\} \quad \text{with} \quad B = (\ell+1)y\sigma .$$

The following correctness property was shown in [APS23, Appendix B.2]:

**Proposition 1.** *Let $\ell > 1$ and $\delta > 0$. Consider the scheme in Fig. 9 with $q$ satisfying (3) for $\ell$ and $\delta$. Let $sk \in \overline{\mathcal{SK}}_\ell$ and $\mathbf{m} \in \mathcal{M}$. Then*

$$\Pr\left[\mathsf{Decrypt}(sk, \mathsf{Encrypt}(\mathsf{pk}(sk), \mathbf{m}) \neq \mathbf{m}\right] \leq \delta .$$

Moreover, $\gamma$-spreadness was shown in [APS23, Appendix B.3].

**Shifty Security of APS scheme.** The basic security notion for UPKE defined in [APS23], and shown to be satisfied by the APS scheme, is IND-CR-CPA. In the security game, the adversary receives a public key $(\mathbf{A}, \mathbf{b}^* = \mathbf{A}\mathbf{s}^* + \mathbf{e}^*)$ and can ask for updates of this key using randomness $(\mathbf{s}_i, \mathbf{e}_i)$ specified by the adversary. This results in key $(\mathbf{A}, \mathbf{b} = \mathbf{b}^* + \mathbf{A}\mathbf{s} + \mathbf{e})$ with $\mathbf{s} := \sum \mathbf{s}_i$ and $\mathbf{e} := \sum \mathbf{e}_i$. The adversary then specifies $(\mathbf{m}_0, \mathbf{m}_1)$ and is given an encryption of $\mathbf{m}_\beta$, for $\beta \leftarrow \{0, 1\}$, under key $(\mathbf{A}, \mathbf{b})$. It can then make more updates (their total number required to be less than $t$), whose sum we denote by $(\hat{\mathbf{s}}, \hat{\mathbf{e}})$, resulting in key $(\mathbf{A}, \hat{\mathbf{b}})$. After one honest update, using $(\overline{\mathbf{s}}, \overline{\mathbf{e}})$, the adversary is (basically[11]) given $(\mathbf{s}' := \mathbf{s}^* + \mathbf{s} + \hat{\mathbf{s}} + \overline{\mathbf{s}}, \mathbf{e}' := \mathbf{e}^* + \mathbf{e} + \hat{\mathbf{e}} + \overline{\mathbf{e}})$ as well as encryption of $\overline{\mathbf{s}}$ under $(\mathbf{A}, \hat{\mathbf{b}})$.

[11] In addition to $\mathbf{s}'$, the adversary is given $pk' := (\mathbf{A}, \mathbf{b}' := \mathbf{A}\mathbf{s}' + \mathbf{e}')$ instead of $\mathbf{e}'$, which can however be computed from the former as $\mathbf{b}' - \mathbf{A}\mathbf{s}'$.

IND-CR-CPA from [APS23] implies a variant of Shifty-IND-CPA where the adversary is only allowed one **Chal** query: Given an adversary $\mathcal{A}$ against Shifty, we construct $\mathcal{B}$ against IND-CR-CPA: $\mathcal{B}$ is given $(\mathbf{A}, \mathbf{b}^*)$ and forwards it to $\mathcal{A}$. When $\mathcal{A}$ makes a query $\mathbf{Chal}(\mathbf{m}_0, \mathbf{m}_1, (\mathbf{s}, \mathbf{e}))$, $\mathcal{B}$ makes one update query $(\mathbf{s}, \mathbf{e})$ and returns $(\mathbf{m}_0, \mathbf{m}_1)$. It receives an encryption $ct$ of $\mathbf{m}_\beta$ under $(\mathbf{A}, \mathbf{b} := \mathbf{b}^* + \mathbf{A}\mathbf{s} + \mathbf{e})$, makes no more update queries and receives $sk' := (\mathbf{s}^* + \mathbf{s} + \overline{\mathbf{s}}, \mathbf{e}^* + \mathbf{e} + \overline{\mathbf{e}})$ for random $(\overline{\mathbf{s}}, \overline{\mathbf{e}})$ as well as an encryption $ct^*$ of $\overline{\mathbf{s}}$. Reduction $\mathcal{B}$ answers $\mathcal{A}$'s **Chal** query with $(ct, sk')$ and returns $\mathcal{A}$'s output bit.

We give a proof of Shifty security of APS, since (i) we achieve better bounds for single-Chal security and (ii) we generalize to multiple challenge queries. (Note that single-challenge security does not imply multi-challenge security by a standard hybrid argument, since $sk'$ requires the secret key.)

**Theorem 2.** *Let $\varepsilon, \delta \in (0,1)$, $\ell > 0$, $q, p$ prime, $\sigma_c \geq \sigma \geq \sqrt{8\ln(2n(1+1/\varepsilon))/\pi}$, and $n, m > 0$, and $B \geq (\ell+1)y\sigma$ for $y = \sqrt{-2\log(\delta/(4n))}$. Then for any adversary $\mathcal{A}$ making at most $k$ Chal queries for secretkeys from $\overline{\mathcal{SK}}_{\ell-1}$, there is a multi HNF-adaptive extended LWE adversary $\mathcal{B}$ and LWE adversary $\mathcal{C}$ s.t.*

$$\mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) \leq k \cdot \mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,n,n,\sigma_c,B}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{LWE}}_{q,n,(k+1)n,\sigma/2}(\mathcal{C}) + (k+6)\frac{2\varepsilon}{1-\varepsilon} + \frac{\delta}{2} \ .$$

We give the proof[12] in Appendix G. From Corollary 1, setting $\gamma := \sigma_c$, $B := (\ell+1)y\sigma)$, and $m', k := \ell$, we immediately get:

**Corollary 2.** *Assume $q \geq 25$, $m \geq 16n + 4\log\log q$, $\sigma \geq \sqrt{8\ln(2(n+1)(1+1/\varepsilon))/\pi}$ and $\sigma_c > \sigma\sqrt{(1+n((\ell+1)y\sigma)^2)/2}$ for $y = \sqrt{-2\log(\delta/(4n))}$.*

*Let $m := 17n + 4\log\log q$, Then for any adversary $\mathcal{A}$ making at most $k$ **Chal** queries, there exist LWE adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) \leq 4kn \cdot \mathsf{Adv}^{\mathsf{LWE}}_{q,n,m,\sigma/2}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{LWE}}_{q,n,(k+1)n,\sigma/2}(\mathcal{C}) + \Big(8kn + k + 6\Big)\frac{2\varepsilon}{1-\varepsilon} + \frac{\delta}{2} \ .$$

We note that following the proof in [APS23] (and including a bad-key event) would result in a bound

$$\mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) \leq k \cdot \mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,(n+1),n,\sigma_c,B}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{LWE}}_{q,n,n,\sigma/2}(\mathcal{C}) + (k(2n+1)+6)2\varepsilon/(1-\varepsilon) + \delta/2 \ .$$

### 6.2 NIZKPoK

**From NIZK to NIZKPoK.** We give a black-box construction of a NIZKPoK from a PKE with a public key space $\mathcal{PK}$ (e.g. for our UKEM, $\mathcal{PK} = \mathbb{Z}_q^m$)[13], a random oracle $G$ with range $\mathcal{PK}$ and a NIZK (that uses a different random oracle $H$). The construction $\Pi$ can be found in Fig. 10; in Fig. 18 we give the additional algorithms alternative setup, simulation and extractor. In a nutshell, the NIZKPoK CRS $crs$ contains a random seed $s$ and a NIZK CRS $crs'$. The seed $s$ is expanded to a PKE public key $pk \in \mathcal{PK}$ via random oracle $G$. To prove that $(x, w) \in \mathcal{R}$, the prover encrypts $w$ under $pk$ and gives a NIZK proof

[12] For simplicity, we assume the Chal oracle in the Shifty-IND-CPA game directly accepts keys from the extended key space $\overline{\mathcal{SK}}_\ell$ rather than vectors of standard secret keys that are then summed up.

[13] Note that $\mathcal{PK}$ may contain values outside support of key generation.

**Construction** $\mathsf{PoK}[\mathsf{PKE}, \Pi, G, H]$

$\underline{\mathsf{Setup}}$

$s \leftarrow\!\!\$\ \{0,1\}^\kappa$
$crs' \leftarrow \Pi.\mathsf{Setup}$
$pp \leftarrow \mathsf{PKE.Setup}$
$crs \leftarrow (s, crs', pp)$
**return** $crs$

$\underline{\mathsf{P}^G(crs = (s, crs', pp), x, w)}$

$r \leftarrow\!\!\$\ \{0,1\}^*$
$pk \leftarrow G(s)$
$c \leftarrow \mathsf{Encrypt}(pp, pk, w; r)$
$\pi' \leftarrow \Pi.\mathsf{P}^H(crs', (x, pk, c), (r, w))$
**return** $\pi = (c, \pi')$

$\underline{\mathsf{V}^G(crs = (s, crs', pp), x, \pi)}$

$pk \leftarrow G(s)$
$(\pi', c) \leftarrow \pi$
**return** $\Pi.\mathsf{V}^H(crs', (x, pk, c), \pi')$

Fig. 10: The NIZKPoK construction making black box use of a PKE scheme PKE, a NIZK $\Pi$ and two random oracles $G$ and $H$, where $\kappa > 0$ and $G$ maps to $\mathcal{PK}$. The additional algorithms SetupAlt, S and Extract are given in Fig. 18.

(using random oracle $H$) that the ciphertext contains a witness for $x$.[14] More precisely, we require a NIZK for the following relation $\tilde{\mathcal{R}}$:

$$(\tilde{x} = (x, pp, pk, c), \tilde{w} = (w, r)) \in \tilde{\mathcal{R}} \iff c = \mathsf{Encrypt}(pp, pk, w; r) \wedge (x, w) \in \mathcal{R}.$$

Instantiating this with $\mathcal{R}$ used by our construction (cf. Eq. (2)), we get

$$\begin{aligned}(\tilde{x} = ((pp_{\mathsf{PKE}}, pk_i, pk_{i+1}), pp, pk, c), \tilde{w} = (\overline{sk}, r)) \in \tilde{\mathcal{R}} \\ \iff c = \mathsf{Encrypt}(pp, pk, \overline{sk}; r) \ \wedge\ \mathsf{pk}(\overline{sk}) = pk_{i+1} - pk_i \ \wedge\ \overline{sk} \in \mathcal{SK}, \quad (4)\end{aligned}$$

*Alternative setup.* The alternative setup algorithm samples $s$ and a key pair $(pk, sk)$. It programs $G(s)$ as $pk$ and sets $sk$ as the extraction trapdoor $td$. Outputs of the alternative setup must be indistinguishable from outputs of the normal setup. For this, public keys outputted by KeyGen must look like uniformly random elements of the public key space $\mathcal{PK}$. This property was previously defined in [BCP+23] where it is called *fuzziness* and proved that for Kyber it is implied by (M-)LWE. The following lemma follows from a straightforward reduction.

**Lemma 3.** *If PKE is fuzzy then NIZKPoK in Fig. 10 guarantees setup-indistinguishability. In particular, for any adversary $\mathcal{A}$, there exists a reduction $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{SI}}_{\mathsf{PoK}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Fuzzy}}_{\mathsf{PKE}}(\mathcal{B}) \ .$$

*Simulator.* The simulator S, instead of encrypting the witness, encrypts zeros (making the statement for the NIZK false). Thus, S uses the NIZK simulator to generate the NIZK proof. This is formalized by the following lemma.

**Lemma 4.** *If PKE is IND-CPA secure and NIZK is zero-knowledge, then NIZKPoK in Fig. 10 is zero-knowledge (Definition 16). In particular, for any adversary $\mathcal{A}$ there exist reductions $\mathcal{B}$, $\mathcal{B}'$ such that*

$$\mathsf{Adv}^{\mathsf{ZK\text{-}PoK}}_{\mathsf{PoK}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{ZK\text{-}NIZK}}_{\Pi}(\mathcal{B}') \ .$$

[14] Technically, PoK is defined with one RO. We use two for exposition, without loss of generality.

*Extractor.* The trapdoor $td$ for the extractor contains the secret key sampled by the alternative setup. Thus, a witness can be extracted from a proof $\pi$ by decrypting the ciphertext contained in $\pi$. If $\pi$ verifies then the contained NIZK verifies and by (simulation-)soundness of NIZK the extractor will return a correct witness. We prove the following lemma.

**Lemma 5.** *If PKE is $\delta$-correct and NIZK is adaptively simulation-sound, then NIZKPoK in Fig. 10 is straightline simulation-sound (Definition 17). More precisely, if PKE is $\delta$-correct then for any adversary $\mathcal{A}$ there exist a reduction $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{Ext}}_{\mathsf{PoK}}(\mathcal{A}) \leq \delta + \mathsf{Adv}^{\mathsf{SS}}_{\Pi}(\mathcal{B}) \ .$$

**Instantiation from Lattices.** To give a lattice-based instantiation of NIZKPoK from Fig. 10, we instantiate $\mathsf{PKE}$ with Kyber [BDK+18] with $\mathcal{PK} = R_q^n$. Fuzziness follows from the Module LWE assumption [BCP+23, Cor. 1]. (Following APS, we propose to instantiate the kh-PKE from Section 6.1 also in the module setting.) We let $pk_{\mathsf{ky}} = (\mathbf{A}', \mathbf{b}')$, $c_{\mathsf{ky}} = (\mathbf{C}, \mathbf{c})$ and $r_{\mathsf{ky}} = (\mathbf{X}, \mathbf{E}, \mathbf{f})$ denote a Kyber public key, ciphertext and randomness, respectively. For more concrete parameters and distribution choices, we refer to [BDK+18]. Hence, formally, the NIZK relation from (4) becomes the following:

$$\begin{aligned}(\tilde{x} = ((\mathbf{A}, \mathbf{b}_i, \mathbf{b}_{i+1}), pk_{\mathsf{ky}}, c_{\mathsf{ky}}), \tilde{w} = ((\mathbf{s}, \mathbf{e}), r_{\mathsf{ky}})) \in \tilde{\mathcal{R}} \\ \iff c_{\mathsf{ky}} = \mathsf{Kyber.Encrypt}(pk_{\mathsf{ky}}, (\mathbf{s}, \mathbf{e}); r_{\mathsf{ky}}) \\ \wedge \ \mathbf{As} + \mathbf{e} = \mathbf{b}_{i+1} - \mathbf{b}_i \ \wedge \ \|\mathbf{s}\|_2 \leq B \wedge \|\mathbf{e}\|_2 \leq B\end{aligned}$$

However, the secret key component $\mathbf{e}$ is implicitly defined as $\mathbf{b}_{i+1} - \mathbf{b}_i - \mathbf{As}$, thus it suffices to only encrypt $\mathbf{s}$, leading to the following simpler relation:

$$\begin{aligned}&(\tilde{x} = ((\mathbf{A}, \mathbf{b}_i, \mathbf{b}_{i+1}), pk_{\mathsf{ky}}, c_{\mathsf{ky}}), \tilde{w} = ((\mathbf{s}, \mathbf{e}), r_{\mathsf{ky}})) \in \tilde{\mathcal{R}} \\ &\iff c_{\mathsf{ky}} = \mathsf{Kyber.Encrypt}(pk_{\mathsf{ky}}, \mathbf{s}; r_{\mathsf{ky}}) \ \wedge \ \|\mathbf{s}\|_2 \leq B \wedge \|\mathbf{b}_{i+1} - \mathbf{b}_i - \mathbf{As}\|_2 \leq B \\ &\iff c_{\mathsf{ky}} = (\mathbf{C}, \mathbf{c}) = (\mathbf{XA}' + \mathbf{E}, \mathbf{Xb}' + \mathbf{f} + \lfloor q/p \rfloor \cdot [\mathbf{s} \mid \mathbf{e}]) \\ &\qquad \wedge \|\mathbf{X}\|_2, \|\mathbf{E}\|_2, \|\mathbf{f}\|_2 \leq B' \ \wedge \ \|\mathbf{s}\|_2, \|\mathbf{b}_{i+1} - \mathbf{b}_i - \mathbf{As}\|_2 \leq B\end{aligned} \tag{5}$$

Proving a statement for $\tilde{\mathcal{R}}$ therefore requires proving 2 norm bounds for matrices, 3 norm bounds for vectors and $n^2 + n$ linear equations over $R_q$. In contrast, [APS23] requires 4 norm bounds for matrices, 4 norm bounds for vectors and $2n^2 + 2n$ linear equations. As [APS23], we suggest to use the Lyubashevsky-Nguyen-Plançon framework [LNP22]. We note that we need adaptive simulation-soundness for our NIZK, which has been argued to hold for [LNP22] in [BBP23, Lemma 2].

## 7 Parameters

Following [APS23], we provide concrete parameters for the module variant of our scheme; we replace the Gaussian distribution by a centered binomial distribution $B_\eta$ defined in [BDK+18] with $\eta > 0$ and elements contained in $[-\eta, \eta]$. Concretely, we suggest to use $\eta = 2$, as done in [BDK+18]. We let $R = \mathbb{Z}[X]/(X^d + 1)$, $R_q = R/qR$ and we consider the base ring to be $R$ instead of $\mathbb{Z}$. For $p$ prime, the message space is $R_p$, which is of size $p^d$. When encrypting vectors (as done when encrypting secret keys), the message space is $R_p^n$.

Table 2: Size comparison for different parameter sets, where $\lambda$ is the security level, $\eta = 2$, $p = 5$ and $\delta \approx 0$ for all schemes. All sizes are without potential optimizations or compression.

| | $\lambda$ | $q$ | $n$ | $d$ | $B$ | $\ell$ | $\lvert pk\rvert$ | $\lvert ct\rvert + \lvert mt\rvert$ |
|---|---|---|---|---|---|---|---|---|
| [APS23] | 115 | $\approx 2^{21}$ | 3 | 256 | 1536 | $2^5$ | 4.1 KB | 61.1 KB |
| [this work] | 115 | $\approx 2^{21}$ | 3 | 256 | 1536 | $2^5$ | 2.0 KB | 44.9 KB |
| [APS23] | 125 | $\approx 2^{26}$ | 4 | 256 | 2048 | $2^{10}$ | 6.7 KB | 88.7 KB |
| [this work] | 125 | $\approx 2^{26}$ | 4 | 256 | 2048 | $2^{10}$ | 3.4 KB | 60.1 KB |
| [APS23] | 105 | $\approx 2^{31}$ | 2 | 512 | 2048 | $2^{15}$ | 8.0 KB | 81.0 KB |
| [this work] | 105 | $\approx 2^{31}$ | 2 | 512 | 2048 | $2^{15}$ | 4.0 KB | 57.2 KB |
| [APS23] | 137 | $\approx 2^{36}$ | 3 | 512 | 3072 | $2^{20}$ | 13.9 KB | 130.7 KB |
| [this work] | 137 | $\approx 2^{36}$ | 3 | 512 | 3072 | $2^{20}$ | 6.9 KB | 86.9 KB |

We adopt the parameters from [APS23], for which they have computed security and correctness for various numbers of updates.[15] The concrete parameters and sizes are given in Table 2. The size of an element in $R_q$ is computed as $\log(q)d$. We can also use the bit dropping technique that is used in Kyber to obtain more compact ciphertexts. In Table 2, however, we naively compute the ciphertext sizes without any optimizations or compression for our scheme and that of [APS23]. We explain how we computed the numbers below.

**Our Sizes.** Our UKEM ciphertext, which we denote by the components $(c, d) \in R_q^{1\times n} \times R_q$, and key $K$ for public key $pk = (\mathbf{A}, \mathbf{b})$ are computed as follows:

$$c = \mathbf{x}^\top \mathbf{A} + \mathbf{e}^\top, \quad d = \mathbf{x}^\top \mathbf{b} + f + \lfloor q/p \rfloor \cdot m, \quad K = H_1(pk, m), \tag{6}$$

where $\mathbf{x}, \mathbf{e} \in R_q^n, f \in R_q$ are derived (deterministically) from $H_2(pk, m)$ for a random $m \in R_p$. Since the secret-key shift $\mathbf{s}, \mathbf{e} \in \text{support}(B_\eta^n)$ is also computed via a random oracle, we do not need to encrypt it explicitly in the ciphertext. However, we do need to encrypt $\mathbf{s}$ as part of the NIZKPoK in the member tag.[16] Hence, we get an additional ciphertext $(c_0, d_0) \in R_q^{n\times n} \times R_q^n$ for public key $pk_{crs} = (\mathbf{A}_0, \mathbf{b}_0)$[17] computed as

$$c_0 = \mathbf{X}_0\mathbf{A}_0 + \mathbf{E}_0, \quad d_0 = \mathbf{X}_0\mathbf{b}_0 + \mathbf{f}_0 + \lfloor q/p \rfloor \cdot \mathbf{s}, \tag{7}$$

where $\mathbf{X}_0, \mathbf{E}_0 \in R_q^{n\times n}$, $\mathbf{f}_0 \in R_q^n$; and $\mathbf{s} \in B_\eta^n$ is represented as an element in $R_p^n$. Note that we now have matrices since we encrypt a vector. The member tag then consists of $(c_0, d_0, \pi)$, where $\pi$ is the proof for the statement in Eq. (5). The combined public key consists of $(\mathbf{A}, \mathbf{b}, \mathbf{A}_0, \mathbf{b}_0)$, where we derive all components except $\mathbf{b}$ via a seed, hence we compute its size by $\log(q)dn + 256$ bits.

We give estimates on the size of $\pi$ using the LaZer library [LSS24]. LaZer lets us specify lattice relations based on which it automatically creates parameters of a proof system

[15] We have corrected the bit security levels using the lattice estimator [APS15] (commit 162c505). The numbers given in [APS23] were incorrect, which was confirmed by the authors. Correctness after $\ell$ updates was shown by [APS23] using the Kyber script in `https://github.com/pq-crystals/security-estimates`.

[16] Technically, our kh-PKE abstraction requires encrypting both $\mathbf{s}$ and $\mathbf{e}$. We note that one can compute $\mathbf{e}$ from $\mathbf{s}$ and the public key $(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$. Thus, we only count encrypting $\mathbf{s}$ towards the ciphertext size.

[17] Our scheme could potentially be further optimized by using the original Kyber parameters here, as outlined in Section 6.2. However, since LaZer does not allow to specify such choices, we use the same parameters as for the key-homomorphic PKE.

along with proof sizes, using either the LNP framework [LNP22] or LaBRADOR [BS23]. We run LaZer on our relation from Section 6.2 for all four parameter sets and obtain proof sizes from 34.1 KB (smallest parameter set) to 50.2 KB (biggest parameter set). Since we consider a centered binomial distribution, we set the $l_2$-norm bounds to $B = B' = n \cdot \eta \cdot d$, which is the worst case bound (in fact a tighter bound can be used, resulting in smaller parameters).

The final sizes for four different parameter sets are summarized in Table 2.

**APS Sizes.** We now also recall how the UKEM scheme from [APS23] is constructed. Note that their scheme (when adding a NIZK) achieves IND-CU-CCA security which is a weaker notion than our IND-CCA definition. Still, our sizes improve upon theirs substantially. We explain the reasons for this below.

Since our scheme is based on the underlying APS UPKE, the APS ciphertext $(c, d)$ is the same as in Eq. (6), however, for notational consistency we denote their public key by $(\mathbf{A}_0, \mathbf{b}_0)$. Then the ciphertext and UKEM key look as follows:

$$c = \mathbf{x}^\top \mathbf{A}_0 + \mathbf{e}^\top, \quad d = \mathbf{x}^\top \mathbf{b}_0 + f + \lfloor q/p \rfloor \cdot m, \quad K = H_1(m, c, d).$$

where $\mathbf{x}, \mathbf{e} \in R_q^n, f \in R_q$ are derived (deterministically) from $H_0(pk, m)$ for a random $m \in R_p$. Since APS use the long syntax, we also need to include an explicit update ciphertext, computed via their dedicated update algorithms, and we obtain a second ciphertext $(c_0, d_0)$:

$$c_0 = \mathbf{X}_0 \mathbf{A}_0 + \mathbf{E}_0, \quad d_0 = \mathbf{X}_0 \mathbf{b}_0 + \mathbf{f}_0 + \lfloor q/p \rfloor \cdot \mathbf{s},$$

where variables (and thus sizes) are the same as in Eq. (7).

Finally, in order to achieve security against chosen updates, APS uses a NIZK proving that two ciphertexts encrypt the same secret key shift. Hence, they compute a another ciphertext $(c_1, d_1)$ just as $(c_0, d_0)$, but to an additional public key $(\mathbf{A}_1, \mathbf{b}_1)$, and add a NIZK proof $\pi_{\text{APS}}$ for the following relation $\tilde{\mathcal{R}}_{\text{APS}}$:

$$\begin{aligned}(\tilde{x} = (\mathbf{A}_0, \mathbf{A}_1, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b},\, c_0, d_0, c_1, d_1), \tilde{w} = (\mathbf{X}_0, \mathbf{X}_1, \mathbf{E}_0, \mathbf{E}_1, \mathbf{f}_0, \mathbf{f}_1, \mathbf{s}, \mathbf{e})) \in \tilde{\mathcal{R}}_{\text{APS}} \\ \iff c_0 = \mathbf{X}_0 \mathbf{A}_0 + \mathbf{E}_0 \ \wedge \ d_0 = \mathbf{X}_0 \mathbf{b}_0 + \mathbf{f}_0 + \lfloor q/p \rfloor \cdot \mathbf{s} \\ \wedge\ c_1 = \mathbf{X}_1 \mathbf{A}_1 + \mathbf{E}_1 \ \wedge \ d_1 = \mathbf{X}_1 \mathbf{b}_1 + \mathbf{f}_1 + \lfloor q/p \rfloor \cdot \mathbf{s} \\ \wedge\ \|\mathbf{X}_0\|_2, \|\mathbf{X}_1\|_2, \|\mathbf{E_0}\|_2, \|\mathbf{E}_1\|_2, \|\mathbf{f}_0\|_2, \|\mathbf{f}_1\|_2 \leq B' \\ \wedge\ \|\mathbf{s}\|_2, \|\mathbf{b} - \mathbf{b}_0 - \mathbf{A}_0 \mathbf{s}\|_2 \leq B.\end{aligned}$$

Here $B$ and $B'$ are the same as in our scheme and we give estimates on the NIZK sizes using LaZer. Since we have more relations here, the proof sizes are larger. In particular, they range from 42.2 KB (smallest parameter set) to 66.5 KB (biggest parameter set).

The total ciphertext (with member tag) is then $(c, d, c_0, d_0, c_1, d_1, \pi_{\text{APS}})$. The additional ciphertext component adds another 8.1 KB (smallest parameter set) to 27.5 KB (biggest parameter set). Moreover, public keys consist of $\mathbf{b}_0$, $\mathbf{b}_1$ and a seed to derive $\mathbf{A}_0$ and $\mathbf{A}_1$, which explains the increase by around a factor 2 compared to our scheme.

**Acknowledgments.** This work was funded by the Vienna Science and Technology Fund (WWTF) [10.47379/VRG18002] and by the Austrian Science Fund (FWF) [10.55776/F8515-N]. The authors would like to thank Calvin Abou Haidar for valuable comments and help on computing lattice parameters.

## References


[AAN+22] Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. CoCoA: Concurrent continuous group key agreement. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 815–844. Springer, Cham, May / June 2022.

[ACDT20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Cham, August 2020.

[ACDT21] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of MLS. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1463–1483. ACM Press, November 2021.

[ACJM20] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 261–290. Springer, Cham, November 2020.

[AFM24] Joël Alwen, Georg Fuchsbauer, and Marta Mularczyk. Updatable public-key encryption, revisited. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 346–376. Springer, Cham, May 2024.

[ALP22] Calvin Abou Haidar, Benoît Libert, and Alain Passelègue. Updatable public key encryption from DCR: Efficient constructions with stronger security. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 11–22. ACM Press, November 2022.

[AMT23] Joël Alwen, Marta Mularczyk, and Yiannis Tselekounis. Fork-resilient continuous group key agreement. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 396–429. Springer, 2023.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[APS23] Calvin Abou Haidar, Alain Passelègue, and Damien Stehlé. Efficient updatable public-key encryption from lattices. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 342–373. Springer, Singapore, December 2023.

[AW23] Kyoichi Asano and Yohei Watanabe. Updatable public key encryption with strong CCA security: Security analysis and efficient generic construction. *IACR Cryptol. ePrint Arch.*, page 976, 2023.

[BBP23] Johannes Blömer, Jan Bobolz, and Laurens Porzenheim. A generic construction of an anonymous reputation system and instantiations from lattices. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 418–452. Springer, Singapore, December 2023.

[BBR+23] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023.

[BCP+23] Hugo Beguinet, Céline Chevalier, David Pointcheval, Thomas Ricosset, and Mélissa Rossi. GeT a CAKE: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 23International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *LNCS*, pages 516–538. Springer, Cham, June 2023.

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 353–367. IEEE, 2018.

[BF11] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 1–16. Springer, Berlin, Heidelberg, March 2011.

[BS23] Ward Beullens and Gregor Seiler. LaBRADOR: Compact proofs for R1CS from module-SIS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 518–548. Springer, Cham, August 2023.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271. Springer, Berlin, Heidelberg, May 2003.

[DHW23] Yevgeniy Dodis, Shai Halevi, and Daniel Wichs. Security with functional re-encryption from CPA. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part II*, volume 14370 of *LNCS*, pages 279–305. Springer, Cham, November / December 2023.

[DKW21] Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Updatable public key encryption in the standard model. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 254–285. Springer, Cham, November 2021.

[EJKM22] Edward Eaton, David Jao, Chelsea Komlo, and Youcef Mokrani. Towards post-quantum key-updatable public-key encryption via supersingular isogenies. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 461–482. Springer, Cham, September / October 2022.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Cham, November 2017.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 159–188. Springer, Cham, May 2019.

[JS18] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 33–62. Springer, Cham, August 2018.

[KPPW+21] Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. In *2021 IEEE Symposium on Security and Privacy*, pages 268–284. IEEE Computer Society Press, May 2021.

[LNP22] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 71–101. Springer, Cham, August 2022.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 382–400. Springer, Berlin, Heidelberg, February 2010.

[LSS24] Vadim Lyubashevsky, Gregor Seiler, and Patrick Steuer. The lazer library: Lattice-based zero knowledge and succinct proofs for quantum-safe privacy. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 3125–3137, New York, NY, USA, 2024. Association for Computing Machinery.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

[PM16] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/, November 2016.

[PR18] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 3–32. Springer, Cham, August 2018.

## A PKE Definitions

For completeness, we first define spreadness and CPA security for PKE.

**Definition 9 ($\gamma$-spreadness [HHK17]).** *Let $\gamma > 0$. We say that a (key-homomorphic) PKE* $\mathsf{PKE}$ *is $\gamma$-spread if for all $m \in \mathcal{M}$, $c \in \{0,1\}^*$, $pp \in \mathsf{PKE.Setup}$ and $(pk, sk) \in \mathsf{PKE.KeyGen}(pp)$, we have $\Pr[\mathsf{Encrypt}(pk, m) = c] \leq 2^{-\gamma}$, where the probability is taken over the random coins of* $\mathsf{Encrypt}$.

**Game PKE IND-CPA Security**

$\underline{\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A})}$
$b \leftarrow\!\!\$\ \{0,1\}$
$pp \leftarrow \mathsf{Setup}$
$(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$
$b' \leftarrow \mathcal{A}^{\mathsf{Chal}}(pk)$
**return** $(b = b')$

$\underline{\mathbf{Oracle}\ \mathbf{Chal}(m_0, m_1)}$
$c \leftarrow \mathsf{Encrypt}(pp, pk, m_b)$
**return** $c$

Fig. 11: The IND-CPA security game for PKE $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ and adversary $\mathcal{A}$.

**Definition 10 (PKE IND-CPA).** *Let game* IND-CPA *for* PKE *be as defined in Fig. 11. The advantage of an adversary $\mathcal{A}$ in that game is defined as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) := 2\Pr\left[\mathrm{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) = 1\right] - 1.$$

We further recall the definition of a fuzzy PKE, originally defined for KEMs [BCP+23].

**Definition 11 (Fuzzy PKE).** *Let* PKE *be a PKE scheme and let $pp \leftarrow \mathsf{Setup}$ such that $pp$ defines a public key space $\mathcal{PK}$. We define the advantage of an adversary $\mathcal{A}$ in breaking fuzziness of* PKE *as*

$$\mathsf{Adv}^{\mathsf{Fuzzy}}_{\mathsf{PKE}}(\mathcal{A}) := \left|\Pr[\mathcal{A}(pp, pk) = 1 \mid (pk, sk) \leftarrow \mathsf{KeyGen}(pp)] - \Pr[\mathcal{A}(pp, pk) = 1 \mid pk \leftarrow\!\!\$\ \mathcal{PK}]\right|.$$

## B Proof Systems

A *Non-Interactive Proof System* (NIPS) for an NP relation $\mathcal{R}$ in the random oracle (RO) model is a triple of algorithms, all given access to an RO evaluation oracle $H$:

**Setup.** $crs \leftarrow \mathsf{Setup}^H$ outputs a common random string $crs$.
**Setup.** $crs \leftarrow \mathsf{Setup}(idx \in I)$ outputs a common random string $crs$.
**Prove.** $\pi \leftarrow \mathsf{P}^H(crs, x, w)$ on input a statement $x$ and a witness $w$, outputs a proof $\pi$ that $(x, w) \in \mathcal{R}$.
**Verification.** $0/1 \leftarrow \mathsf{V}^H(crs, x, \pi)$ verifies a proof $\pi$ of statement $x$, i.e., that $(x, w) \in \mathcal{R}$ for some $w$.

*Completeness.* For a NIPS we only define completeness. It requires that proofs outputted by P are accepted by V for any statements generated by a (possibly undounded) adversary *adaptively* depending on the random $crs$.[18]

**Definition 12 (NIPS Completeness).** *Let $\Pi$ be a NIPS and $\mathsf{Exp}^{\mathsf{Complete}}_{\Pi}$ be as defined in Fig. 12. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against completeness of $\Pi$ is defined as*

$$\mathsf{Adv}^{\mathsf{Complete}}_{\Pi}(\mathcal{A}) := \Pr\left[\mathsf{Exp}^{\mathsf{Complete}}_{\Pi}(\mathcal{A}) = 1\right].$$

*We shorthand call a NIPS (and later NIZK, NIZKPoK) $\delta$-complete if $\mathsf{Adv}^{\mathsf{Complete}}_{\Pi}(\mathcal{A}) \leq \delta$.*

[18] One sometimes considers a weaker notion of non-adaptive completeness that holds for all $x$ and a $crs$ sampled independently from $x$. The following NIPS separates the two notions: Let $\mathcal{R}'$ be an NP relation. Define relation $\mathcal{R}$ as the set of $(x = (a, x'), w)$ where $a$ is arbitrary and $(x', w) \in \mathcal{R}$. Take any adaptively-complete NIPS' for $\mathcal{R}'$ and construct a NIPS for $\mathcal{R}$ where $\mathsf{P}((a, x'), w)$ outputs $\mathsf{P}(x', w)$ if $a \neq crs$ *and* $\perp$ *otherwise*. An adaptive adversary can break completeness with any statement $(crs, *)$.

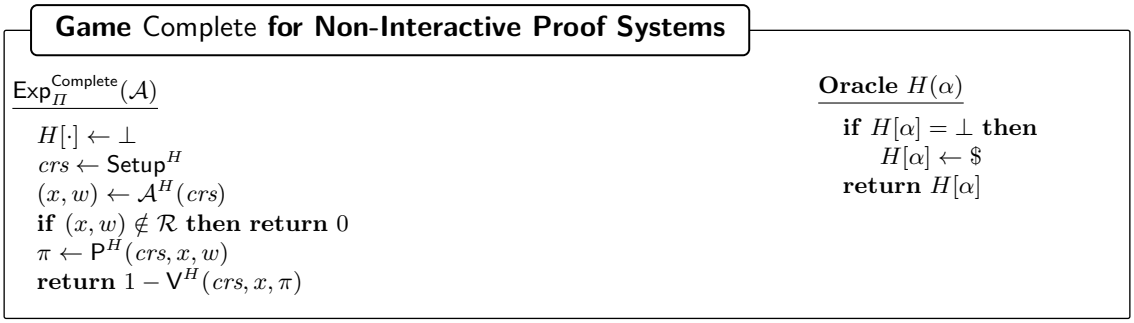



Fig. 12: Completeness game for NIPS $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ for NP relation $\mathcal{R}$.

**Non-Interactive Zero-Knowledge (NIZK).** In the programmable random oracle (RO) model, a *Non-Interactive Zero Knowledge proof system* (NIZK) for an NP relation $\mathcal{R}$ is a NIPS for $\mathcal{R}$ with the following additional algorithm, given access to an RO evaluation oracle $H$ and an RO programming oracle $\mathcal{P}$.

**Simulation.** $\pi \leftarrow \mathsf{S}^{H,\mathcal{P}}(crs, x)$ outputs a simulated proof $\pi$ for statement $x$.[19]

We define two security properties of a NIZK: adaptive simulation-soundness and zero-knowledge.

**Definition 13 (NIZK Adaptive Simulation-Soundness).** *Let $\Pi$ be a NIZK and $\mathsf{Exp}^{\mathsf{SS}}_{\Pi}$ be as defined in Fig. 13. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against adaptive simulation-soundness of $\Pi$ is defined as*

$$\mathsf{Adv}^{\mathsf{SS}}_{\Pi}(\mathcal{A}) \coloneqq \Pr\left[\mathsf{Exp}^{\mathsf{SS}}_{\Pi}(\mathcal{A}) = 1\right].$$

**Definition 14 (NIZK Zero-Knowledge).** *Let $\Pi$ be a NIZK and $\mathsf{Exp}^{\mathsf{ZK\text{-}NIZK}}_{\Pi}$ be as defined in Fig. 14. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against zero-knowledge of $\Pi$ is defined as*

$$\mathsf{Adv}^{\mathsf{ZK\text{-}NIZK}}_{\Pi}(\mathcal{A}) \coloneqq 2\Pr\left[\mathsf{Exp}^{\mathsf{ZK\text{-}NIZK}}_{\Pi}(\mathcal{A}) = 1\right] - 1.$$

**NIZK Proof of Knowledge (NIZKPoK).** In the programmable random oracle model, a *NIZK Proof of Knowledge* (NIZKPoK) for an NP relation $\mathcal{R}$ is a NIPS for $\mathcal{R}$ with three additional algorithms, both given access to an RO evaluation oracle $H$ and an RO programming oracle $\mathcal{P}$:

**Alternative Setup.** $(crs, td) \leftarrow \mathsf{SetupAlt}^{H,\mathcal{P}}$ outputs a common random string $crs$ with trapdoor $td$.

**Simulation.** $\pi \leftarrow \mathsf{S}^{H,\mathcal{P}}(td, x)$ outputs a simulated proof $\pi$ for statement $x$.

**Extraction.** $w \leftarrow \mathsf{Extract}^{H,\mathcal{P}}(td, x, \pi)$ outputs a witness $w$ for theorem $x$ given trapdoor $td$ and a proof $\pi$ for $x$.

*Remark 1.* In contrast to NIZK, we define NIZKPoK with an alternative setup that outputs a trapdoor for the simulator and extractor. For simplicity, we use the same trapdoor in both algorithms. This is the setting needed for our construction.[20]

[19] For simplicity, we define NIZK with CRS but without an alternative setup algorithm outputting a CRS with a trapdoor for the simulator. This is the setting required for the lattice-based NIZK we use; the simulator only needs to program the RO.

[20] In fact, our simulator does not need a trapdoor but we need the zero-knowledge definition to use the trapdoored setup.

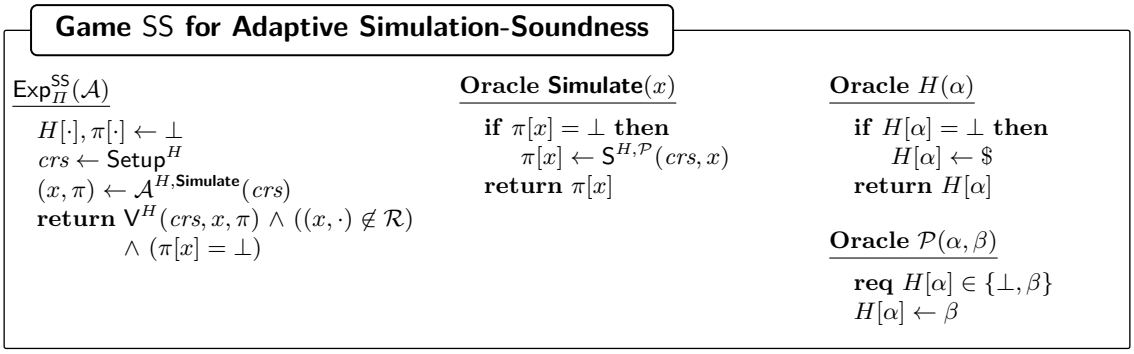



Fig. 13: The adaptive simulation-soundness security game for NIZK $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V}, \mathsf{S})$ for NP relation $\mathcal{R}$.

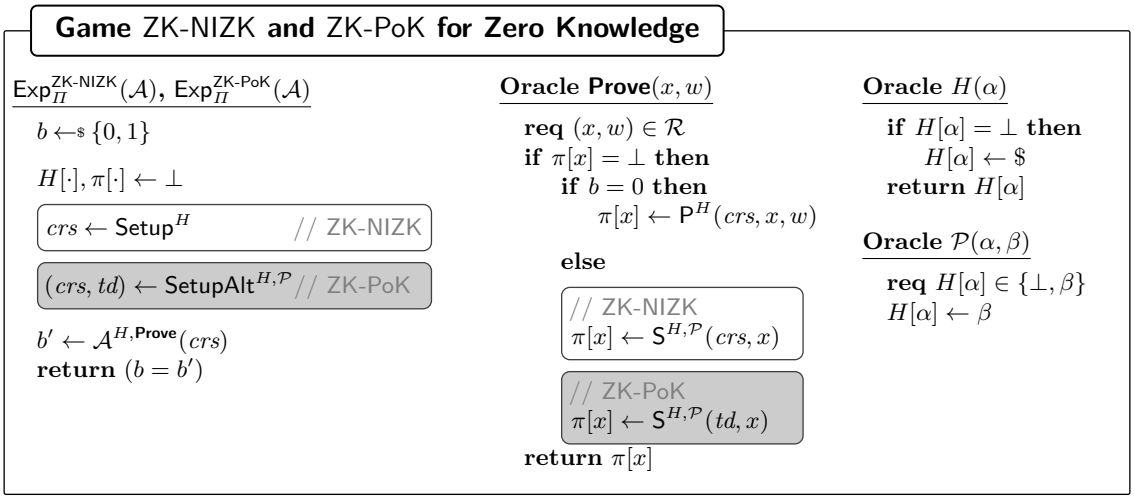



Fig. 14: The adaptive zero knowledge security game for NIZK and NIZKPoK for NP relation $\mathcal{R}$.

In addition to NIZK properties, a NIZKPoK should provide the following: setup-indistinguishability and straightline simulation-extractability. We note that the latter implies soundness. We also modify the zero-knowledge definition by replacing $\mathsf{Setup}$ with $\mathsf{SetupAlt}$.

**Definition 15 (NIZKPoK Setup Inditstinguishability).** *Let $\Pi$ be a NIZKPoK and $\mathsf{Exp}^{\mathsf{SI}}_{\Pi}$ be as defined in Fig. 15. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against setup indistinguishability of $\Pi$ is defined as*

$$\mathsf{Adv}^{\mathsf{SI}}_{\Pi}(\mathcal{A}) \coloneqq 2 \Pr\left[\mathsf{Exp}^{\mathsf{SI}}_{\Pi}(\mathcal{A}) = 1\right] - 1.$$

**Definition 16 (NIZKPoK Zero-Knowledge).** *Let $\Pi$ be a NIZKPoK and $\mathsf{Exp}^{\mathsf{ZK\text{-}PoK}}_{\Pi}$ be as defined in Fig. 14. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against zero-knowledge of $\Pi$ is defined as*

$$\mathsf{Adv}^{\mathsf{ZK\text{-}PoK}}_{\Pi}(\mathcal{A}) \coloneqq 2 \Pr\left[\mathsf{Exp}^{\mathsf{ZK\text{-}PoK}}_{\Pi}(\mathcal{A}) = 1\right] - 1.$$

**Game SI for Setup Indistinguishability**

$\underline{\mathsf{Exp}^{\mathsf{SI}}_{\Pi}(\mathcal{A})}$
$b \leftarrow\!\!\$ \{0,1\}$
$H[\cdot], \pi[\cdot] \leftarrow \bot$
$crs_0 \leftarrow \mathsf{Setup}^H$
$(crs_1, *) \leftarrow \mathsf{SetupAlt}^{H,\mathcal{P}}$
$b' \leftarrow \mathcal{A}^H(crs_b)$
**return** $(b = b')$

**Oracle** $\underline{H(\alpha)}$
**if** $H[\alpha] = \bot$ **then**
$H[\alpha] \leftarrow \$$
**return** $H[\alpha]$

**Oracle** $\underline{\mathcal{P}(\alpha, \beta)}$
**req** $H[\alpha] \in \{\bot, \beta\}$
$H[\alpha] \leftarrow \beta$

Fig. 15: The setup-indistinguishability game for NIZKPoK $\Pi$ for NP relation $\mathcal{R}$ with setup Setup and trapdoor setup SetupAlt.

**Game Ext for Straight-Line Simulation-Extractability**

$\underline{\mathsf{Exp}^{\mathsf{Ext}}_{\Pi}(\mathcal{A})}$
$result \leftarrow 0$
$H[\cdot], \pi[\cdot] \leftarrow \bot$
$(crs, td) \leftarrow \mathsf{SetupAlt}^{H,\mathcal{P}}$
$\mathcal{A}^{H,\mathsf{Simulate},\mathsf{Test}}(crs)$
**return** $result$

**Oracle** $\underline{\mathsf{Test}(x, \pi)}$
**req** $\mathsf{V}^H(crs, x, \pi) = 1$
$w \leftarrow \mathsf{Extract}^{H,\mathcal{P}}(td, x, \pi)$
**if** $\pi[x] = \bot \wedge (x, w) \notin \mathcal{R}$ **then**
$result \leftarrow 1$
**return** $w$

**Oracle** $\underline{\mathsf{Simulate}(x)}$
**if** $\pi[x] = \bot$ **then**
$\pi[x] \leftarrow \mathsf{S}^{H,\mathcal{P}}(td, x)$
**return** $\pi[x]$

**Oracle** $\underline{H(\alpha)}$
**if** $H[\alpha] = \bot$ **then**
$H[\alpha] \leftarrow \$$
**return** $H[\alpha]$

**Oracle** $\underline{\mathcal{P}(\alpha, \beta)}$
**req** $H[\alpha] \in \{\bot, \beta\}$
$H[\alpha] \leftarrow \beta$

Fig. 16: The extractability security game for NIZKPoK $\Pi = (\mathsf{Setup}, \mathsf{S}, \mathsf{P}, \mathsf{V}, \mathsf{SetupAlt}, \mathsf{Extract})$ for NP relation $\mathcal{R}$ and random oracle $H$.

**Definition 17 (NIZKPoK Straightline Simulation-Extractability).** *Let $\Pi$ be a NIZKPoK and $\mathsf{Exp}^{\mathsf{Ext}}_{\Pi}$ be as defined in Fig. 16. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against setup indistinguishability of $\Pi$ is defined as*

$$\mathsf{Adv}^{\mathsf{Ext}}_{\Pi}(\mathcal{A}) := \Pr\left[\mathsf{Exp}^{\mathsf{Ext}}_{\Pi}(\mathcal{A}) = 1\right].$$

## C UKEM Robustness

Correctness, as defined in Section 4, concerns honest executions of the scheme. To capture correctness in the presence of adversarially chosen updates we define the notion of robustness. Intuitively, in UKEM applications the adversary may inject arbitrary ciphertexts to an honest Alice and arbitrary corresponding updates to an honest Bob. Robustness guarantees that (if Alice and Bob accept the ciphertexts and updates) then Alice's updated secret key decrypt Bob's ciphertexts encrypted to the updated public key. In other words, the adversary cannot make honest parties accept a non-functional key pair.

**Definition 18 (UKEM Robustness).** *Let $\mathrm{Exp}^{\mathsf{Rob}}_{\mathsf{UKEM}}(\mathcal{A})$ be as defined in Fig. 17. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ in that game is defined as*

$$\mathsf{Adv}^{\mathsf{Rob}}_{\mathsf{UKEM}}(\mathcal{A}) := \Pr\left[\mathrm{Exp}^{\mathsf{Rob}}_{\mathsf{UKEM}}(\mathcal{A}) = 1\right].$$

**Game** Rob **for Robustness**

$\underline{\mathbf{Exp}^{\mathsf{Rob}}_{\mathsf{UKEM}}(\mathcal{A})}$
$(j, win) \leftarrow (0, 0)$
$pp \leftarrow \mathsf{Setup}$
$(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(pp)$
$\mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}(pp, pk_0, sk_0)$
**return** $win$

$\underline{\textbf{Oracle Dec}(i, c, pk)}$
**req** $sk_i \neq \bot$
$j \leftarrow j + 1$
$(K, sk_j) \leftarrow \mathsf{Decaps}(pp, sk_i, c, pk)$
**req** $sk_j \neq \bot$
$pk_j \leftarrow pk$
**return** $(K, sk_j)$

$\underline{\textbf{Oracle Enc}(i)}$
**req** $sk_i \neq \bot$
$(K, c, pk, mt) \leftarrow \mathsf{Encaps}(pp, pk_i)$
$(K', sk') \leftarrow \mathsf{Decaps}(pp, sk_i, c, pk)$
**if** $K = \bot \vee K \neq K'$ **then**
  $win \leftarrow 1$
**if** $\neg\mathsf{Verify}_{\mathsf{mt}}(pp, pk_i, pk, mt)$ **then**
  $win \leftarrow 1$

Fig. 17: Robustness game for $\mathsf{UKEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps}, \mathsf{Verify}_{\mathsf{mt}})$ and adversary $\mathcal{A}$. We use **req** *condition* to denote that if condition is false, then the current function, and any function calling it, stops and returns $\bot$.

**Construction** PoK Alternative Setup, Simulator and Extractor

$\underline{\mathsf{SetupAlt}^{G,H,\mathcal{P}}}$
$s \leftarrow\!\!\$\ \{0,1\}^\kappa$
$crs' \leftarrow \Pi.\mathsf{Setup}$
$crs \leftarrow (s, crs')$
$(pk, sk) \leftarrow \mathsf{PKE.KeyGen}$
$\mathcal{P}(s, pk)$// Program $G(s) \leftarrow pk$
**return** $(crs, td = (crs, sk))$

$\underline{\mathsf{S}^{G,H,\mathcal{P}}(td, x)}$
$(crs, *) \leftarrow td$
$(s, crs') \leftarrow crs$
$pk := G(s)$
$c \leftarrow \mathsf{Encrypt}(pk, 0^\ell)$
$x' := (x, pk, c)$
$\pi' \leftarrow \Pi.\mathsf{S}(crs', x')$
**return** $\pi = (c, \pi')$

$\underline{\mathsf{Extract}^{G,H,\mathcal{P}}(td, x, \pi)}$
$(*, sk) \leftarrow td$
$(c, *) \leftarrow \pi$
$w \leftarrow \mathsf{PKE.Decrypt}(sk, c)$
**return** $w$

Fig. 18: The alternative setup, simulator and extraction algorithms for NIZKPoK construction $\mathsf{PoK}[\mathsf{PKE}, \Pi, G, H]$ in Fig. 10. We assume $w$ has constant length $\ell$.

*Further, for $\ell > 0$ and $\delta > 0$, we say that* UKEM *is $(\ell, \delta)$-robust if for any (possibly unbounded) adversary $\mathcal{A}$ that creates a tree of depth at most $\ell$, we have*

$$\Pr\left[\mathrm{Exp}^{\mathsf{Rob}}_{\mathsf{UKEM}}(\mathcal{A}) = 1\right] \leq \delta.$$

## D Security of NIZKPoK construction

**Additional Algorithms.** The additional algorithms of NIZKPoK are in Fig. 18.

### Completeness

**Lemma 6.** *If PKE is $\delta_1$-correct and NIZK is $\delta_2$-complete, then NIZKPoK in Fig. 10 is $(\delta_1 + \delta_2)$-complete.*

*Proof.* Consider the completeness game with the NIZKPoK from Fig. 10. The challenger first picks a seed $s$ and then runs the NIZK $crs' \leftarrow \Pi.\mathsf{Setup}$. The adversary $\mathcal{A}$ receives $(s, crs')$ and has access to random oracle $G$ which maps bit strings to PKE public keys. When $\mathcal{A}$ outputs a statement $x = (pk_i, pk_{i+1})$ and witness $w = \overline{sk}$, the challenger checks whether $(x, w) \in \mathcal{R}$ via Eq. (2). It samples randomness $r$ and computes a ciphertext $c$ that is an encryption of $w$ to $pk = G(s)$ using $r$. Then it runs the NIZK prover and verifier algorithm. If verification fails, $\mathcal{A}$ wins. In order to bound $\mathcal{A}$'s advantage, note that the

prover algorithm may fail if encryption fails. By the second property of PKE correctness, this happens with probability at most $\delta_1$. Then, by completeness of the NIZK, verification only fails with probability at most $\delta_2$. Hence, $\mathcal{A}$ wins with probability at most $\delta_1 + \delta_2$. □

**Zero-Knowledge**

*Proof (of Lemma 4).* The proof proceeds in two steps. We start with the ZK-PoK game with the bit $b = 0$. In the first step we modify the honest prover to produce a simulated proof $\pi'$. The zero knowledge property of $\Pi$ implies that the adversaries views in the original and modified experiments are identical.

In the second step we modify the prover to encrypt 0 instead of the witness. The CPA security of PKE implies that the adversaries views in the previous and modified experiments are indistinguishable.

Finally, we observe that after these two modifications, the view of the adversary is identical to that in the ZK-PoK game when $b = 1$ which concludes the proof. □

**Straightline Simulation-Extractability**

*Proof (of Lemma 5).* Let $\mathcal{A}$ be an adversary against extractability of PoK. If $\mathcal{A}$ wins (i.e. $result = 1$), then the following event occurs

**Event Win.** Occurs when $\mathcal{A}$ calls Test with $x$ and $\pi = (c, \pi')$ s.t. the following conditions hold:

1. Extract outputs an incorrect witness $w'$ ($(x, w') \notin \mathcal{R}$) and
2. no proof has been simulated for $x$ ($\pi[x] = \bot$) and
3. $\mathsf{V}(crs, x, \pi) = 1$.

Condition 1 can happen for two reasons: either Decrypt fails or the decrypted witness is incorrect. Observe that we cannot simply say that the former implies breaking correctness of PKE since $\mathcal{A}$ may have generated a malformed ciphertext $c$. Fortunately, verifying the NIZK proof $\pi$ in Condition 3 enforces that only well-formed ciphertexts $c$ are passed to Decrypt. We formalize this next using a game hop.

*Game 0.* This is the Ext experiment for PoK.

$$\mathsf{Adv}^{\mathsf{Ext}}_{\mathsf{PoK}}(\mathcal{A}) \coloneqq \Pr\left[\mathbf{Exp}_0(\mathcal{A})\right] . \tag{8}$$

*Game 1.* The experiment $\mathbf{Exp}_1$ differs from $\mathbf{Exp}_0$ in that the challenger stops and $\mathcal{A}$ loses if the following event occurs:

**Event BrkSS.** Occurs when Win occurs with $x, c$ and $\pi'$ such that there does *not* exist $r$ and $w$ with $c = \mathsf{Encrypt}(pk, w; r)$ and $(x, w) \in \mathcal{R}$.

*Claim.* There exists a reduction $\mathcal{B}$ such that

$$|[\mathbf{Exp}_1(\mathcal{A})] - [\mathbf{Exp}_0(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{SS}}_{\mathsf{NIZK}}(\mathcal{B}) .$$

The above claim is straightforward with the reduction $\mathcal{B}$ simulating the Ext game for $\mathcal{A}$ as follows: It receives $crs'$ from the NIZK challenger and picks a PKE key pair $(pk, sk)$. If $\mathcal{A}$ asks for a simulated proof, then $\mathcal{B}$ encrypts $0^\ell$ and queries its own simulate oracle to obtain a proof $\pi'$. If BrkSS occurs for $x, c$ and $\pi'$, then $\mathcal{B}$ breaks simulation soundness with statement $\tilde{x} = (x, pk, c)$ and proof $\pi'$. Indeed, Condition 3 of Win implies that NIZK.V accepts $\tilde{x}$ and $\pi'$. Further, Condition 2 of Win implies that no proof for $\tilde{x}$ has been simulated by $\mathcal{B}$'s Simulate oracle. Finally BrkSS implies that $\tilde{x}$ is not in the NIZK language.

*Claim.* In $\mathbf{Exp}_1$ we have $\Pr[\mathsf{Win}] \leq \delta$.

If $\mathcal{A}$ wins in $\mathbf{Exp}_1$ then $\mathsf{Win}$ occurs but $\mathsf{BrkSS}$ does not. So assume $\mathcal{A}$ wins with $x, c, \pi'$. Since $\mathsf{BrkSS}$ does not occur, there exists some randomness $r \in \mathcal{RS}$ such that $c = \mathsf{Encrypt}(pk, w; r)$ and $w$ is a correct witness for $x$. But by Condition 1, $\mathsf{Decrypt}$ outputs an incorrect witness $w'$ (which may also be $\perp$). By correctness of PKE, since $w \in \mathcal{M}$, $r \in \mathcal{RS}$, this can only happen if $sk \notin \overline{\mathcal{SK}}$, which in turn happens with probability at most $\delta$. □

## E Robustness of the LuKEM Scheme

**Lemma 7.** *Let* PKE *be key-homomorphic and* $(\ell, \delta_1)$*-correct and let* PoK *be* $\delta_2$*-complete. Then* LuKEM[PKE, PoK] *is* $(\ell, \delta')$*-robust for* $\delta' \leq q_{\text{enc}}(\delta_1 + \delta_2)$*, where* $q_{\text{enc}}$ *is the number of Enc queries* $\mathcal{A}$ *makes.*

**Proof.** Let $\mathcal{A}$ be an adversary in the robustness game. Note that when $\mathsf{Decaps}$ does not fail, the secret key at depth $j$ is of the form $sk_0 + \cdots + sk_j$, where all individual secret keys are in $\mathcal{SK}$, because the decryption algorithm of LuKEM generates them using $\mathsf{KeyGen}$ (with the randomness sampled using the random oracle) which (c.f., the syntax) outputs keys in $\mathcal{SK}$. Since $\mathcal{A}$ can make $\ell$ queries to Dec, this is consistent with the correctness definition of key-homomorphic PKE. Using a union bound over the number of encryption queries, we can upper bound robustness by the correctness of PKE and completeness of PoK. □

## F Proof of Theorem 1

We prove Theorem 1 via Theorems 3 to 5. To this end, we define the two intermediate security notions IND-CPA$^+$ and IND-CCA$^-$ (Appendix F.1). First, we show that if PKE is Shifty-IND-CPA secure, then LuKEM$_{\text{no-pv}}$ is IND-CPA$^+$ secure (Appendix F.2). Second, we show that if LuKEM$_{\text{no-pv}}$ is IND-CPA$^+$ secure and PKE is correct and $\gamma$-spread, then LuKEM$_{\text{no-pv}}$ is also IND-CCA$^-$ secure (Appendix F.3). Finally, we show that if LuKEM$_{\text{no-pv}}$ is IND-CCA$^-$ secure and PoK is straightline simulation extractable (along with the other PoK properties), then LuKEM is IND-CCA secure (Appendix F.4).

### F.1 Intermediate Security Notions

Consider an IND-CPA notion which differs from IND-CCA in that there is no Dec oracle. Our first intermediate notion, IND-CPA$^+$, strengthens IND-CPA by adding an Upd oracle which allows the adversary to create new nodes by providing the difference $\overline{sk}$ between the new and the old secret key (for this to make sense, we assume that the UKEM scheme is key-homomorphic). This means that all nodes in the IND-CPA$^+$ experiment are full, unless a correctness error occurs.

Our second intermediate notion, IND-CCA$^-$, strengthens IND-CPA$^+$ by adding a Dec$^-$ oracle that differs from Dec in that it does not takes as input a member-tag and it only creates a new node if decryption succeeds. In that case it outputs $K$ and $\perp$ otherwise. The Upd and Dec$^-$ oracle are shown in Fig. 19.

*Remark 2.* We note that IND-CPA$^+$ is technically similar to "chosen-randomness IND-CPA" notions from the literature, e.g., [ACDT20]. In the latter, the adversary has an Upd oracle where it provides *randomness* used in Encaps. This is motivated by modeling attacks on random-number generators used by honest parties.

**The Upd and Dec$^-$ Oracles for Intermediate Security Notions**

**Oracle Upd**$(i', \overline{sk})$

**req** $pk_{i'} \neq \bot$ // $i$-th node exists
**req** $\overline{sk} \in \mathcal{SK}$
$j$++
$pk' \leftarrow pk_{i'} + \mathsf{pk}(\overline{sk})$
$sk' \leftarrow sk_{i'} + \overline{sk}$
$(pk_j, sk_j, par_j, rev_j, typ_j) \leftarrow (pk', sk', i', 0, \text{“Dec”})$
**return** $\bot$

**Oracle Dec$^-$**$(i', c', pk')$

**req** $sk_{i'} \neq \bot$ // $i$-th node exists and full
**if** $i^* \neq \bot$ **then** // implies $c^* \neq \bot$
  **req** $c^* \neq c' \vee pk^* \neq pk_{i'}$
$(K, sk') \leftarrow \mathsf{Decaps}(pp, sk_{i'}, c', pk')$
**if** $K \neq \bot$ **then**
  $j$++
  $(pk_j, par_j, rev_j, typ_j, sk_j) \leftarrow (pk', i', 0, \text{“Dec”}, sk')$
  **return** $K$
**return** $\bot$

Fig. 19: Oracles for the intermediate notions: IND-CPA$^+$ is the same as IND-CCA in Fig. 3 except that in IND-CPA$^+$ the Dec oracle is replaced by Upd. IND-CCA$^-$ is the same as IND-CPA$^+$ and additionally provides the Dec$^-$ oracle.

Our IND-CPA$^+$ is (strictly) stronger, since the adversary can create nodes for arbitrary $\overline{sk}$, while chosen-randomness notions restrict it to creating nodes for which it knows Encaps randomness $r$. For example in our scheme, $\overline{sk}$ is generated by hashing $r$ and running key generation on the hash output.

We do not see any real-world motivation for this strengthening. However, it is needed for our reduction from IND-CPA$^+$ to IND-CCA. Indeed, the reduction will extract $\overline{sk}$ from the PoK proofs sent by the adversary to the Dec oracle. However, it will not know the randomness $r$, i.e. (part of) the hash *input* resulting in $\overline{sk}$. We note that alternative constructions that do allow to extract $r$ are significantly less efficient. For instance, in previous constructions [DKW21, APS23] Encaps encrypts a random and independent $\overline{sk}$ using a PKE.

**Definition 19 (UKEM IND-CPA$^+$ and IND-CCA$^-$ Security).** *Let* $\mathsf{ATK} \in \{\mathsf{CPA}^+, \mathsf{CCA}^-\}$. *Let* $\mathrm{Exp}^{\mathsf{IND\text{-}ATK}}(\mathcal{A})$ *be as defined in Fig. 19. The advantage of an adversary* $\mathcal{A}$ *against the* IND-ATK *security of a key-homomorphic* UKEM *scheme is defined as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}ATK}}_{\mathsf{UKEM}}(\mathcal{A}) := 2\Pr\left[\mathrm{Exp}^{\mathsf{IND\text{-}ATK}}_{\mathsf{UKEM}}(\mathcal{A}) = 1\right] - 1 \ .$$

### F.2 From Shifty-IND-CPA to IND-CPA$^+$

In the first step, we show that the scheme $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ is IND-CPA$^+$ secure.

**Theorem 3.** *Let* PKE *be a key-homomorphic PKE with message space* $\{0,1\}^\kappa$. *If* PKE *is* $(\ell, \delta)$*-correct and* Shifty-IND-CPA *secure, then the construction* $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ *in Fig. 5 is* IND-CPA$^+$ *secure in the ROM. Specifically, for any adversary* $\mathcal{A}$ *against* IND-CPA$^+$ *security of* $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ *that issues* $q_{\mathrm{ro}}$ *RO queries,* $q_{\mathrm{enc}}$ *Enc queries,* $q_{\mathrm{upd}}$ *Upd queries and creates a tree of depth at most* $\ell$, *there exists an adversary* $\mathcal{B}$ *against* Shifty-IND-CPA *security of* PKE *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}^+}_{\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}}(\mathcal{A}) \leq (q_{\mathrm{enc}}+1)\cdot\delta + (q_{\mathrm{enc}}+q_{\mathrm{upd}})\cdot \mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{B}) + q_{\mathrm{ro}}\cdot(q_{\mathrm{enc}}+1)\cdot 2^{-\kappa} \ ,$$

*where* $\mathcal{B}$ *issues at most* $q_{\mathrm{enc}}+1$ *Chal queries.*

For an intuition we refer to Section 5.1. Here we give the full proof.

**Proof of Theorem 3.** We now prove the theorem formally. In the following, we denote by $\Pr[\mathbf{Exp}_i(\mathcal{A})]$ the probability of game $i$ returning 1.

*Game 0.* This is the $\mathsf{IND\text{-}CPA}^+$ experiment for $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ and adversary $\mathcal{A}$. We have

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}^+}_{\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}}(\mathcal{A}) = 2\Pr\left[\mathbf{Exp}_0(\mathcal{A})\right] - 1 \ .$$

*Game 1.* In this game, the challenger aborts when a correctness error occurs during the creation on an honest node. This way, it is ensured that all nodes will indeed be full nodes. We can bound the difference between Game 0 and Game 1 by robustness of $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ and hence by the $(k, \delta)$-correctness of $\mathsf{PKE}$. We have

$$\left|\Pr\left[\mathbf{Exp}_0(\mathcal{A})\right] - \Pr\left[\mathbf{Exp}_1(\mathcal{A})\right]\right| \leq (q_{\mathrm{enc}} + 1) \cdot \delta \ .$$

*The "break" event.* Let $\mathsf{Brk}$ be the event from the above intuition. More concretely let $i_0, \ldots, i_c$ be the challenge path from the root 0 to the node $i_c$ inputted by $\mathcal{A}$ to MChal. For an index $i^*$, the event $\mathsf{Brk}(i^*)$ occurs if all of the following hold:

(1) $i^*$ is on the challenge path,
(2) $i^*$ was created by an Enc query (and not an Upd query),
(3) there exists a $j^*$ s.t. $j^* = i^*$ or $j^*$ is reachable from $i^*$ via only Dec edges (these are created via Upd queries), and for some child $j$ of $j^*$, $\mathcal{A}$ queries any RO on the message $M_j$ that the challenger encrypted to obtain $c_j$ returned by $\mathrm{Enc}(j^*)$ or $\mathrm{MChal}(j^*)$.

We will show that

$$2\Pr\left[\mathbf{Exp}_1(\mathcal{A})\right] - 1 \leq \Pr[\exists i^* : \mathsf{Brk}(i^*)]. \tag{9}$$

We then define the following event $\mathsf{Brk}^*$: an index $i^*$ is chosen (by the reduction) at random independently of $\mathcal{A}$'s view. $\mathsf{Brk}^*$ occurs if $\mathsf{Brk}(i^*)$ occurs and among the nodes $i$ on the challenge path for which $\mathsf{Brk}(i)$ occurs, $i^*$ is the one closest to the root.

$$\Pr[\mathsf{Brk}] \leq (q_{\mathrm{enc}} + q_{\mathrm{upd}}) \cdot \Pr[\mathsf{Brk}^*]. \tag{10}$$

Finally, we construct a reduction $\mathcal{B}$ such that

$$\Pr[\mathsf{Brk}^*] \leq \mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{B}) + q_{\mathrm{ro}} \cdot (q_{\mathrm{enc}} + 1) \cdot 2^{-\kappa}. \tag{11}$$

Combining Eqs. (9) to (11), we get the bound from the theorem.

*Proof of Eq. (9).* The reason Eq. (9) holds is that if $\mathcal{A}$ does not trigger $\mathsf{Brk}$ then in particular $\mathcal{A}$ does not query the RO $H_1$ on $M^*$ encrypted by the challenger to obtain $\mathcal{A}$'s challenge ciphertext $c^*$. Recall that the real key is $K^{(1)} = H_1(pp_{\mathsf{PKE}}, pk^*, M^*)$. So until $\mathcal{A}$ sends $M^*$ to the RO the experiment is independent of the challenge bit $b$.

*Reduction (proof of Eq. (11)).* We now construct a reduction $\mathcal{B}$ against $\mathsf{Shifty\text{-}IND\text{-}CPA}$ that guesses $i^*$ and simulates the experiment for $\mathcal{A}$ until event $\mathsf{Brk}^*$ occurs or until the guess becomes incorrect, in which case it aborts.

$\mathcal{B}$ gets as input the public key $pk^*$, which it embeds as follows. If $i^* = 0$ then $\mathcal{B}$ embeds $pk^*$ as $pk_0$ and proceeds as described in the next paragraph. If $i^* > 0$ then $\mathcal{B}$ samples $(pk_0, sk_0)$ itself. It answers all Enc queries from $\mathcal{A}$ honestly by running $\mathsf{Encaps}$ until $i^*$ is created. Assume the node $i^*$ is created via an $\mathrm{Enc}(p^*)$ query; else condition (2) of $\mathsf{Brk}(i^*)$ is false and the experiment stops. Thus, $\mathcal{B}$ can embed $pk^*$ as the update from $pk_{p^*}$ to

$pk_{i^*}$, i.e., it sets $pk_{i^*} := pk_{p^*} + sk^*$. For the ciphertext returned by $\mathrm{Enc}(p^*)$, $\mathcal{B}$ encrypts a random message $m$. It also samples the key $K$ returned by $\mathrm{Enc}(p^*)$ at random. Observe that this simulation is identical to the $\mathsf{IND\text{-}CPA}^+$ experiment until $\mathcal{A}$ queries $M$ to an RO (which for $H_3$ would output $sk^*$). However, if $\mathcal{A}$ makes such a query, then $\mathsf{Brk}(p^*)$ occurs — this means that $\mathsf{Brk}$ occurs for a node closer to the root than $i^*$, so $\mathsf{Brk}^*$ can no longer happen and the experiment stops.

Observe that $\mathcal{B}$ knows the secret keys of all nodes outside the subtree rooted at $i^*$ and thus can answer all queries perfectly for these nodes.

We now describe how $\mathcal{B}$ deals with queries related to descendants $j^*$ of $i^*$. Recall that the $\mathsf{IND\text{-}CPA}^+$ experiment (c.f. Fig. 3) allows $\mathcal{A}$ to corrupt nodes outside the set $\mathsf{dec\text{-}closure}(S)$ where $S$ contains the challenge path (and any nodes with the same public keys) and $\mathsf{dec\text{-}closure}$ extends $S$ by all nodes reachable from it via only Dec edges. $\mathcal{B}$ will (in principle) support more corruptions — it will know the secret keys of all nodes outside $\mathsf{dec\text{-}closure}(\{i^*\})$ which is a subset of $\mathsf{dec\text{-}closure}(S)$ since $i^*$ is on the challenge path (else the guess is incorrect and the experiment stops).

This means that $\mathcal{B}$ can answer queries for nodes outside $\mathsf{dec\text{-}closure}(\{i^*\})$ perfectly. It remains to show how $\mathcal{B}$ deals with queries for nodes $j^* \in \mathsf{dec\text{-}closure}(\{i^*\})$. $\mathrm{Rev}(j^*)$ is not allowed and $\mathrm{Upd}(j^*)$ is answered honestly (recording $\overline{sk}$ provided by $\mathcal{A}$). The remaining queries are $\mathrm{Enc}(j^*)$ and $\mathrm{MChal}(j^*)$. For such a query creating a node $j' \notin \mathsf{dec\text{-}closure}(\{i^*\})$, $\mathcal{B}$ queries its Chal oracle.

If $j^* = i^*$ then $\mathcal{B}$ queries Chal on $sk_{p^*}$ and two random messages $m_{j'}^{(0)}$ and $m_{j'}^{(1)}$. Recall that $\mathcal{B}$ knows all the individual secret keys that sum up to $sk_{p^*}$ which are those expected by the $\mathsf{Shifty\text{-}IND\text{-}CPA}$ challenger. It receives $(c_{j'}^*, pk_{j'}, sk_{j'})$, where $pk_{j'} = \mathsf{pk}(sk_{p^*}) + pk^* + \overline{pk}$ and $sk_{j'} = sk_{p^*} + sk^* + \overline{sk}$ for a fresh key pair $(\overline{pk}, \overline{sk})$. Further, $c_{j'}^*$ is an encryption of $m_j^{(b)}$, where $b$ is the challenge bit in the $\mathsf{Shifty\text{-}IND\text{-}CPA}$ game. $\mathcal{B}$ samples $K_{j'}^*$ uniformly at random and outputs $K_{j'}^*, c_{j'}^*, pk_{j'}$ to $\mathcal{A}$. Hence, it will know $sk_{j'}$ to answer future Rev queries of $\mathcal{A}$.

$\mathcal{B}$ proceeds similarly when there are Upd edges in between $i^*$ and $j^*$. Since $\mathcal{A}$ provides $\overline{sk}$ whenever it queries the Upd oracle, $\mathcal{B}$ can query its Chal oracle on $sk_{p^*} + sk_{i^* \to j^*}$ and random messages $M_j^{(0)}, m_j^{(1)}$. It will receive $(c_j^*, pk_j, sk_j)$ for the descendant $j$ of $j^*$. It will again sample a random key $K_j^*$ and return $(K_j^*, c_j^*, pk_j)$ to $\mathcal{A}$.

Using the above strategy $\mathcal{B}$ can simulate all nodes, no matter which one will end up being on the challenge path. It remains to explain that when $\mathsf{Brk}^*$ indeed happens, $\mathcal{B}$ can win the $\mathsf{Shifty\text{-}IND\text{-}CPA}$ game. It observes $\mathcal{A}$'s RO queries and as soon as $\mathcal{A}$ makes a query on an input containing $m_j^{(b')}$ for some $b'$ and $j$, then $\mathcal{B}$ outputs $b'$. If $\mathcal{B}$'s bit is $b$, then the probability that an RO query from $\mathcal{A}$ contains $m_j^{(1-b)}$ (which is random and independent of $\mathcal{A}$'s view) for some $j$ is at most $(q_{\mathrm{enc}} + 1) \cdot 2^{-\kappa}$ (since $\mathcal{B}$ embeds at most $(q_{\mathrm{enc}} + 1)$ challenges). So if $\mathcal{A}$ triggers $\mathsf{Brk}$ then $\mathcal{B}$ wins except with probability $q_{\mathrm{ro}} \cdot (q_{\mathrm{enc}} + 1) \cdot 2^{-\kappa}$, which proves Eq. (11). □

### F.3 From $\mathsf{IND\text{-}CPA}^+$ to $\mathsf{IND\text{-}CCA}^-$ by observing the RO

We show that the scheme already satisfies the stronger notion of $\mathsf{IND\text{-}CCA}^-$. Compared to the previous proof, we have to show that the reduction can correctly simulate the (restricted) decapsulation oracle $\mathrm{Dec}^-$ from Fig. 19. We will later see how this notion enables us to prove full $\mathsf{IND\text{-}CCA}$ security.

**Theorem 4.** *If* $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ *is* $\mathsf{IND\text{-}CPA}^+$ *secure and* $\mathsf{PKE}$ *is* $\delta$*-correct and* $\gamma$*-spread, then* $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ *is* $\mathsf{IND\text{-}CCA}^-$ *secure in the* non-programmable *ROM. More specifically, for*

*any adversary $\mathcal{A}$ against* IND-CCA$^-$ *security of* LuKEM$_{\text{no-pv}}$ *that issues $q_{\text{ro}}$ random oracle queries, $q_{\text{enc}}$ encryption queries, $q_{\text{dec}}$ decryption queries and creates a tree of depth at most $\ell$, there exists an adversary $\mathcal{B}$ against* IND-CPA$^+$ *security of* LuKEM$_{\text{no-pv}}$ *such that*

$$\mathsf{Adv}^{\text{IND-CCA}^-}_{\text{LuKEM}_{\text{no-pv}}}(\mathcal{A}) \leq 2q_{\text{ro}} \cdot \delta + q_{\text{dec}} \cdot 2^{-\gamma} + \mathsf{Adv}^{\text{IND-CPA}^+}_{\text{LuKEM}_{\text{no-pv}}}(\mathcal{B}) \ ,$$

*where $\mathcal{B}$ issues at most $3q_{\text{ro}}$ random oracle queries, $q_{\text{enc}}$ encryption queries and as many Upd queries as $\mathcal{A}$.*

**Proof Intuition.** We extend the strategy from Theorem 3 to deal with $\mathcal{A}$'s Dec$^-$ queries. For this, we will use the fact that the scheme is based on the Fujisaki-Okamoto (FO) transform.

We want to construct a reduction $\mathcal{B}$ that plays in the IND-CPA$^+$ game, meaning it has only access to an Upd oracle. Fortunately, whenever we decryption succeeds, we will also know the secret key shift that the adversary used to create the new public key $pk'$. Thus, using the techniques from security proofs of the FO transform, we construct $\mathcal{B}$ that extracts not only $K$ but also $\overline{sk}$ from $\mathcal{A}$'s RO queries as follows.

*Observing random oracles.* We now explain how $\mathcal{B}$ answers Dec$^-$ queries in more detail. Since $\mathcal{B}$ is playing in the IND-CPA$^+$ game for LuKEM$_{\text{no-pv}}$, $\mathcal{B}$'s challenger also provides access to the random oracles. $\mathcal{B}$ will simulate $\mathcal{A}$'s random oracle queries by forwarding them to its challenger.

To simulate the Dec$^-$ oracle, $\mathcal{B}$ uses a list $\mathcal{L}_D$ that tracks $\mathcal{A}$'s random oracle queries. Essentially, $\mathcal{B}$ will only need to answer Dec$^-$ queries when $\mathcal{A}$ has previously queried $H_1$. To simplify the analysis, we will simulate all random oracles together, meaning that as soon as $\mathcal{A}$ queries one of the RO's on a new value $(pk, m)$, $\mathcal{B}$ sends $(pk, m)$ to all RO's $H_1$, $H_2$ and $H_3$ (emulated by its challenger), which output, respectively, $r$, $K$ and $d$. Then $\mathcal{B}$ stores in $\mathcal{L}_D$ a record $(pk, pk', \overline{sk}, m, c, K)$ where $c = \mathsf{Encrypt}(pp, pk, m; r)$, $(\overline{pk}, \overline{sk}) = \mathsf{KeyGen}(d)$ and $pk' = pk + \overline{pk}$. This will allow us to answer Dec$^-$ queries without searching through all random oracle queries. More specifically, if $\mathcal{A}$ queries Dec$^-$ on input $(i, c', pk')$, $\mathcal{B}$ simply checks whether there exists an entry $(pk_i, pk', \overline{sk}, m, c', K) \in \mathcal{L}_D$ for some $\overline{sk}, m, K$. If so, the "re-encryption check" has succeeded. $\mathcal{B}$ creates a new node by querying the Upd oracle on $(i, \overline{sk})$ and outputs $K$. Else, it outputs $\perp$.

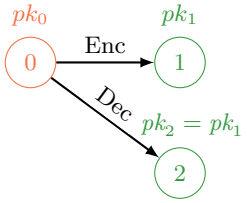
We now give the full proof, where we also deal with correctness errors and the probability that a decryption query succeeds, where the adversary did not query the random oracle.

### F.4 From IND-CCA$^-$ to IND-CCA using PoK

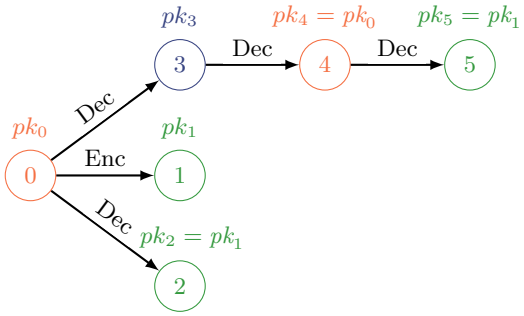
In the final step, we consider LuKEM shown in Fig. 6. It differs from LuKEM$_{\text{no-pv}}$ in that it uses a member-tag $mt_{i+1}$ which is a PoK proving knowledge of $\overline{sk}$ such that $pk_i + \mathsf{pk}(\overline{sk}) = pk_{i+1}$ and $\overline{sk} \in \mathcal{SK}$. We show that LuKEM is IND-CCA secure if the underlying key-homomorphic UKEM is IND-CCA$^-$ secure and PoK is straightline simulation-extractable. For our final scheme, we instantiate UKEM with LuKEM$_{\text{no-pv}}$.

**Theorem 5.** *If* UKEM *is key-homomorphic and* IND-CCA$^-$ *secure and* PoK *is straightline simulation extractable, then* LuKEM *is* IND-CCA *secure. More specifically, for any adversary $\mathcal{A}$ against* IND-CCA *security of* LuKEM*, there exist adversaries $\mathcal{B}$, $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ against the* IND-CCA$^-$ *security of* UKEM*, the setup indistinguishability game of pk, the zero-knowledge game of* PoK *and the extractability security game of* PoK *such that*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\text{LuKEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{IND-CCA}^-}_{\text{UKEM}}(\mathcal{B}) + 2 \cdot (\mathsf{Adv}^{\text{SI}}_{\text{PoK}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{ZK}}_{\text{PoK}}(\mathcal{B}_2) + \mathsf{Adv}^{\text{Ext}}_{\text{PoK}}(\mathcal{B}_3)) \ .$$




(a) $\mathcal{A}$ queries Enc(0), which creates node 1, and then queries $\mathrm{Dec}(1, \bot, pk_1, mt_1)$ which creates a half-node 2 with the same key. The reduction $\mathcal{B}$ only creates a node on the Enc query.




(b) A more subtle scenario. $\mathcal{A}$ again queries Dec and creates node 3 with $pk_3 := pk_0 + \mathsf{pk}(\overline{sk})$ using Encaps in its head. Then it creates a half-node 4 by "subtracting" the difference $\mathsf{pk}(\overline{sk})$ i.e. it queries $\mathrm{Dec}(3, \bot, pk_0, mt)$. Note, $\mathcal{A}$ can compute $mt$ which proves knowledge of the secret key corresponding to $pk_0 - pk_3 = -\mathsf{pk}(\overline{sk}) = -\overline{sk}$ (latter equality holds for any homomorphism pk). Then it forwards the outputs of the first Enc query to create a child of 4.

Fig. 20: Illustration of how $\mathcal{A}$ can copy honestly generated keys to Dec nodes. Each colour marks nodes with the same public key, corresponding to one node in the IND-CCA$^-$ game.

**Proof Intuition.** Let $\mathcal{A}$ be an adversary against the IND-CCA security of LuKEM. We construct a reduction $\mathcal{B}$ against IND-CCA$^-$ security of UKEM. In general, there will be a one-to-one correspondence between nodes in the IND-CCA experiment and nodes in the IND-CCA$^-$ experiment. Note that all nodes in the IND-CCA$^-$ game are full (up to robustness), so half-nodes in the IND-CCA experiment will correspond to full nodes in the IND-CCA$^-$ game. The reduction $\mathcal{B}$ keeps track itself of which nodes are half-nodes in the IND-CCA experiment it emulates. In the following, we look at how nodes are created in both experiments.

When $\mathcal{A}$ instructs its emulated challenger to create an Enc-node $j$ by calling Enc($i$) or MChal($i$), $\mathcal{B}$ also instructs the IND-CCA$^-$ challenger to create an Enc-node $j$ by calling Enc($i$) or MChal($i$). Recall, these oracles output $K$, $c$, $pk_j$ and $mt$. The schemes UKEM and LuKEM differ only on the last output $mt$; in UKEM $mt$ is empty, and in LuKEM $mt$ is a PoK $\pi$ proving knowledge of $\overline{sk}$ such that $pk_i + \mathsf{pk}(\overline{sk}) = pk_j$. The reduction $\mathcal{B}$ does not know $\overline{sk}$ chosen by its challenger. Thus, it simulates $\pi$ using PoK.S. Finally, $\mathcal{B}$ marks the new node $j$ as full if node $i$ was full.

When $\mathcal{A}$ instructs its emulated challenger to create a Dec-node $j$ by calling $\mathrm{Dec}(i', c', pk', mt')$, $\mathcal{B}$ first checks whether $mt'$ is a valid member-tag. Then it queries its restricted Dec$^-$ oracle to check whether $c'$ successfully decrypts in which case it gets $K$ as a response and knows that its challenger has created a Dec-node. If $c'$ is not a valid ciphertext, $\mathcal{B}$ must still create a Dec-node for $\mathcal{A}$. It simulates those nodes using its own Upd oracle. Recall that Upd takes as input $i'$ and $\overline{sk}$ such that $pk_{i'} + \mathsf{pk}(\overline{sk}) = pk'$. $\mathcal{B}$ uses the (straightline) extractor PoK.Extract to extract $\overline{sk}$ from $mt'$ provided by $\mathcal{A}$.[21]

Whenever $\mathcal{A}$ calls Rev($i$), $\mathcal{B}$ does as follows: If $i$ is a half-node, it outputs $\bot$. Else, it calls its Rev oracle to get $sk_i$.

[21] This means that in the simulated experiment we have multiple simulated proofs interleaved with multiple extractions. This is why we require PoK to be straightline simulation-extractable.

*Extracting from adversarial proofs.* So far, we assumed that whenever $\mathcal{A}$ calls Dec and decapsulation fails, $\mathcal{B}$ can extract $\overline{sk}$ from the PoK $mt'$ sent by $\mathcal{A}$ to the Dec oracle. However, extraction does not work for statements $(pk_p, pk_i)$ for which $\mathcal{B}$ previously simulated PoK's using PoK.S.

For example, say $\mathcal{A}$ queries Enc($p$), which creates node $i$ with $pk_i$ and outputs $K$, $c$, $pk_i$ and the simulated proof $mt$. Then $\mathcal{A}$ replays $p$, $pk_i$ and $mt$ together with some $c'$ to the Dec oracle. See the illustration in Fig. 20. If decapsulation fails (maybe $c'$ is random), $\mathcal{B}$ has to create a half-node $i'$. However, it cannot use the strategy from the previous paragraph since it cannot extract from a simulated proof $mt$ (nor from any proof $mt'$ for the corresponding statement $(pk_p, pk_i)$). However, extraction is not needed, since nodes $i'$ and $i$ are equivalent! That is, $i'$ and $i$ have the same public keys and the half-node $i'$ has no secret key, thus, $\mathcal{B}$ can use its oracles for $i$ to deal with $i'$.

More precisely, we adjust $\mathcal{B}$'s strategy as follows: Each Enc-node $i$ in the IND-CCA$^-$ experiment corresponds to multiple nodes in the IND-CCA experiment: one Enc-node with public key $pk_i$ and a set copied($i$) of half-nodes with the same public key $pk_i$ that $\mathcal{A}$ created by copying as described above. (This means that indices no longer match between the experiments, complicating bookkeeping a bit.) If $\mathcal{A}$ queries Dec with the parent of $i$ (in the IND-CCA experiment) and $pk_i$, and Dec$^-$ returns $\perp$ (and $\mathcal{A}$'s member tag is valid), then $\mathcal{B}$ simply creates a new half-node with $pk_i$ and adds it to copied($i$). If $\mathcal{A}$ queries Enc (or MChal) with a node in copied($i$), $\mathcal{B}$ queries Enc (or MChal) with $i$. Thus, a new node is created in both experiments.

Now say $\mathcal{A}$ queries Dec with a node in copied($i$) and some $pk_j$ and $mt$. Fortunately, this only creates a half-node in IND-CCA. If $\mathcal{B}$ has not simulated a PoK for $(pk_i, pk_j)$, which would be the case if $pk_j$ was outputted by Enc (or MChal) with a node in copied($i$), then $\mathcal{B}$ can extract $\overline{sk}_j$ from $mt$ and forward it to the Upd oracle as before to create a new (full) node in IND-CCA$^-$. Otherwise, $\mathcal{B}$ creates a new half-node in copied($j$).

*Trivial wins with copied nodes.* Assume $\mathcal{A}$ queries MChal($i_c$) for a node $i_c$ such that $\mathcal{B}$, mapping $i_c$ to the respective node $i^*$ in its own game, queries MChal($i^*$). Recall that $\mathcal{A}$ is not allowed to corrupt nodes in the *extd-base* set which includes all nodes with the same public key as $pk_{i_c}$. Since all nodes in copied($i^*$) are half-nodes, $\mathcal{A}$ cannot violate trivial-win restrictions by corrupting them. All other nodes that $\mathcal{A}$ creates also exist in $\mathcal{B}$'s game. For example, note that $\mathcal{A}$ may create a full-node copy of an Enc-node by replaying not only $pk_j$ and $mt$ but also the corresponding ciphertext outputted by $\mathcal{B}$'s challenger. Although $\mathcal{B}$ cannot extract from $mt$, it can query Dec$^-$ to create a new node. If on the challenge path, such a node will be in the *extd-base* set in both games.

This reasoning extends to other nodes on the challenge path 0-$i^*$. This way, we can argue that if $\mathcal{A}$'s corruptions do not violate trivial-win restrictions in IND-CCA, then $\mathcal{B}$'s corruptions do not violate the trivial-wins either.

A similar argument applies to the trivial-win restriction in the Dec oracle. If $\mathcal{B}$ violates the restriction by sending $c^*$ or $pk_{i^*}$ to Dec$^-$, then $\mathcal{A}$ is violating its restriction too. Indeed, if $\mathcal{B}$ called MChal($i^*$) which returned $c^*$, then $\mathcal{A}$ called MChal($i_c$) for some $i_c \in \text{copied}(i^*) \cup \{i^*\}$ which also returned $c^*$. So, $\mathcal{A}$ cannot call Dec on $c^*$ or $pk_{i^*} = pk_{i_c}$.

**Proof of Theorem 5.** We define a sequence of games to prove the theorem, where we will follow the strategy explained in the intuition and denote by $\Pr[\mathbf{Exp}_i(\mathcal{A})]$ the probability of game $i$ returning 1.

*Game 0.* This is the IND-CCA experiment for LuKEM[UKEM, PoK], where UKEM is a key-homomorphic UKEM and the PoK statement is defined in Eq. (2). Hence, for an

adversary $\mathcal{A}$ in that game we have

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{LuKEM}}(\mathcal{A}) \coloneqq 2\Pr\left[\mathbf{Exp}_0(\mathcal{A})\right] - 1 \ . \tag{12}$$

*Game 1 (Switch setup).* Game 1 differs from Game 0 in that the challenger replaces the call to $\mathsf{PoK.Setup}$ during initialization to $\mathsf{PoK.SetupAlt}^H$. The challenger also emulates the random oracle $H$ for $\mathsf{PoK.SetupAlt}$ and $\mathcal{A}$ by running its code. For the straightforward reduction $\mathcal{B}_1$ we have

$$|[\mathbf{Exp}_0(\mathcal{A})] - [\mathbf{Exp}_1(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{SI}}_{\mathsf{PoK}}(\mathcal{B}_1) \ . \tag{13}$$

*Game 2 (Simulate proofs).* In Game 2, the challenger replaces all proofs $mt$ outputted by the Enc and MChal oracles by simulated ones. Indistinguishability is implied by the zero-knowledge property of $\mathsf{PoK}$.

More precisely, for each Enc and the MChal query, it first computes $(K, c, pk_j, \overline{sk})$ as in the underlying UKEM. However, instead of computing $mt_j \leftarrow \mathsf{PoK.Prove}(crs, (pk_i, pk_j), \overline{sk})$, it uses the simulator algorithm to compute $mt_j \leftarrow \mathsf{PoK.S}^H(td, (pk_i, pk_j))$, where $td$ is the trapdoor output by $\mathsf{PoK.SetupAlt}$. It outputs $(K, c, pk_j, mt_j)$ to $\mathcal{A}$. As in Game 1, it emulates the random oracle $H$ for $\mathsf{PoK.SetupAlt}$ and $\mathsf{PoK.S}$ by running its code. For the straightforward reduction $\mathcal{B}_2$ we have

$$|[\mathbf{Exp}_1(\mathcal{A})] - [\mathbf{Exp}_2(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{PoK}}(\mathcal{B}_2) \ . \tag{14}$$

*Game 3 (Extract from $\mathcal{A}$'s proofs).* In Game 3, the challenger additionally extracts witnesses from member-tags provided by $\mathcal{A}$ using $\mathsf{PoK.Extract}^H$. Indistinguishability is implied by straight-line simulation-extractability of $\mathsf{PoK}$.

We now also introduce the helper set $\mathsf{copied}(i)$ that stores indices of half-nodes that are copies of node $i$ and hence have the same public key $pk_i$. This way, the challenger can identify Dec queries for which it will need to extract (and store) the secret key $\overline{sk}$ and those which are simply copies of Enc-nodes. More specifically, when $\mathcal{A}$ makes a Dec query $(i', c', pk', mt')$ such that $mt'$ is valid (otherwise, the challenger outputs $\perp$ directly), we distinguish 2 cases:

1. *Replay of public key from Enc query.* If $pk'$ was previously output of an Enc query, then the challenger has previously simulated a proof for statement $(pk_{i'}, pk')$. However, in order to define the set $\mathsf{copied}(\cdot)$ such that the final reduction to $\mathsf{IND\text{-}CCA}^-$ works, we need to make a more subtle case distinction. The reason is that $i'$ might have been created via Dec itself. Hence, we first let $i$ be the index of the node with public key $pk'$ that was previously created via Enc, i.e., $pk_{par(i)} = pk_{i'}$, but crucially $par(i)$ may not be the same as $i'$.[22] Node $i$ will be the one that determines the set $\mathsf{copied}(i)$. More specifically, we distinguish the following cases:

   (a) If $i'$ is a half-node, the challenger records $j$ as a half-node, adds $j$ to $\mathsf{copied}(i)$ and outputs $\perp$.
   (b) If $i'$ is a full-node, the challenger decrypts $c'$. If successful, it records $j$ as a full-node and outputs the result to $\mathcal{A}$. (It does not use $\mathsf{copied}(i)$ here.) In case decryption fails, this node is a half-node and is added to $\mathsf{copied}(i)$.

[22] To see this, consider Fig. 20b. The given scenario is illustrated for $i = 1$, $par(i) = 0$, $i' = 4$, and $j = 5$ (assuming at least one of these nodes is a half-node). Note also that the set $\mathsf{copied}(1)$ does not only contain 5, but also 2 in case it is a half-node.

2. *All other queries.* In this case, the challenger has not previously output a simulated proof for statement $(pk_{i'}, pk')$. The challenger hence runs the extractor $\mathsf{PoK.Extract}(td, (pk_{i'}, pk'), mt')$ to obtain $\mathcal{A}$'s witness $\overline{sk}$. It additionally checks whether $pk_{i'} + \mathsf{pk}(\overline{sk}) = pk'$. If this is not the case, the challenger aborts the game. Note that this means that $\mathcal{A}$ can be turned into an adversary $\mathcal{B}'$ that wins the extractability game. If $i'$ is a full node, the challenger additionally decrypts $c'$ and creates a full node (if successful) or a half-node (if decryption fails). If $i'$ is a half-node, the challenger creates a half-node.

Based on the above, we can construct an adversary $\mathcal{B}_3$ in the extractability game (cf. Fig. 16). To create simulated proofs, $\mathcal{B}_3$ calls the Simulate oracle. To extract from member-tags, $\mathcal{B}_3$ calls the Test oracle. Random oracle queries from $\mathcal{A}$ can simply be forwarded. If at any point in time, $pk_{i'} + \mathsf{pk}(\overline{sk}) \neq pk'$ happens in Case 2 above, then $\mathcal{B}_3$ has queried the Test oracle on $(x = (pk_{i'}, pk'), mt')$ such that $\pi[pk_{i'}, pk'] = \bot$ before the query and hence $\mathcal{B}_3$ wins. We get

$$|[\mathbf{Exp}_2(\mathcal{A})] - [\mathbf{Exp}_3(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{Ext}}_{\mathsf{PoK}}(\mathcal{B}_3) \ .$$

*Final reduction.* We can now construct an adversary $\mathcal{B}$ such that

$$2\Pr\left[\mathbf{Exp}_3(\mathcal{A})\right] - 1 \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}^-}_{\mathsf{UKEM}}(\mathcal{B}) \ .$$

Adversary $\mathcal{B}$ simulates Game 2 as follows. It gets as input $pk_0$ and has access to oracles Enc, Rev, MChal, Upd as well as the $\mathrm{Dec}^-$ oracle that only creates nodes if decryption does not fail. When $\mathcal{A}$ queries its own oracles Enc, Rev and MChal, $\mathcal{B}$ simply forwards these queries. Note that for this $\mathcal{B}$ needs to use of the set $\mathsf{copied}$ to map indices from the $\mathsf{IND\text{-}CCA}$ game to the $\mathsf{IND\text{-}CCA}^-$ game. Additionally, it uses $\mathsf{PoK.S}$ to compute member-tags $mt$ and $\mathsf{PoK.Extract}$ to extract from $\mathcal{A}$'s proofs as described above. The main challenge is the simulation of $\mathcal{A}$'s Dec queries, however, we will argue that $\mathcal{B}$'s oracles Upd and $\mathrm{Dec}^-$ allow to simulate $\mathcal{A}$'s view perfectly.

$\mathcal{B}$ simulates decryption queries following the case distinction above. If $\mathcal{A}$ makes a Dec query $(i', c', pk', mt')$ and there exists an index $i$ with $(pk_{par_i}, pk_j) = (pk_{i'}, pk')$, then this was a replay. If this is a half-node, it simply creates a half-node $j$ and adds $j$ to $\mathsf{copied}(i)$. Otherwise, $\mathcal{B}$ forwards this query to $\mathrm{Dec}^-$. If successful, $\mathcal{B}$'s challenger created a node and $\mathcal{B}$ simply forwards the result to $\mathcal{A}$. If decryption failed, $\mathcal{B}$ creates a half-node $j$ locally, adds $j$ to $\mathsf{copied}(i)$ and outputs $\bot$.

For all other Dec queries, $\mathcal{B}$ first queries $\mathrm{Dec}^-$. If successful, $\mathcal{B}$'s challenger created a node and $\mathcal{B}$ again simply forwards the result. Otherwise, it runs the extractor to get $\overline{sk}$ and creates a new node via Upd. Otherwise, it records that the new node is also a half-node and returns $\bot$.

Eventually, $\mathcal{A}$ will output a guess $b'$ and $\mathcal{B}$ will return the same $b'$ to its challenger. If $\mathcal{A}$ does not make any queries that violate its winning condition, $\mathcal{B}$ also does not make such queries. This is due to the fact that by definition $\mathsf{copied}$ sets only contain half-nodes and all other nodes are the same in both experiments. Hence, the full-nodes in set *extd-base* are the same for both $\mathcal{A}$ and $\mathcal{B}$. Further, if $\mathcal{A}$ wins, then so does $\mathcal{B}$. This concludes the proof of Theorem 5. □

## G Proof of Theorem 2

**Proof of Theorem 2.** In Fig. 21 (ignoring all boxes) we instantiate the security notion from Definition 2 with the scheme given in Fig. 9. Note that we compute the second

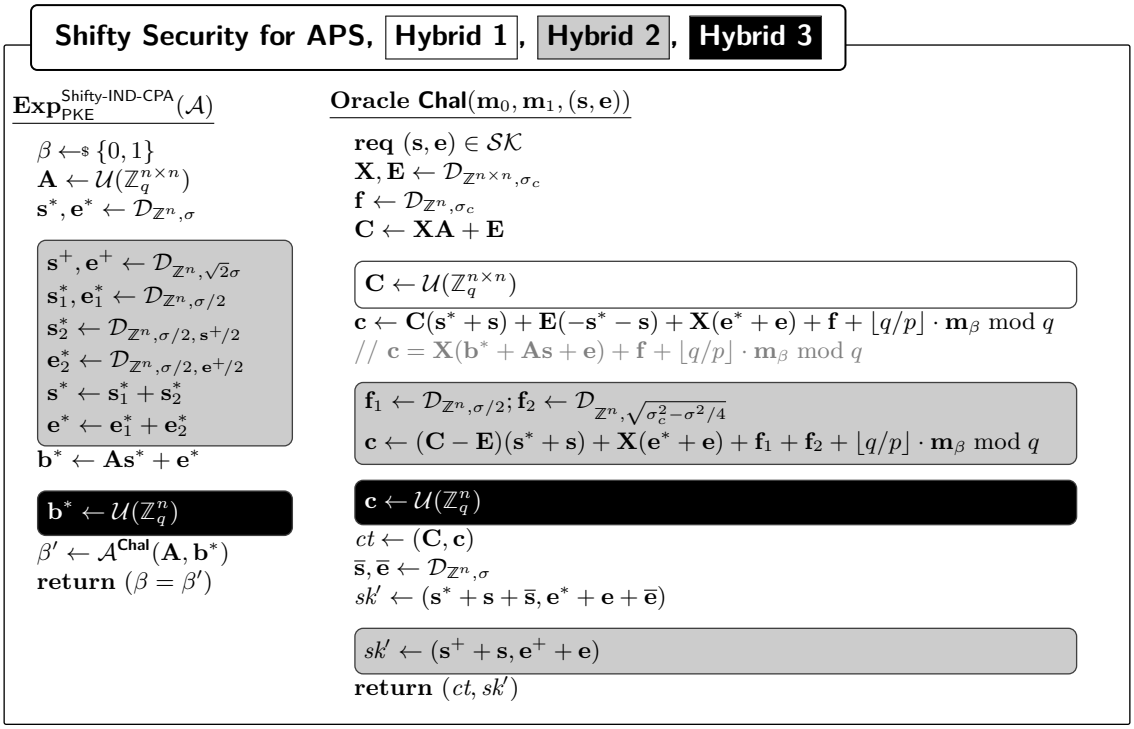



Fig. 21: The Shifty-IND-CPA security game for homomorphic PKE from [APS23] and hybrid games in the proof of Theorem 2: Hybrid 1 only includes the white box, Hybrid 2 additionally the gray boxes and Hybrid 3 contains all boxes.

ciphertext component $\mathbf{c}$ in a different but equivalent way. We proceed via hybrid games, all defined in Fig. 21 as well, and argue that all games are indistinguishable and the last one can only be won with probability 1/2.

**Original game → Hybrid 1.** Indistinguishability of Hybrid 1 is shown via $k$ game hops (where $k$ is the maximal number of **Chal** queries), from $\mathsf{G}_0$, which is Shifty-IND-CPA to $\mathsf{G}_k$, which is Hybrid 1. In $\mathsf{G}_i$, the first $i$ **Chal** queries are answered as in Hybrid 1, while the remaining queries are answered as in the original game.

Thus, the hop from $\mathsf{G}_{i-1}$ to $\mathsf{G}_i$ replaces, in the $i$-th query, the component $\mathbf{C} = \mathbf{X}\mathbf{A} + \mathbf{E}$ of the ciphertext by a uniform element (which also changes the distribution of $\mathbf{c}$). Indistinguishability of the hop follows from the (multi-secret) HNF adaptive extended LWE assumption, which, in addition to a challenge $\mathbf{C}$, provides the adversary with $\mathbf{X}\mathbf{z}_0 + \mathbf{E}\mathbf{z}_1 + \mathbf{f}$ for adversarially chosen short $\mathbf{z}_0$ and $\mathbf{z}_1$ and a Gaussian $\mathbf{f}$. In Fig. 22 we define the reduction $\mathcal{B}^{(i)}$, which, depending on its input, simulates either $\mathsf{G}_{i-1}$ or $\mathsf{G}_i$ to adversary $\mathcal{A}$. W.l.o.g. we assume that $\mathcal{A}$ makes exactly $k$ queries and uses correct values $(\mathbf{s}, \mathbf{e})$.
Note that $\mathcal{B}^{(i)}$'s query satisfies $\|\mathbf{z}_1\|_\infty \leq \|\mathbf{s}^*\|_\infty + \|\mathbf{s}\|_\infty \leq y\sigma + \ell y\sigma$ (by the above abort condition and $(\mathbf{s}, \mathbf{e}) \in \overline{\mathcal{SK}}_{\ell-1}$), which is thus bounded by $B$. Analogously, we have $\|\mathbf{z}_0\|_\infty \leq B$, and thus

$$\left|\mathsf{Adv}^{\mathsf{G}_{i-1}}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{G}_i}(\mathcal{A})\right| \leq \mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,n,n,\sigma_c,B}(\mathcal{B}^{(i)}) \ .$$

Therefore, there exists $\mathcal{B} := \mathcal{B}^{(i)}$ for some $i$ s.t.

$$\left|\mathsf{Adv}^{\mathsf{Hybrid\ 1}}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A})\right| \leq k \cdot \mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,n,n,\sigma_c,B}(\mathcal{B}) \ . \tag{15}$$

$\underline{\mathcal{B}_1^{(i)}(\mathbf{A})}$

$\beta \leftarrow\!\!\$\ \{0,1\}; j \leftarrow 0$
$\mathbf{s}^*, \mathbf{e}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$
$\mathbf{b}^* \leftarrow \mathbf{A}\mathbf{s}^* + \mathbf{e}^*$
**run** $\mathcal{A}^{\mathsf{Chal}}(\mathbf{A}, \mathbf{b}^*)$
**while query** $\mathbf{Chal}(\mathbf{m}_0, \mathbf{m}_1, (\mathbf{s}, \mathbf{e}))$:
  $j \leftarrow j + 1$
  **if** $j = i$
    $st = (\mathbf{A}, \mathbf{s}^*, \mathbf{e}^*, \mathbf{m}_\beta, (\mathbf{s}, \mathbf{e}), \mathcal{A}\text{'s state})$
    **stop and return**
      $(st, \mathbf{z}_0 = \mathbf{e}^* + \mathbf{e}, \mathbf{z}_1 = -\mathbf{s}^* - \mathbf{s})$
  $\mathbf{X}, \mathbf{E} \leftarrow \mathcal{D}_{\mathbb{Z}^{n\times n},\sigma_c}; \mathbf{f} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_c}$
  $\mathbf{C} \leftarrow \mathcal{U}(\mathbb{Z}_q^{n\times n})$
  $\mathbf{c} \leftarrow \mathbf{C}(\mathbf{s}^* + \mathbf{s}) + \mathbf{E}(-\mathbf{s}^* - \mathbf{s}) +$
    $\mathbf{X}(\mathbf{e}^* + \mathbf{e}) + \mathbf{f} + \lfloor q/p \rfloor \cdot \mathbf{m}_\beta \bmod q$
  $\bar{\mathbf{s}}, \bar{\mathbf{e}} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$
  $sk' \leftarrow (\mathbf{s}^* + \mathbf{s} + \bar{\mathbf{s}}, \mathbf{e}^* + \mathbf{e} + \bar{\mathbf{e}})$
  **answer** $((\mathbf{C}, \mathbf{c}), sk')$

$\underline{\mathcal{B}_2^{(i)}(st, \mathbf{C}, \mathbf{h})}$

// $\mathbf{C} = \mathbf{X}\mathbf{A} + \mathbf{E}$ or $\mathbf{C} \leftarrow \mathbb{Z}_q^{n\times n}$
// $\mathbf{h} = \mathbf{X}(\mathbf{e}^* + \mathbf{e}) + \mathbf{E}(-\mathbf{s}^* - \mathbf{s}) + \mathbf{f}$
$\mathbf{c} \leftarrow \mathbf{C}(\mathbf{s}^* + \mathbf{s}) + \mathbf{h} + \lfloor q/p \rfloor \cdot \mathbf{m}_\beta \bmod q$
$\bar{\mathbf{s}}, \bar{\mathbf{e}} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$
$sk' \leftarrow (\mathbf{s}^* + \mathbf{s} + \bar{\mathbf{s}}, \mathbf{e}^* + \mathbf{e} + \bar{\mathbf{e}})$
**answer** $\mathcal{A}$**'s** $i$**-th query:** $((\mathbf{C}, \mathbf{c}), sk')$
**while query** $\mathbf{Chal}(\mathbf{m}_0, \mathbf{m}_1, sk = (\mathbf{s}, \mathbf{e}))$:
  $ct \leftarrow \mathsf{Encrypt}((\mathbf{A}, \mathbf{b}^*) + \mathsf{pk}(sk), \mathbf{m}_\beta)$
  $(\overline{pk}, \overline{sk}) \leftarrow \mathsf{KeyGen}(pp)$
  $sk' \leftarrow sk^* + sk + \overline{sk}$
  **reply with** $(ct, sk')$
**get** $\mathcal{A}$**'s output** $\beta'$
**return** $(\beta = \beta')$

Fig. 22: The reduction $\mathcal{B}^{(i)}$ to (multi-secret) HNF adaptive extended LWE for showing indistinguishability of the hop $\mathsf{G}_{i-1}$ to $\mathsf{G}_i$ between Hybrids 1 and 2.

**Hybrid 1 → Hybrid 2.** This statistical argument follows from the Convolution Lemma (Lemma 1).

We first define an intermediate hybrid as follows. Instead of sampling $\mathbf{s}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$ and $\bar{\mathbf{s}} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$ and defining $\mathbf{s}^+ := \mathbf{s}^* + \bar{\mathbf{s}}$ (implicit in $sk'$), the game samples $\mathbf{s}^+$ directly from $\mathcal{D}_{\mathbb{Z}^n,\sigma'}$ with $\sigma' := \sqrt{2}\sigma = \sqrt{\sigma^2 + \sigma^2}$ which, by Lemma 1, is $\varepsilon' := 2\varepsilon/(1-\varepsilon)$-close to the distribution of $\mathbf{s}^+$ in Hybrid 1. Note that, since $\sigma > \sqrt{8\ln(2n(1+1/\varepsilon))/\pi}$, we have $8/\sigma^2 < \pi/\ln(2n(1+1/\varepsilon))$, and thus the premise of Lemma 1 is satisfied for $\sigma_1, \sigma_2 := \sigma/2$ (required below), and a forteriori for $\sigma_1, \sigma_2 := \sigma$.

The original secret $\mathbf{s}^*$ is now sampled from $\mathcal{D}_{\mathbb{Z}^n,\sigma/\sqrt{2},\mathbf{e}^+/2}$, which in the adversary's view is $\varepsilon'$-close to its distribution in Hybrid 1, as shown in [APS23]: for any $\mathbf{x}$, we have

$$\Pr[\mathbf{s}^* = \mathbf{x}] = \sum_{\mathbf{y}\in\mathbb{Z}^n} \Pr[\mathbf{s}^* = \mathbf{x} \,|\, \mathbf{s}^+ = \mathbf{y}] \Pr[\mathbf{s}^+ = \mathbf{y}] \quad = \sum_{\mathbf{y}\in\mathbb{Z}^n} \mathcal{D}_{\mathbb{Z}^n,\sigma/\sqrt{2}}(\mathbf{x} - \mathbf{y}/2)\mathcal{D}_{\mathbb{Z}^n,\sqrt{2}\sigma}(\mathbf{y})$$
$$= \sum_{\mathbf{y}\in\mathbb{Z}^n} \mathcal{D}_{\mathbb{Z}^n,\sqrt{2}\sigma}(2\mathbf{x} - \mathbf{y})\mathcal{D}_{\mathbb{Z}^n,\sqrt{2}\sigma}(\mathbf{y}) \ ,$$

by the definition of Gaussian distributions (Definition 5). The latter is the evaluation of the convolution (Definition 6) of two Gaussian distributions with $\sigma_1 = \sigma_2 = \sqrt{2}\sigma$, evaluated at $2\mathbf{x}$. By Lemma 1, this convolution is $\varepsilon'$-close to $\mathcal{D}_{\mathbb{Z}^n,\sqrt{\sigma_1^2+\sigma_2^2}} = \mathcal{D}_{\mathbb{Z}^n,2\sigma}$. The above is thus $\varepsilon'$-close $\mathcal{D}_{\mathbb{Z}^n,2\sigma}(2\mathbf{x}) = \mathcal{D}_{\mathbb{Z}^n,\sigma}(\mathbf{x})$, which is how $\mathbf{s}^*$ was sampled in Hybrid 1.

In Hybrid 2, instead of sampling $\mathbf{s}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma/\sqrt{2},\mathbf{s}^+/2}$, we sample it as $\mathbf{s}^* := \mathbf{s}_1^* + \mathbf{s}_2^*$ for $\mathbf{s}_1^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma/2}$ and $\mathbf{s}_2^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma/2,\mathbf{s}^+/2}$, which, again by Lemma 1, is $\varepsilon'$-close. (We have already argued above that the premise of Lemma 1 is satisfied for $\sigma_1, \sigma_2 := \sigma/2$.)

The exact same changes happens to $\mathbf{e}^*$ (becoming $\mathbf{e}_1^* + \mathbf{e}_2^*$) and $\mathbf{e}^+$, which induces another statistical difference of $3\varepsilon'$.

Finally, for all (up to $k$) **Chal** queries, instead of sampling $\mathbf{f} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_c}$, it is defined as $\mathbf{f} := \mathbf{f}_1 + \mathbf{f}_2$ for $\mathbf{f}_1 \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma/2}$ and $\mathbf{f}_2 \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sqrt{\sigma_c^2-\sigma^2/4}}$, which is $\varepsilon'$-close to the distribution of $\mathbf{f}$ in Hybrid 1. (Note that, since $\sigma_c > \sigma/\sqrt{2}$, we have $\sqrt{\sigma_c^2 - \sigma^2/4} > \sigma/2$, and thus, by the above, the premise is again satisfied.) In total, we get:

$$\left|\mathsf{Adv}^{\mathsf{Hybrid}\ 2}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{Hybrid}\ 1}(\mathcal{A})\right| \le (6+k)\cdot 2\varepsilon/(1-\varepsilon) \ . \tag{16}$$

$\underline{\mathcal{C}(\overline{\mathbf{A}}, \overline{\mathbf{b}})}$

$j \leftarrow 0$
**parse** $\overline{\mathbf{A}}$ as $[\mathbf{A}^\top \| \mathbf{D}_1^\top \| \dots \| \mathbf{D}_k^\top]^\top$
**parse** $\overline{\mathbf{b}}$ as $[\mathbf{b}^\top \| \mathbf{d}_1^\top \| \dots \| \mathbf{d}_k^\top]^\top$
// $\mathbf{b} = \mathbf{A}\mathbf{s}_1^* + \mathbf{e}_1^*$ or $\mathbf{b} \leftarrow \mathbb{Z}_q^n$
// $\mathbf{d}_i = \mathbf{D}_i \mathbf{s}_1^* + \mathbf{f}_{i,1}^*$ or $\mathbf{d}_i \leftarrow \mathbb{Z}_q^n$
$\beta \leftarrow\$ \{0,1\}$
$\mathbf{s}^+, \mathbf{e}^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n, \sigma\sqrt{2}}$
$\mathbf{s}_2^* \leftarrow \mathcal{D}_{\mathbb{Z}^n, \sigma/2, \mathbf{s}^+/2}$
$\mathbf{e}_2^* \leftarrow \mathcal{D}_{\mathbb{Z}^n, \sigma/2, \mathbf{e}^+/2}$
$\mathbf{b}^* \leftarrow \mathbf{b} + \mathbf{A}\mathbf{s}_2^* + \mathbf{e}_2^*$
$\beta' \leftarrow \mathcal{A}^{\mathsf{Chal}}(\mathbf{A}, \mathbf{b}^*)$
**return** $(\beta = \beta')$

$\underline{\textbf{Oracle } \mathsf{Chal}(\mathbf{m}_0, \mathbf{m}_1, (\mathbf{s}, \mathbf{e}))}$

$j \leftarrow j + 1$
**req** $(\mathbf{s}, \mathbf{e}) \in \overline{\mathcal{SK}}$
$\mathbf{X}, \mathbf{E} \leftarrow \mathcal{D}_{\mathbb{Z}^{n \times n}, \sigma_c}$
$\mathbf{C} \leftarrow \mathbf{X}\mathbf{A} + \mathbf{E} + \mathbf{D}_j$
$\mathbf{f}_2 \leftarrow \mathcal{D}_{\mathbb{Z}^n, \sqrt{\sigma_c^2 - \sigma^2/4}}$
// $\mathbf{c} = (\mathbf{C} - \mathbf{E})(\mathbf{s}^* + \mathbf{s}) + \mathbf{X}(\mathbf{e}^* + \mathbf{e}) + \mathbf{f}_1$
// $\quad + \mathbf{f}_2 + \lfloor q/p \rfloor \cdot \mathbf{m}_\beta \bmod q$
$\mathbf{c} \leftarrow \mathbf{X}(\mathbf{b} + \mathbf{A}(\mathbf{s}_2^* + \mathbf{s}) + \mathbf{e}_2^* + \mathbf{e}) + \mathbf{d}_j$
$\quad + \mathbf{D}_j(\mathbf{s}_2^* + \mathbf{s}) + \mathbf{f}_2 + \lfloor q/p \rfloor \cdot \mathbf{m}_\beta \bmod q$
$ct \leftarrow (\mathbf{C}, \mathbf{c})$
$sk' \leftarrow (\mathbf{s}^+ + \mathbf{s}, \mathbf{e}^+ + \mathbf{e})$
**return** $(ct, sk')$

Fig. 23: The reduction $\mathcal{C}$ to LWE showing indistinguishability of Hybrids 2 and 3.

**Hybrid 2 → Hybrid 3.** The last hop is by reduction to LWE with parameters $m := (k+1)n$ and variance $\sigma/2$, against which we construct the reduction $\mathcal{C}$ in Fig. 23:
We first analyze the case when $[\mathbf{b}^\top \| \mathbf{d}_1^\top \| \dots \| \mathbf{d}_k^\top]^\top$ follows the LWE-distribution, that is,

$$\mathbf{b} = \mathbf{A}\mathbf{s}_1^* + \mathbf{e}_1^* \qquad\qquad \mathbf{d}_j = \mathbf{D}_j \mathbf{s}_1^* + \mathbf{f}_{j,1} \quad \text{for all } j$$

for $\mathbf{s}_1^*, \mathbf{e}_1^*, \mathbf{f}_{j,1} \leftarrow \mathcal{D}_{\mathbb{Z}^n, \sigma/2}$. Reduction $\mathcal{C}$ then simulates Hybrid 2, since

$$\mathbf{b}^* = (\mathbf{A}\mathbf{s}_1^* + \mathbf{e}_1^*) + \mathbf{A}\mathbf{s}_2^* + \mathbf{e}_2^* = \mathbf{A}(\mathbf{s}_1^* + \mathbf{s}_2^*) + \mathbf{e}_1^* + \mathbf{e}_2^*$$

and moreover for the $j$-th call to **Chal**:

$$\begin{aligned}\mathbf{c} &= \mathbf{X}(\mathbf{A}(\mathbf{s}_1^* + \mathbf{s}_2^* + \mathbf{s}) + \mathbf{e}_1^* + \mathbf{e}_2^* + \mathbf{e}) + \mathbf{D}_j(\mathbf{s}_1^* + \mathbf{s}_2^* + \mathbf{s}) \\ &\qquad + \mathbf{f}_{j,1} + \mathbf{f}_2 + \lfloor q/p \rfloor \cdot \mathbf{m}_\beta \bmod q \\ &= (\mathbf{C} - \mathbf{E})(\mathbf{s}_1^* + \mathbf{s}_2^* + \mathbf{s}) + \mathbf{X}(\mathbf{e}_1^* + \mathbf{e}_2^* + \mathbf{e}) + \mathbf{f}_{j,1} + \mathbf{f}_2 + \lfloor q/p \rfloor \cdot \mathbf{m}_\beta \bmod q\ ,\end{aligned}$$

since $\mathbf{X}\mathbf{A} + \mathbf{D}_j = \mathbf{C} - \mathbf{E}$.

On the other hand, if $\mathbf{b}$ and $\mathbf{d}_j$ are independently uniform then $\mathbf{b}^*$ and $\mathbf{c}$ are independently uniform as well and $\mathcal{C}$ thus simulates Hybrid 3. We thus have

$$\left|\mathsf{Adv}^{\mathsf{Hybrid}\ 3}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{Hybrid}\ 2}(\mathcal{A})\right| \leq \mathsf{Adv}^{\mathsf{LWE}}_{q,n,(k+1)n,\sigma/2}(\mathcal{C})\ . \tag{17}$$

In Hybrid 3, the ciphertext is independent of $\beta$ and thus the adversary's advantage is 0. This, together with Equations (15), (16) and (17) implies the theorem statement. □