Lattice-Based Updatable KEM 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 of an Updatable Key Encapsulation Mechanism (UKEM), implying UPKE, that satisfies their notion with classic security based on the Computational Diffie-Hellman (CDH) assumption in the Random Oracle Model (ROM).

No existing post-quantum UPKE/UKEM construction is known to meet the AFM definition. We present and implement practical secret-key recovery attacks in the AFM adversarial model for all proposed parameter sets of two PQ schemes including the most efficient one to date, due to Abou Haidar, Passelègue and Stehlé (APS, Asiacrypt'23). If the UKEM schemes were used in a real-world group messaging application, the attacks would correspond to realistic execution scenarios, even when targeting a 100% success probability.

Next, we present the first post-quantum UKEM construction meeting (a slight relaxation of) the AFM security notion. When based on the Module-LWE assumption, our construction is more efficient than prior PQ constructions, while achieving stronger security. More concretely, public key sizes are about 1/2 that of APS and ciphertext sizes are about 14% smaller. As the AFM security proof relies on random self-reducibility of CDH, which has no analogue for lattices, we develop a new proof technique for strong UKEM, identifying the core properties required from the underlying (lattice-based) encryption scheme.

Table of Contents

1 Introduction . . . 3
1.1 Technical Overview . . . 5
1.2 On the Forking (In)security of Prior PQ-Secure UKEMs . . . 11
1.3 On the (Un)limited Number of Updates . . . 12
2 Preliminaries . . . 12
3 Shifty-Secure PKE . . . 13
4 Updatable KEM . . . 14
5 Our UKEM Construction . . . 17
5.1 Security of LuKEM . . . 18
6 Instantiation of Building Blocks . . . 22
7 Parameters . . . 25
8 Weak Joiner Security . . . 28
8.1 Extending the Syntax and Security Definition . . . 28
8.2 Our Construction . . . 29
8.3 Security of JUKEM . . . 30
A Further Related Work and Background . . . 33
A.1 (Non-)Requirements and Desired Properties for MLS . . . 33
A.2 Differences in Security Notions . . . 34
A.3 Comparison to ABL25 . . . 34
B Forking Attack on the Parameters of Previous Work . . . 35
B.1 The APS Scheme . . . 35
B.2 The DHW Scheme . . . 37
B.3 Attacking MLS with Existing UPKE Schemes . . . 38
C Additional Preliminaries . . . 39
C.1 PKE Definitions . . . 39
C.2 Proof Systems . . . 40
D UKEM Correctness and Robustness . . . 43
D.1 Robustness of the LuKEM Scheme . . . 44
E Proof of Theorem 1 . . . 45
E.1 Intermediate Security Notions . . . 45
E.2 From Shifty-IND-CPA to IND-M-CPA$^+$ . . . 46
E.3 From IND-M-CPA$^+$ to IND-M-CCA$^-$ by observing the RO . . . 48
E.4 From IND-M-CCA$^-$ to IND-M-CCA using PoK . . . 51
F Proof of Lemma 2 . . . 55
G Proof of Theorem 2 . . . 56
H NIZKPoK . . . 60
H.1 Instantiation from Lattices . . . 63

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 that 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 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, and 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 adds FS guarantees to traditional PKE. Intuitively, in FS-PKE, when decrypting a ciphertext C, the decryption key gets "punctured", which removes its ability to ever decrypt C again.

Unfortunately, to date, the efficiency of FS-PKE constructions remains insufficient 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 was required for 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}. Correctness for UPKE (i.e., the guarantee that honest parties can decrypt each other's ciphertexts) is now 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.[4] In fact, potential real-world UPKE applications (such as MLS) are already formulated as using a KEM (not PKE). Internally, they 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 the same. So, for the remainder of this work, we opt to stick with the slightly simpler UKEM primitive.

[4] To port a UKEM definition to UPKE, we simply replace the real/random key challenge oracle with a chosen-message challenge oracle.

UKEM security. The goal of our work is to build UPKE in the post-quantum setting strong enough for use in multi-party protocols like MLS. The only prior work with a UKEM (or UPKE) security notion sufficient for such applications is the one in [AFM24], in the following denoted by AFM. As it is constructed over discrete-logarithm-hard groups, it only considers the classic setting. We start with adapting their UKEM security notion in three ways that preserve its applicability to multi-party applications. First, we allow challenging fewer generations of an initial key pair. Second, our notion generalizes the correctness property from perfect to "only" being statistical (to accommodate LWE-based constructions). It is easy to see that this has no impact on multi-party UKEM applications such as MLS (which is why ML-KEM, despite also having a non-zero decapsulation error probability, can still be used with such protocols). Finally, we weaken the security for special calls of executions (a property called joiner security) by disallowing corruptions in those executions. Fortunately, as pointed out in [AFM24], for the applications under consideration, joiner security without corruptions is sufficient. We provide more background on the requirements for applications like MLS in Appendix A.1 and details on the differences in security definitions in Appendix A.2.

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, DFV24, ABL25] satisfying post-quantum (PQ) security. While [EJKM22, DFV24] are isogeny-based, both [DKW21, APS23, ABL25] are lattice-based. The remaining PQ constructions rely exclusively on generic primitives with known post-quantum-secure instantiations.

Of all constructions, only [AFM24] is shown to provide sufficient security for higher-level multi-party protocol applications; the rest are analyzed w.r.t. significantly weaker notions. Our construction is most closely related to the one in [APS23] (albeit with significant differences described below), but achieves stronger security. Like [APS23], we rely on the NIZK techniques in [LNP22]. Concurrently to our work, [ABL25] proposed a significantly less efficient UPKE scheme (focusing on other properties).

Practical efficiency. From the perspective of *applied* cryptography, an overarching goal in the study of UPKE is to realize the promised FS gains from replacing PKE with UPKE in MLS *in the wild*.[5] We view our results as making substantial progress towards this goal. Indeed, demonstrating a practically efficient PQ UPKE scheme sufficiently strong for MLS was probably the single biggest hurdle until 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 (for example, by the IETF). Indeed, MLS and every primitive in every one of its standardized ciphersuites are IETF standards. Hardcoding a concrete UPKE/UKEM construction in MLS directly would not be a viable approach, as MLS (like TLS, SSH, IPSec, Wireguard, etc.) is designed to work with a generic ciphersuite to be negotiated at runtime by participants. For classic security, the practical efficiency of the UKEM in [AFM24] makes real progress towards laying the foundations required to build such a standard.

[5] MLS is already in use in several applications including Cisco's Webex and Discord, with many more to come. For example, Google and Apple are working on deploying MLS to billions of mobile devices as the E2EE layer for securing all RCS messages.

However, with the advent of the post-quantum era, it is problematic (at best) to establish standards for new primitives (such as 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 plausibly 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 consider only PQ security, this work also takes a step forward for the wider goals of work on UPKE.

Our Contributions. In summary, we make the following steps on the UKEM level towards providing a PQ solution for MLS with strong security properties:

- **Insecurity of Prior Schemes:** We show that the suggested parameter sets of previous work [DKW21, APS23], which are proven secure in the weaker model, lead to efficient attacks on UPKE in the AFM model. We also explain how this translates to a concrete attack on the FS of MLS.
- **First Secure and Efficient Construction:** We give a generic UKEM construction with an LWE-based instantiation and concrete parameters. To thwart attacks similar to the above, our LWE parameters are larger than in the scheme of [APS23], the most efficient post-quantum scheme to date. However, due to careful optimizations, our scheme has in fact *smaller* ciphertexts and public keys than the scheme from [APS23]; we save around 14% for ciphertexts.
- **Novel Proof Technique:** The proofs of [AFM24] rely heavily on properties of the Diffie-Hellman problem, while the techniques of [APS23] do not extend to the AFM model. So to prove security of our scheme, we build it modularly from random oracles, NIZKs and a PKE with our new property called Shifty security, which we define to modularize the proof. The LWE-based PKE construction is based on [APS23]. However, to prove its Shifty security, we need to extend their techniques to deal with forks, which results in a bigger security loss in the reduction and non-trivial changes to argue about parameters. The proof that Shifty implies security of our scheme involves significant changes to the framework of AFM.
- **Joiner Security:** The AFM model also captures joiner security, aiming for additional security in edge cases of the MLS application. Their scheme requires heavy tools, such as pairings and the algebraic group model, to achieve this property. Instead, we provide a generic construction that combines UKEM and KEM, therefore suitable to be instantiated in the post-quantum setting, to achieve a weak variant of joiner security that is in line with the current MLS specification.

In Section 1.1 we give an overview of our construction, proof technique and joiner security. Section 1.2 discusses the insecurity of prior constructions.

1.1 Technical Overview

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

1. We prove it satisfies 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 build UKEM 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 (a variant of) the PKE of Kyber [BDK+18]. We introduce a security notion for kh-PKE that we call *Shifty-IND-CPA*. 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. This should hold even if the adversary receives $sk^+ := sk + sk'$, where sk corresponds to pk and sk' is freshly sampled.[6]

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 achieve IND-CCA security, we use the Fujisaki-Okamoto [FO99] transform and derive the encryption randomness and the encapsulated key K 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 $pk_{i+1} := pk_i + \overline{pk}$ and decrypting a ciphertext yields m, from which one can derive the key K and $\overline{sk}$, which yields sk_{i+1}.

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 a ciphertext a non-interactive zero-knowledge proof of knowledge (PoK) of $\overline{sk}$ such that $\mathsf{pk}(\overline{sk}) = pk_{i+1} - pk_i$. In the UKEM syntax of [AFM24], such a proof is called a *member-tag*.

To achieve the strong security we aim for, the PoK needs to satisfy *straightline simulation extractability*. We construct such a PoK generically from a (simulation-sound) NIZK (not PoK) and an IND-CPA secure PKE. The PoK's common reference string CRS (which will be part of the UKEM public key, sampled freshly 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 the above. A straight-line extractor can then extract $\overline{sk}$ by decrypting c. We optimize the above PoK construction so that pk_{crs} is replaced by a *random* string; see below.

Generic joiner security. Up to this point, we only prove a relaxed UKEM security notion for our construction that omits the class of executions covered by the joiner-security property. We present a black-box transformation from any scheme satisfying the relaxed notion into one satisfying the full UKEM definition. Compared to AFM, we slightly relax the notion of joiner-security. This relaxation enables the resulting UKEM to have almost the same efficiency as the original one, by only requiring an additional IND-CCA-secure KEM. Technically, the transformation combines a UKEM with a regular KEM akin to how a typical KEM combiner combines two KEM's by running them in parallel and hashing the two keys and ciphertexts (we prove security in the ROM). The only previous scheme achieving (the stronger notion of) joiner security by [AFM24] relies on pairings and statement-malleable NIZK proofs of knowledge, for which we do not have efficient PQ replacements.

An instantiation from LWE. Our starting point is the construction of Abou Haidar, Passelègue and Stehlé (APS) [APS23], in particular, their basic (IND-CPA-only) lattice-

[6] 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 $(c_1, c_2 + \overline{sk} \cdot c_1)$; moreover, sk^+ is uniform and independent. For LWE-based schemes, this is in general not the case.

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

(a) Comparison for concrete parameters, for at most 2^5 updates, where [APS23] achieves 115 bits of security for $k = 1$ child nodes and our scheme achieves 140 bits of security for $k = 25\,000$ child nodes. See details and more parameter sets in Section 7.

	public key	ciphertext	member-tag
[APS23]	4.1 KB	10.8 KB	50.3 KB
[this work]	2.3 KB	3.1 KB	49.6 KB

(b) Generic comparison depending on the following: the security parameter κ and the sizes of: a public key pk and ciphertext c (encrypting κ 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, as well as a proof π for the NIZK used in the NIZKPoK.

	public key[8]	ciphertext	member-tag
[APS23]	$\lvert pk\rvert + \lvert pk_{crs}\rvert$	$\lvert c\rvert + \lvert c_{crs}\rvert$	$\lvert c_{crs}\rvert + \lvert \pi_{\text{APS}}\rvert$
[this work]	$\lvert pk\rvert + \kappa$	$\lvert c\rvert$	$\lvert c_{crs}\rvert + \lvert \pi\rvert$

(c) Statement proved in π and π_{APS} as a proxy for its size. We note that the concrete sizes are still about the same due to stronger security and the resulting need for larger parameters in our scheme, see Section 1.2.

	NIZK statement for π and π_{APS}
π_{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 $\mathsf{pk}(sk) = pk_{i+1} - pk_i$
π [this work]	$(pk_i, pk_{i+1}, pk_{crs}, c_{crs})$ s.t. c_{crs} encr. sk to pk_{crs} and $\mathsf{pk}(sk) = pk_{i+1} - pk_i$

based UPKE (without the updating functionality). 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. This is because every time a public key pk_i is updated to $pk_{i+1} := pk_i + \overline{pk}$, the noise contained in it increases, which for standard Kyber parameters would make decryption impossible after a number of updates.

Their scheme [APS23] is key-homomorphic, and it satisfies *Shifty-IND-CPA* if the adversary can only make *one* challenge query. That is, given a challenge key pk, the adversary can query $(m_0, m_1, \overline{sk})$ and receives an encryption of m_b under key $pk + \mathsf{pk}(\overline{sk})$ and an "updated" secret key[7] $sk + sk'$ for a fresh sk'. (This basically corresponds to the basic UPKE security notion IND-CR-CPA [APS23, Def. 6].)

To guarantee forking security of our UPKE (as explained below), we require Shifty-security against adversaries that can make multiple challenge queries. For the suggested parameters, APS' scheme does not satisfy the notion. The reason is that secret keys are "short" (values sampled from a Gaussian distribution). Thus, the more values $sk_i^+ = sk + sk'_i$ are revealed, the more information about the secret key sk is leaked. (This follows from our attack, described in Section 1.2 and Appendix B, against their UPKE using their suggested parameters.)

To guarantee Shifty security against up to k queries, we show that it suffices to increase the standard deviation σ_{APS} of the distribution of secret keys used by APS to $\sqrt{\frac{k+1}{2}} \cdot \sigma_{\text{APS}}$.

[7] More precisely, the adversary gets $sk + \overline{sk} + sk'$, but since it can add/subtract $\overline{sk}$ itself, we omit it from the technical overview.

[8] The 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 $|pk|$.

Let σ be the parameter used to sample secret keys. To prove one-challenge security, a crucial step in the security proof is to sample the pair (sk, sk^+) differently. Instead of sampling sk, sk' from the standard distribution and setting $sk^+ := sk + sk'$, the reduction first samples sk^+ (with parameter $\sqrt{2}\sigma$) and *afterwards* samples sk with mean $sk^+/2$ (and $\sigma/\sqrt{2}$). In the original proof [APS23, Theorem 2], it can be argued that the joint distribution of (sk, sk^+) sampled this way is statistically close to their original distribution.

This change in the sampling allows the reduction to sample sk^+ (which it gives to the adversary) and embed the unknown sk from an LWE challenge (which can be implicitly shifted by $sk^+/2$). Due to the change of distribution, LWE has to be assumed to hold for an error distribution with parameter $\sigma/\sqrt{2}$ [APS23].

We generalize the proof [APS23] and give a method of first sampling $sk_1^+, \ldots, sk_k^+$ (with the distribution of sk_i^+ depending on that of $sk_1^+, \ldots sk_{i-1}^+$) and, finally, sk with parameter $\sigma/\sqrt{k+1}$, and prove that the joint distribution of $(sk, sk_1^+, \ldots, sk_k')$ is statistically close to their distribution in the original security game. Assuming LWE for the same parameters as APS, we thus need to increase σ by a factor $\sqrt{(k+1)/2}$.

Finally, for the NIZKPoK used by our construction, we instantiate the used PKE with the *standard* IND-CPA scheme of Kyber and use the efficient lattice-based NIZK from [LNP22], which has been argued to be (adaptively) simulation-sound in [BBP23, Lemma 2].

Efficiency improvements over [APS23]. In Table 1 we compare the efficiency of our scheme to [APS23]. We cut the public key size roughly in half by observing that since sk_{crs} is not used in the protocol, the corresponding pk_{crs} can be "fake": 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, but under LWE, Kyber public keys look like random matrices. Modeling the hash function as a random oracle then allows us to embed a real key in pk_{crs}.

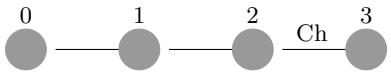
Further, in APS the shift $\overline{sk}$ is freshly sampled and must be encrypted under pk_i, so that the holder of the corresponding sk_i can update sk_i to $sk_{i+1} := sk_i + \overline{sk}$. We avoid this by deriving $\overline{sk}$ as a hash of the encrypted random message. Thus, our ciphertexts are Kyber ciphertexts c encrypting the random message m to pk_i, whereas APS include and additional Kyber ciphertext c_{up} encrypting $\overline{sk}$ to pk_i.

Another source of efficiency improvement 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. Our statement involves only one encryption of the secret key shift instead of two as in APS (cf. Table 1). Note that an encryption of a secret key is larger than an encryption of a short random message.

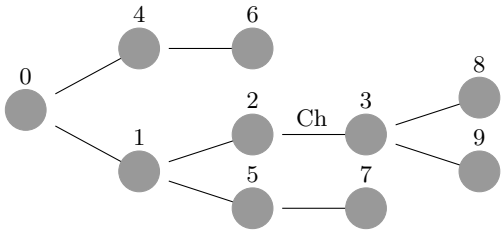
Finally, in contrast to APS, for the CRS, we use a PKE scheme different from the kh-PKE used for updates. Like in APS, the modulus q for the kh-PKE needs to be increased to guarantee correct decryption under (possibly adversarially) updated keys. This increase of key/ciphertext sizes is not needed for the PKE, and we could 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 UKEM keys as a path of nodes containing evolutions (pk_i, sk_i) of a key pair. 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.

To model *asynchronous* executions of the application using UKEM (such as MLS), [AFM24] consider not paths but more general (directed) *trees* (see Fig. 1). The root of a



(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 to 2.



(b) A UKEM execution from the notion of [AFM24]. The adversary defines the structure of the tree and can corrupt any node not resulting 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.

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, 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 in a 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. (See Appendix A.2 for a more detailed description of the variation.) At a high level, a security experiment is defined between 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. A tree with "honest" and "adversarial" edges is created by the adversary's calls to the `Enc` and the `Dec` oracles, respectively. For an honest edge, the key pair of the child node is created by the challenger. For an adversarial edge, the key pair is created by the adversary: the adversary provides a (potentially maliciously created) 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 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. Then pk_2 is updated by the experiment to pk_3 and the adversary receives ("corrupts") the corresponding sk_3.

Our goal is 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 challenger 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 (if they do not lead to a trivial break). 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 fails, as it can no longer embed a Shifty challenge ciphertext in pk_4 (for which it knows the secret key). Thus, to adjust the previous reduction strategy would require guessing the challenge path, which incurs an exponential loss.

To deal with this, we could follow the strategy by [AFM24]. In our schemes 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 must thus guess which public key will be broken first and embeds its challenge there. If the guess was correct then the simulation up to this point was perfect. Note that this guessing strategy only incurs a linear security loss, while allowing the adaptively key corruptions.

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 honest *update* $\overline{pk}_2$, which follows the original distribution. Using Shifty-IND-CPA security, the reduction can then "shift" (challenge) ciphertexts encrypted to $\overline{pk}_2$ to ones encrypted to pk_2.

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 not knowing sk. 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 stronger security of 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).

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 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 $\overline{sk}_1$, must simulate the PoK mt_1. Then, when the adversary asks for the challenge encrypted to pk_2, the reduction asks for a Shifty-IND-CPA challenge c^* (encrypted to $\overline{pk}_1$), which it must shift to $pk_2 = \overline{pk}_1 + (\overline{pk}_2 + pk_0)$. This requires extracting $\overline{sk}_2$ from mt_2. Moreover, the proof for the tag mt_3 attached to c^* again needs to be simulated.

Finally, we note that the NIZK in the scheme of [AFM24] must be simulation-extractable, but not straight-line. This is because their reduction to computational Diffie-Hellman need not extract during the simulation. Instead, it only extracts at the end of the experiment in order to "shift" its CDH solution.

1.2 On the Forking (In)security of Prior PQ-Secure UKEMs

We show that the parameter sets suggested by the works [DKW21, APS23] lead to realistic attacks on messaging protocols like MLS. To this end, we construct two attacks: First, in Appendices B.1 and B.2 we describe and implement secret-key recovery attacks in the AFM adversarial model against both APS [APS23] and DKW [DKW21]. The attacks are practically efficient for all proposed parameters sets. (They even apply to the DDH-based scheme in [DKW21].) Second, in Appendix B.3 we show how to mount the above attacks against MLS in a realistic execution with network forks. The attack removes the benefit of using UKEM instead of KEM in MLS.[9] See also the high-level discussion of forks in MLS executions in Appendix A.1. We stress that these attacks are outside the model of [DKW21, APS23], as they do not consider forks in UKEM executions.

We next give an overview of the key-recovery attack against the parameter set of [APS23]. At a high level, our attack considers a UKEM execution with "forks", that is, the adversary was able to cause a single UKEM key pair to be updated in a few different ways. The attack recovers the original key pair after corrupting the updated secret keys from forks, thus breaking FS.

In more detail, our attacks recover the original secret key sk_0 after getting N updates $sk_1^{(1)}, sk_1^{(2)}, \ldots, sk_1^{(N)}$ of sk_0, thus breaking FS. For all schemes, our attack succeeds with high probability for as few as $N = 50$ forks. We run the attack simulation for a couple of choices for N: for APS, the attack success probability is 100% for $N = 100$, 50% for

[9] Roughly, the authors of [ACDT20] propose to replace MLS's use of PKE by UPKE to improve its FS. However, an execution with network forks may result in multiple independent updates to the same UPKE key pair. Thus, with our attack, the FS of the [ACDT20] protocol in such execution collapses to the security of MLS.

$N = 50$, 8% for $N = 40$ and 0.004% for $N = 30$. For DKW, the attack always succeeds against the DDH scheme for $N = 15$ and against the LWE scheme for $N = 25$.

In APS, sk_0 is a vector of integers in $[-4, 4]$. Updating sk_0 adds to it a vector with elements in $[-4, 4]$, each element sampled from a central binomial distribution (CBD), similar to Gaussian, centered at 0. Thus, for each fork j, the leaked i-th element $sk_1^{(j)}[i]$ is from a CBD centered at $sk_0[i]$. Our attack guesses $sk_0[i]$ as the median of the leaked values $sk_1^{(1)}[i], sk_1^{(2)}[i], \ldots, sk_1^{(N)}[i]$.

In DKW, sk_0 is a vector of ℓ bits and updating it adds a random bit to each element over $\mathbb{Z}$. To simulate the attack, we create forks until for each $i \in [1, \ell]$ two of the leaked keys differ on the i-th coordinate. The value of $sk_0[i]$ must be the smaller of the i-th coordinates of these two.

For [APS23], our results on Shifty-security (cf. Section 1.1) imply that by increasing the standard deviation of the distribution of keys and updates, these simple attacks can be prevented. However, it remains unclear whether this results in a scheme satisfying the AFM-model, which we show for our scheme.

1.3 On the (Un)limited Number of Updates

Recent work by Albrecht, Benčina and Lai [ABL25] has studied UPKE with unlimited updates, which we discuss further in Appendix A.3. Although clearly of interest, in the context of CGKA protocols (like MLS) the practical importance of supporting extremely large numbers of updates for UPKE is less obvious. A primary security goal of CGKA is *post-compromise security* (PCS), meaning security is eventually restored after a party's compromise. PCS requires that parties regularly rotate out their (possibly leaked) UPKE keys, that is, that they sample completely fresh root UPKE key pairs. Note that updating a UPKE key of a party (which is done by another party sending a message to it) does not provide PCS. This dynamic creates a strong incentive for parties to rotate out their UPKE keys as often as possible. Indeed, to quote the MLS RFC9420:

> "*Update messages SHOULD be sent at regular intervals of time as long as the group is active, and members that don't update SHOULD eventually be removed from the group.*"

Following such a policy implies that, when using UPKE in place of PKE, an unbounded number of UPKE updates is not a requirement, since key pairs would be rotated out or removed before the bound on updates kicks in.[10] As further evidence of the desire to avoid stale (U)PKE keys, we refer to [FHT23, CLMT24], which introduce different mechanisms for rotating out old key material.

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 A returns y. A^{O} denotes that A has access to an oracle O.

[10] Moreover, in almost all CGKA protocols (e.g. in MLS), for any given UPKE key pair in use in a CGKA session, the more parties can update the keys, the more parties can rotate them out too.

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. In algorithms and security games, we further use **req** *condition* to denote that if the *condition* is false, then the current function, and any function calling it, stops and returns $\perp$.

Public-Key Encryption. A PKE scheme consists of the following PPT algorithms:

Setup. $pp \leftarrow \mathsf{PKE.Setup}$ outputs public parameters pp.
Key Generation. $(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(pp)$, on input the public parameters, outputs a public key pk and a secret key sk.
Encryption. $c \leftarrow \mathsf{PKE.Encrypt}(pp, pk, m)$, on input a public key pk and a message, outputs a ciphertext c.
Decryption. $m \leftarrow \mathsf{PKE.Decrypt}(pp, sk, c)$, on input a secret key sk and a ciphertext c, outputs a message m.

We provide additional preliminaries for PKE in Appendix C.1.

Non-Interactive Proof System (NIPS). Our scheme uses straightline extractable *Non-Interactive Zero-Knowledge Proof of Knowledge* (NIZKPoK). We formally define it in Appendix C.2 (and construct in Appendix H). A NIZKPoK for an NP-relation $\mathcal{R}$ in the random oracle model contains the following algorithms, all with access to a random oracle H (there are also simulation and extraction algorithms used in the security proof):

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 π that $(x, w) \in \mathcal{R}$.
Verification. $0/1 \leftarrow \mathsf{V}^H(crs, x, \pi)$ verifies a proof π of statement x, i.e., that $(x, w) \in \mathcal{R}$ for some w.

3 Shifty-Secure PKE

The main building block for our UKEM construction is kh-PKE – a PKE with a weak form of key homomorphism. Roughly, let $\mathcal{SK}$ and $\mathcal{PK}$ denote, respectively, the secret and public key spaces of a kh-PKE. Further, let $\mathsf{pk} : \mathcal{SK} \to \mathcal{PK}$ denote the function mapping secret keys to public keys. Our key homomorphism notion requires that $\mathcal{SK}$ is a subspace of some bigger set $\overline{\mathcal{SK}}$, such that $\overline{\mathcal{SK}}$ and $\mathcal{PK}$ are groups, pk is also defined on $\overline{\mathcal{SK}} \setminus \mathcal{SK}$ and $\mathsf{pk} : \overline{\mathcal{SK}} \to \mathcal{PK}$ is a homomorphism. Note that $\mathcal{SK}$ may not even be a group (and it will not be in our instantiation). Formally, a kh-PKE consists of the following algorithms:

Setup. $pp \leftarrow \mathsf{PKE.Setup}$ outputs public parameters pp, which also define the corresponding sets described above.
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.

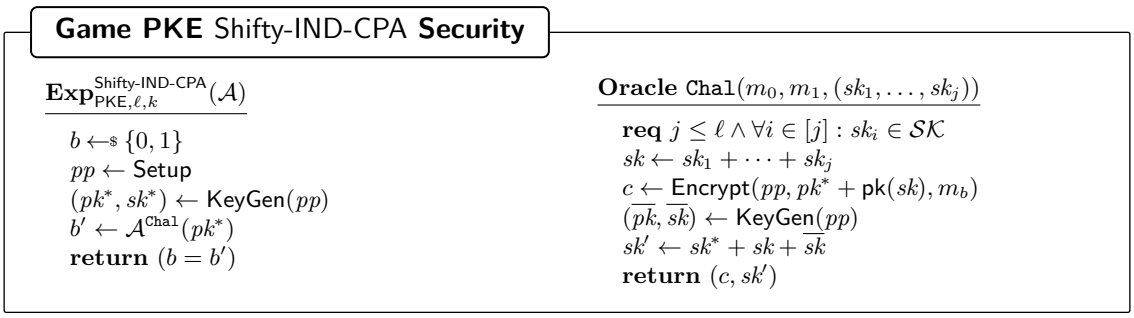


Fig. 2: The Shifty-IND-CPA security game for kh-PKE $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, positive integers ℓ, k and adversary $\mathcal{A}$, where k is the number of queries to `Chal`.

Key homomorphism. We require that kh-PKE schemes have the following property[11]:

Definition 1. *A kh-PKE is key-homomorphic if pp also defines a set $\overline{\mathcal{SK}} \supseteq \mathcal{SK}$ and a function $\mathsf{pk} : \overline{\mathcal{SK}} \to \mathcal{PK}$ with the following properties:*

1. *for all $sk \in \mathcal{SK}$, $\mathsf{pk}(sk)$ is the public key corresponding to sk,*
2. *$(\overline{\mathcal{SK}}, +)$ and $(\mathcal{PK}, +)$ are abelian groups and $\mathsf{pk} : \overline{\mathcal{SK}} \to \mathcal{PK}$ is a homomorphism.*

Correctness. For kh-PKE we require correctness when using a sum of up to ℓ public keys and up to ℓ corresponding secret keys. This is more general than PKE correctness.

Let $\ell > 0$ be an integer and $\delta \geq 0$. Let $pp \leftarrow \mathsf{Setup}$ and let $(pk_t, sk_t) \leftarrow \mathsf{KeyGen}(pp)$ for $t \in [0, \ell]$. For all t and all messages m, we say that kh-PKE PKE is (ℓ, δ)-correct if

$$\Pr[\mathsf{Decrypt}(pp, sk_0 + \ldots + sk_t, \mathsf{Encrypt}(pp, pk_0 + \ldots + pk_t, m)) \neq m] < \delta,$$

where the probability is taken over the random coins of the encryption algorithm.

Security. We define a security notion for kh-PKE required for its use in our UKEM construction. In UKEM security, the adversary is 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. The notion is parameterized by an integer ℓ, denoting the maximum number of updates after which security is still guaranteed.

Definition 2 (Shifty-IND-CPA). *Let $\ell, k > 0$ be integers. Let game Shifty-IND-CPA for kh-PKE be as defined in Fig. 2. The advantage of an adversary $\mathcal{A}$ in that game is defined as*

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

4 Updatable KEM

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.

[11] We emphasize that we do not require any stronger homomorphic properties. In particular, we do not assume any homomorphism on (encrypted) messages.

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}.

We define correctness and robustness in Appendix D.

Remark 1. APS [APS23] use the "long syntax". To update public and secret keys, they have separate algorithms UpdatePK, which returns an *update ciphertext*, and UpdateSK. The long syntax can be easily transformed to the short one by running those algorithms inside Encaps and Decaps. For forward secrecy, keys should be updated after encryption. The short syntax allows to reduce bandwidth by reusing the encryption randomness to derive the update. An additional difference is that the 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].

Member Security. The IND-M-CCA security of UKEM schemes is defined by the experiment in Fig. 3, which we adapted from [AFM24], focusing on *member security*. We removed the parts related to *joiner security*, another notion they define. Its relevance is discussed in Appendix A.2 and we provide a weakened model for joiner security in Section 8.

At a high level, the IND-M-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. Roughly, a half node is created when the member-tag verifies, but the ciphertext does not decrypt correctly.

The key pair of the root node is generated by the challenger using 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 $\mathtt{Enc}(i)$, the challenger "honestly" generates a child node of i with the public key generated using Encaps. If i is a full node, then the secret key of the new node is generated using Decaps (which may result in a half node if 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 $\mathtt{Dec}(i, c', pk', mt')$, 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 Decaps with c' (which could return $\perp$).

MChal edge. The adversary can make one call to the $\mathtt{MChal}$ 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 $\mathtt{Rev}(i)$, which reveals the secret key of node i. IND-M-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 $\mathtt{MChal}$.

Game UKEM IND-M-CCA Security

$\mathbf{Exp}^{\text{IND-M-CCA}}_{\text{UKEM}}(\mathcal{A})$

$pp \leftarrow \mathsf{Setup}$
$(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(pp)$
$(par_0, rev_0, j, typ_0) \leftarrow (\epsilon, 0, 0, \epsilon)$
$(i^*, c^*) \leftarrow (\perp, \perp)$
$b \leftarrow\!\!\$\ \{0,1\}$
$b' \leftarrow \mathcal{A}^{\texttt{Enc},\texttt{Dec},\texttt{Rev},\texttt{MChal}}(pp, pk_0)$
$S \leftarrow \mathsf{chall\text{-}set}(i^*)$
return $(b = b') \wedge (\forall j \in S : \neg rev_j)$

Oracle $\texttt{MChal}(i)$

req $i^* = \perp$
$i^* \leftarrow i$
$K^{(0)} \leftarrow\!\!\$\ \mathcal{K}$
$(K^{(1)}, c^*, mt) \leftarrow \mathsf{create\text{-}honest\text{-}node}(i)$
return $(K^{(b)}, c^*, pk_j, mt, (sk_j = \perp))$

Oracle $\texttt{Enc}(i)$

$(K, c, mt) \leftarrow \mathsf{create\text{-}honest\text{-}node}(i)$
return $(K, c, pk_j, mt, (sk_j = \perp))$

Oracle $\texttt{Rev}(i)$

req $sk_i \neq \perp$
$rev_i \leftarrow 1$
return sk_i

Oracle $\texttt{Dec}(i', c', pk', mt')$

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

Helper $\mathsf{create\text{-}honest\text{-}node}(i)$

req $pk_i \neq \perp$ // i-th node exists
j++
$(K, c, pk_j, mt) \leftarrow \mathsf{Encaps}(pp, pk_i)$
$(par_j, rev_j, typ_j) \leftarrow (i, 0, \text{“Enc”})$
if $sk_i \neq \perp$ **then** // i-th node is *full*
  $(*, sk_j) \leftarrow \mathsf{Decaps}(pp, sk_i, c, pk_j)$
else
  $sk_j \leftarrow \perp$
return (K, c, mt)

Helper $\mathsf{chall\text{-}set}(i^*)$

if $i^* \neq \perp$ **then**
  $base \leftarrow \{i_0, \ldots, i_\ell\}$ where $i_0, \ldots, i_\ell$ is the path from $i_0 = 0$ to $i_\ell = i^*$
else return $\emptyset$
$extd\text{-}base \leftarrow \{i' \mid \exists\, i \in base : pk_{i'} = pk_i\}$
// In [AFM24]: $extd\text{-}base \leftarrow$
// $\quad \{i' \mid \exists\, i \in base : (pk_{i'}, mt_{i'}) = (pk_i, mt_i)\}$
return $\mathsf{dec\text{-}closure}(extd\text{-}base)$

Helper $\mathsf{dec\text{-}closure}(S)$

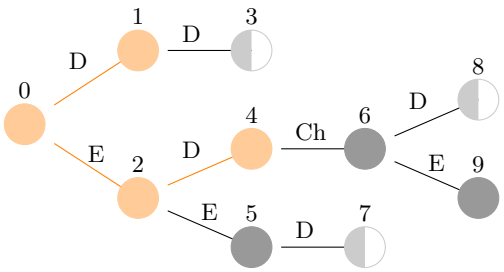
Return the set of all j reachable from some $i \in S$ via only edges created by `Dec` queries.

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

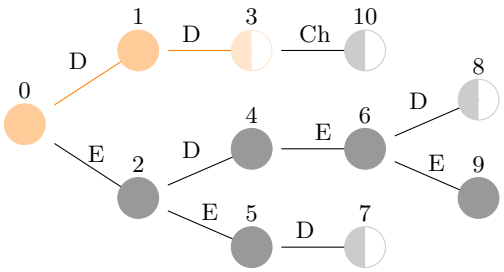
Challenge set. The challenge set S of a node i^* contains the nodes on the challenge path from the root to i^* (corrupting any of these allows for a trivial win for any correct scheme). Moreover, S includes *duplicates* (i.e., nodes that have the same public key) of nodes on the challenge path. Finally, 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.)

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

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



(a) $\mathcal{A}$ queries $\texttt{MChal}(i^* = 4)$, 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 $\texttt{MChal}(i^* = 3)$.

Fig. 4: Example trees in executions of the IND-M-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 **Enc**, a **Dec** or the **Challenge** query. The challenge set is marked in orange. Any node outside of it can be corrupted.

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. We employ the explicit rejection approach and hashing the public key along with m is sufficient for CCA security [HHK17, DHK+21]. To instantiate H_i, one would follow common practice and add prefixes for domain-separation. In Section 6, 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*.

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 $\mathcal{R}$:

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

where pp_{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

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

Setup

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

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

$m \leftarrow\!\!\$\ \{0,1\}^\kappa$
// 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 $(K, c, pk_{i+1}, mt_{i+1} = \epsilon)$

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

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

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

$m \leftarrow \mathsf{PKE.Decrypt}(pp_{\mathsf{PKE}}, sk'_i, c)$
// 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 the key space, PKE KeyGen and Encrypt randomness spaces and message space to be $\{0,1\}^\kappa$.

Construction $\mathsf{LuKEM}[\mathsf{UKEM}, \mathsf{PoK}]$

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

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

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

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

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

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

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

Fig. 6: Our final construction, which achieves UKEM member security (Definition 3) by extending a key-homomorphic UKEM (Definition 4) with a proof of knowledge PoK for the language in Eq. (1).

lattice-based instantiation of PoK in Appendix H. We also prove in Appendix D.1 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 of LuKEM

Our main theorem shows that our UKEMconstruction LuKEM is IND-M-CCA secure.

Theorem 1. *Let* PKE *be a key-homomorphic PKE with message space* $\{0,1\}^\kappa$ *that is* (ℓ, δ)*-correct,* γ*-spread and* Shifty-IND-CPA *secure, and let* PoK *be a straightline extractable*

zero-knowledge proof system for the relation in (1)*. Then* LuKEM[PKE, PoK] *is* IND-M-CCA *secure in the ROM.*

Specifically, for any adversary $\mathcal{A}$ *against* IND-M-CCA *security of* LuKEM *that issues* q_{ro} *RO queries,* q_{enc} `Enc` *queries and* q_{dec} `Dec` *queries and creates a tree of depth at most* ℓ*, where each node has at most* k *child nodes, there exist adversaries* $\mathcal{B}$*,* $\mathcal{B}_1$*,* $\mathcal{B}_2$ *and* $\mathcal{B}_3$ *such that*

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

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

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

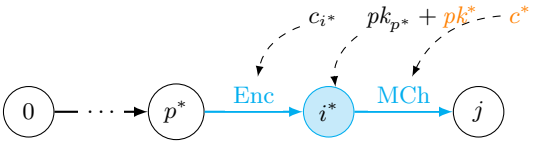
From IND-CPA to IND-M-CPA$^+$. We show that if PKE is Shifty-IND-CPA secure and correct, then $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ is IND-M-CPA$^+$ secure.

Relying on δ-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.

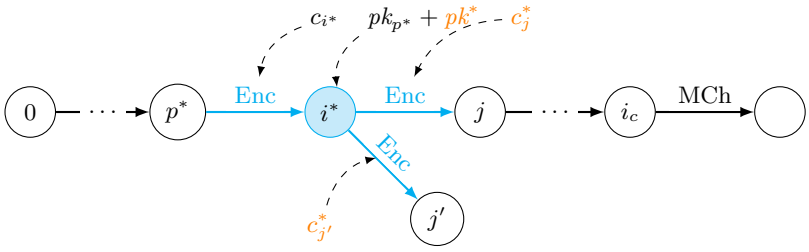
Let $\mathcal{A}$ be an adversary against the IND-M-CPA$^+$ security of $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$. Consider the event Brk that occurs when $\mathcal{A}$ queries (one of) the RO on the message m^* encrypted in c^* returned by `MChal`. Until Brk occurs, the "real" and "random" IND-M-CPA$^+$ 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 $\mathcal{B}$ assuming that $\mathcal{A}$ does not make `Upd` queries. $\mathcal{B}$ is given pk^* and access to the Chal oracle in the Shifty game. It guesses the index i^*, hoping that $\mathcal{A}$ queries $\texttt{MChal}(i^*)$. 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 it to embed its challenge c^*: when $\mathcal{A}$ queries $\texttt{MChal}(i^*)$, $\mathcal{B}$ queries its oracle `Chal` with two random messages $m^{(0)}, m^{(1)} \leftarrow\!\!\$\ \{0,1\}^\kappa$ and the key shift sk_{p^*}[12] (since there are no `Upd` queries, $\mathcal{B}$

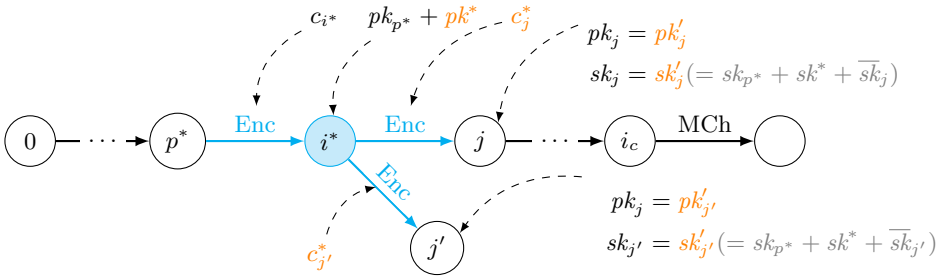
[12] The `Chal` oracle of Shifty-IND-CPA expects a vector of secret keys as input and sums them. For readability, we directly consider this sum. Since the `Upd` oracle requires the same condition on individual secret keys and the maximum number of keys ℓ is that for correctness, the reduction will never 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 $\overline{sk}$.

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 $q_{\text{ro}} \cdot 2^{-\kappa}$ (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 if $\mathcal{A}$ queries m_{i^*} to the RO H_3. But if $\mathcal{A}$ makes this query, then it actually attacked a ciphertext of node p^*! So $\mathcal{B}$ should have guessed differently and embedded pk^* as the update from the parent of p^* to p^*, and c^* as c_{i^*}.

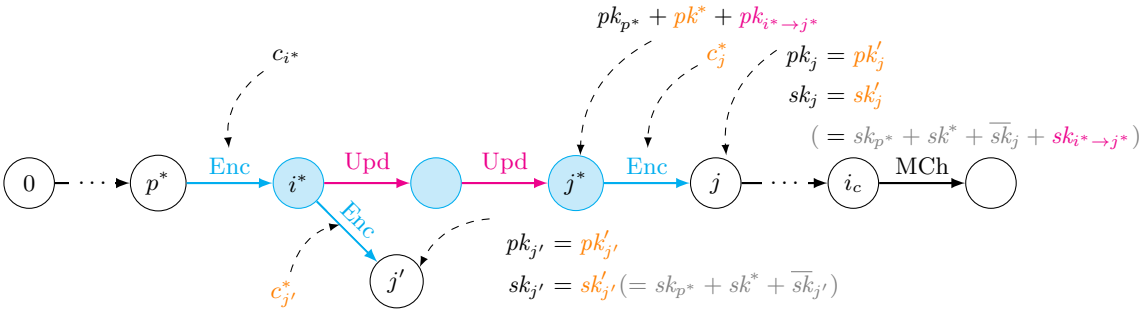


Fig. 8: Adjusting the strategy of the reduction $\mathcal{B}$ to deal with `Upd` queries. Magenta marks edges for which $\mathcal{A}$ (playing in the IND-M-CPA$^+$ game) provides secret keys $\overline{sk}$. Later, these edges will represent `Dec` queries for which the reduction will extract secret keys from ciphertexts and RO queries (for $\mathcal{A}$ in the IND-M-CCA$^-$ game) or PoKs (for $\mathcal{A}$ in the IND-M-CCA game).

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 $\texttt{MChal}(i_c)$. 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 one of the ROs. $\mathcal{A}$ breaking c_j thus 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, for each query $\texttt{Enc}(i^*)$ and $\texttt{MChal}(i^*)$ creating node j, it calls `Chal` on fresh random messages $m_j^{(0)}$ and $m_j^{(1)}$ and embeds the returned ciphertext as c_j. For this strategy we define a "break event" Brk that is more general than the one above; it occurs whenever $\mathcal{A}$ queries any RO on the message encrypted in c_j returned by $\texttt{Enc}(i^*)$ or $\texttt{MChal}(i^*)$ for some i^* on the challenge path. As soon as $\mathcal{A}$ makes a query that contains any $m_j^{(b')}$, $\mathcal{B}$ returns the bit b'. 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 $\texttt{Enc}(i^*)$ or $\texttt{MChal}(i^*)$ which creates a new node j'. If j' does not end up on the challenge path, $\mathcal{A}$ can query $\texttt{Rev}(j')$. Thus, $\mathcal{B}$ needs to generate an updated key pair $(pk_{j'}, sk_{j'})$ 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 with independent randomness, 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 (which create nodes with keys shifted by $\overline{sk}$ chosen by $\mathcal{A}$). First, observe that $\mathcal{B}$ can easily deal with such queries for ancestors of p^*, since it generated the root key itself, so it can emulate the oracle perfectly. (This means it knows sk_{p^*} too.)

When considering `Upd` queries, 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-M-CPA$^+$ defines for an (honest) node

i the set $\mathsf{dec\text{-}closure}(i)$ 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 $j^* \in \mathsf{dec\text{-}closure}(i^*)$ 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 $\mathsf{dec\text{-}closure}(i^*)$.

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 6 and 7), we will explain how Dec queries can be translated to `Upd` queries.

6 Instantiation of Building Blocks

The instantiation of the NIZKPoK is deferred to Appendix H.

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 σ is defined as $\mathcal{D}_{\mathbb{Z}^m,\sigma,\mathbf{c}}(\mathbf{x}) = \rho_{\sigma,\mathbf{c}}(\mathbf{x}) / \rho_{\sigma,\mathbf{c}}(\mathbb{Z}^m)$ with $\rho_{\sigma,\mathbf{c}}(\mathbb{Z}^m) := \sum_{\mathbf{y} \in \mathbb{Z}^m} \rho_{\sigma,\mathbf{c}}(\mathbf{y})$. 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 columns of a matrix in $\mathbb{Z}^{m \times n}$.

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

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

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 X^+ be the distribution of $X_1 + X_2$ and let $\varepsilon > 0$ and set $\bar{\sigma} := \sqrt{\sigma_1^2 + \sigma_2^2}$. If*

$$\sigma_1 \sigma_2 / \bar{\sigma} > \sqrt{\left(\ln(2n(1 + 1/\varepsilon)\right)/\pi}$$

then

$$\Delta(X^+, \mathcal{D}_{\mathbb{Z}^n,\bar{\sigma},\mathbf{c}_1+\mathbf{c}_2}) < 2\varepsilon/(1-\varepsilon) \ .$$

By extending their proof technique, in Appendix F we generalize it to the following.

Lemma 2. *Let $\mathbf{c}_0 \in \mathbb{Z}^n$, let $X \sim \mathcal{D}_{\mathbb{Z}^n,\sigma_0,\mathbf{c}_0}$ and, conditioned on $X = \mathbf{x}$, let $X^+ \sim \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{x}}$. Furthermore, let $Y^+ \sim \mathcal{D}_{\mathbb{Z}^n,\bar{\sigma},\mathbf{c}_0}$, with $\bar{\sigma} := \sqrt{\sigma_0^2 + \sigma^2}$, and, conditioned on $Y^+ = \mathbf{z}$, let $Y \sim \mathcal{D}_{\mathbb{Z}^n,\sigma',c'(\mathbf{z})}$ with $\sigma' := \sigma_0 \sigma / \bar{\sigma}$ and $c'(\mathbf{z}) := \mathbf{c}_0 + \sigma_0^2 / \bar{\sigma}^2 \cdot (\mathbf{z} - \mathbf{c}_0)$.*

If $\sigma' > \sqrt{\left(\ln(2n(1 + 1/\varepsilon)\right)/\pi}$ then the statistical distance between the joint distributions (X, X^+) and (Y, Y^+) is bounded by $\frac{4\varepsilon}{(1-\varepsilon)^2}$.

Definition 7 (LWE assumption). *Let $q, n, m \geq 0$, $\mathcal{S}$ be a distribution on $\mathbb{Z}_q^n$ and χ 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*

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

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 $q, n, m, \ell, B \geq 0$ and χ be an error distribution on $\mathbb{Z}^m$. We define the following game for $\beta \in \{0, 1\}$:*

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

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

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

As for LWE, we write $\mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,m,\ell,\sigma,B}$ for $\mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,m,\ell,\mathcal{D}_{\mathbb{Z}^m,\sigma},B}$. 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 $\sigma/\sqrt{k+1}$, 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*

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

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

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

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 ℓ (called k in APS) subsequent key updates, the decryption failure probability is bounded by δ, we require that for $y := \sqrt{-2\ln(\delta/(4n))}$ we have

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

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.

For security, in terms of a statistical security parameter ε, we require that

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

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

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

Lemma 3. $\Pr\left[\|\mathbf{s}\|_\infty \geq y\sigma \vee \|\mathbf{e}\|_\infty \geq y\sigma\right] < \delta/2$.

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 ℓ times) as

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

The following correctness property was shown in [APS23, Appendix B.2] and γ-spreadness was shown in [APS23, Appendix B.3].

Proposition 1. *Let $\ell > 1$ and $\delta > 0$. Consider the scheme in Fig. 9 with q satisfying (2) for ℓ and δ. Let $sk \in \mathcal{SK}_\ell$ and $\mathbf{m} \in \mathcal{M}$. Then*

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

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[13]) 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 $(\mathbf{A}, \hat{\mathbf{b}})$.

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 $\texttt{Chal}(\mathbf{m}_0, \mathbf{m}_1, (\mathbf{s}, \mathbf{e}))$, $\mathcal{B}$ 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.

A crucial step in the proof [APS23] is for the reduction to first sample $\mathbf{s}^+ := \mathbf{s}^* + \overline{\mathbf{s}}$ (with a higher standard deviation) and then sample $\mathbf{s}^*$ with mean $\mathbf{s}^+/2$. This allows the simulator to return sk' in a `Chal` query even when s^* is sampled by its own challenger in a game hop that relies on LWE. By the convolution lemma (Lemma 1), the distribution if $\mathbf{s}^*$ is statistically close to its original distribution.

In the multi-challenge case, the adversary receives $\mathbf{s}_i^+ := \mathbf{s}^* + \overline{\mathbf{s}}_i$ for independently sampled $\overline{\mathbf{s}}_i$ for $1 \le i \le k$. To apply LWE, we thus need to sample $\mathbf{s}_1^+, \ldots, \mathbf{s}_k^+$ and then $\mathbf{s}^*$ and argue that their joint distribution is statistically close to the distribution in the original game. In Corollary 3 in Appendix G we give such a sampling strategy. For this to work, $\mathbf{s}^*$ will have to be sampled with a smaller standard deviation $\frac{1}{\sqrt{k+1}}\sigma$ (where σ defines the distribution of keys in the scheme). In our scheme we therefore need to increase σ so we can make the LWE assumption for parameter $\frac{1}{\sqrt{k+1}}\sigma$.

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

$$\mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE},\ell,k}(\mathcal{A}) \le k\,\mathsf{Adv}^{\mathsf{mHaeLWE}}_{q,n,n,n,\sigma_c,B}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{LWE}}_{q,n,(k+1)n,\sigma/\sqrt{k+1}}(\mathcal{C}) + \frac{5k\cdot 2\varepsilon}{(1-\varepsilon)^2} + \frac{\delta}{2}\ .$$

We give the proof[14] in Appendix G. From Corollary 1, setting $\gamma := \sigma_c$, $B := (\ell+1)y\sigma)$, and $m', k := \ell$, we immediately get:

Corollary 2. *Let $q \ge 25$, $m \ge 16n + 4\log\log q$, $\sigma \ge \sqrt{8\ln(2(n+1)(1+1/\varepsilon))/\pi}$ and $\sigma_c > \sigma\sqrt{(1+n((\ell+1)y\sigma)^2)/(k+1)}$ 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*

$$\mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE},\ell,k}(\mathcal{A}) \le 4kn\cdot\mathsf{Adv}^{\mathsf{LWE}}_{q,n,m,\sigma/\sqrt{k+1}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{LWE}}_{q,n,(k+1)n,\sigma/\sqrt{k+1}}(\mathcal{C}) + \frac{8kn\cdot 2\varepsilon}{1-\varepsilon} + \frac{5k\cdot 2\varepsilon}{(1-\varepsilon)^2} + \frac{\delta}{2}\ .$$

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_η defined in

[13] 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 $\mathbf{b}' - \mathbf{A}\mathbf{s}'$.

[14] For simplicity, we assume the `Chal` oracle directly accepts keys from the extended key space $\mathcal{SK}_{\ell-1}$ rather than vectors of standard secret keys that are then summed up.

Table 2: Size comparison for different parameter sets, where λ is the security level, $\delta \approx 0$ for all schemes based on the number of updates ℓ and k is the number of child nodes. All sizes are without potential optimizations or compression. Further parameters are $p = 2\eta+1$ and $B = \log_2(q)n\eta$. Our numbers refer to the instantiation of LuKEM.

	λ	q	n	d	η	ℓ	k	$\lvert pk\rvert$	$\lvert ct\rvert$	$\lvert mt\rvert$
[APS23]	115	$\approx 2^{21}$	3	256	2	2^5	1	4.1 KB	10.8 KB	8.1 + 42.2 KB
[this work]	140	$\approx 2^{24}$	3	256	317	2^5	25,000	2.3 KB	3.1 KB	9.2 + 40.4 KB
[APS23]	125	$\approx 2^{26}$	4	256	2	2^{10}	1	6.7 KB	20.8 KB	16.6 + 51.3 KB
[this work]	147	$\approx 2^{29}$	4	256	317	2^{10}	25,000	3.7 KB	4.6 KB	18.6 + 47.7 KB
[APS23]	105	$\approx 2^{31}$	2	512	2	2^{15}	1	8.0 KB	17.9 KB	11.9 + 51.2 KB
[this work]	120	$\approx 2^{34}$	2	512	317	2^{15}	25,000	4.4 KB	6.5 KB	13.1 + 48.8 KB
[APS23]	137	$\approx 2^{36}$	3	512	2	2^{20}	1	13.9 KB	36.9 KB	27.6 + 66.5 KB
[this work]	155	$\approx 2^{39}$	3	512	317	2^{20}	25,000	7.5 KB	10.0 KB	30.0 + 61.3 KB

[BDK+18] with $\eta > 0$ and elements contained in $[-\eta, \eta]$. 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.

More concretely, [APS23] provides four different parameter sets for different numbers of updates, see Table 2, which all use $\eta = 2$, as done in [BDK+18]. In the table we add the additional parameter k, which is the number of child nodes. The scheme in [APS23] is only proven for $k = 1$. As demonstrated in Appendix B, we can leverage larger values of k for concrete attacks, which is why our scheme will use larger parameters.

In terms of sizes, note that one can use the bit dropping technique that is used in Kyber to obtain more compact ciphertexts. For better comparison, we recompute the sizes for their parameters without any optimizations or compression, i.e., the size of an element in R_q is computed as $\log(q)d$.

Our Parameters. We take into account a larger number of child nodes, namely $k = 25,000$, since MLS targets large groups. We use $\eta = \lceil 2 \cdot \sqrt{k+1} \rceil$ (based on the value of σ in Corollary 1). We keep n and d the same as [APS23], but use larger q to guarantee correctness after ℓ updates. We then compute security levels using the lattice estimator [APS15] (commit `66771ec` from January 12, 2026).

Using these parameters, 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:

$$c = \mathbf{x}^\top \mathbf{A} + \mathbf{e}^\top, \quad d = \mathbf{x}^\top \mathbf{b} + f + \lfloor q/p \rfloor \cdot m, \quad K = H_1(pk, m), \tag{3}$$

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.[15] Hence, we get an additional

[15] 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.

ciphertext $(c_0, d_0) \in R_q^{n\times n} \times R_q^n$ for public key $pk_{crs} = (\mathbf{A}_0, \mathbf{b}_0)$ [16]computed as

$$c_0 = \mathbf{X}_0\mathbf{A}_0 + \mathbf{E}_0, \quad d_0 = \mathbf{X}_0\mathbf{b}_0 + \mathbf{f}_0 + \lfloor q/p \rfloor \cdot \mathbf{s}, \tag{4}$$

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, π), where π is the proof for the statement in Eq. (20). 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 π using the LaZer library [LSS24]. LaZer lets us specify lattice relations based on which it automatically creates parameters of a proof system along with proof sizes, using either the LNP framework [LNP22] or LaBRADOR [BS23]. We run LaZer on our relation from Appendix H for all four parameter sets and obtain the proof sizes shown as the second value in the last column of Table 2. 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. Remarkably, our sizes are comparable to (and even slightly improve upon) those of APS, even though we allow for a large k. For completeness, we describe below how the sizes of the APS scheme are derived.

Sizes of APS Scheme. The scheme from [APS23] (when adding a NIZK) achieves IND-CU-CCA security which is a weaker notion than our IND-CCA definition. Yet, to achieve this, they use more expensive building blocks, which we explain below. The APS ciphertext (c, d) is the same as above (since we rely on their base scheme), however, for notational consistency we also 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) as in Eq. (4).

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 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 π_{APS} for the following relation $\tilde{\mathcal{R}}_{\text{APS}}$:

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

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

Here B and B' are computed in the same way as in our scheme, but for $\eta = 2$, and we give estimates on the NIZK sizes using LaZer. The proof sizes here are slightly larger as statements contain more relations, even though our q is larger.

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 27.6 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.

8 Weak Joiner Security

In this section we show how to strengthen the security of UKEM schemes. Informally, joiner security protects parties that join groups with malformed states created by the adversary. The notion was introduced by [AFM24] and is specifically relevant for applications like MLS. Here, we consider a variant of joiner security that is weaker than the original one, which does not allow the adversary to corrupt secret keys. Such a notion is still very meaningful for applications. Indeed, as the authors of [AFM24] also note, it "is sufficient to prove that MLS with UPKE achieves the same fake-group security as today's MLS with PKE".

The weaker notion we consider allows for much more efficient post-quantum secure constructions. Recall that the scheme in [AFM24] relies on statement-malleable NIZK proofs of knowledge, which they instantiate directly in the algebraic group model, for which we do not have an analogue in the lattice setting.

8.1 Extending the Syntax and Security Definition

We recall the extended UKEM syntax of [AFM24] that allows to define joiner security. In particular, the key generation and encapsulation algorithms include joiner tags and there is an additional algorithm to verify joiner tags. The other algorithms stay the same.

Key Generation. $\mathsf{KeyGen}(pp) \to (pk_0, sk_0, jt_0)$, on a public parameter, outputs a key pair (pk_0, sk_0) and the first joiner tag jt_0.

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

Verification of joiner tags. $\mathsf{Verify}_{\mathsf{jt}}(pp, pk_0, pk_{i+1}, jt_{i+1}) \to 0/1$ verifies the update from pk_0 to pk_{i+1} using the tag jt_{i+1}.

For completeness, we give the relaxed security notion in Fig. 10.[17]

Definition 9 (UKEM Security). *Let* $\mathrm{Exp}^{\mathsf{IND\text{-}J\text{-}CCA}}_{\mathsf{UKEM}}(\mathcal{A})$ *be as defined in Fig. 10. The advantage of an adversary* $\mathcal{A}$ *against* IND-J-CCA *security of* UKEM *is defined as*

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

[17] While we could simplify the definition by dropping the reveal oracle, we copy the notion from [AFM24] for clearer comparison.

Game UKEM IND-J-CCA Security

$\mathbf{Exp}^{\text{IND-J-CCA}}_{\text{UKEM}}(\mathcal{A})$

$pp \leftarrow \mathsf{Setup}$
$(pk_0, sk_0, jt_0) \leftarrow \mathsf{KeyGen}(pp)$
$(par_0, rev_0, j, typ_0) \leftarrow (\epsilon, 0, 0, \epsilon)$
$(pk^*, c^*) \leftarrow (\bot, \bot)$
$b \leftarrow\!\!\$\ \{0,1\}$
$b' \leftarrow \mathcal{A}^{\texttt{Enc},\texttt{Dec},\texttt{Rev},\texttt{JChal}}(pp, pk_0, jt_0)$
return $(b = b') \wedge (\forall \ell \in [j] : \neg rev_\ell)$

Oracle $\texttt{JChal}(pk', jt')$

req $pk^* = \bot$
req $\mathsf{Verify}_{\mathsf{jt}}(pk_0, pk', jt')$
$pk^* \leftarrow pk'$
$K^{(0)} \leftarrow\!\!\$\ \mathcal{K}$
$(K^{(1)}, c^*, pk, mt, jt) \leftarrow \mathsf{Encaps}(pp, pk', jt')$
return $(K^{(b)}, c^*, pk, mt, jt)$

Oracle $\texttt{Enc}(i)$

$(K, c, mt) \leftarrow$ create-honest-node(i)
return $(K, c, pk_j, mt, jt_j, (sk_j = \bot))$

Oracle $\texttt{Rev}(i)$

req $sk_i \neq \bot$
$rev_i \leftarrow 1$
return sk_i

Oracle $\texttt{Dec}(i', c', pk', mt', jt')$

req $pk_{i'} \neq \bot$ // i-th node exists
if $pk^* \neq \bot$ **then** // implies $c^* \neq \bot$
  req $c^* \neq c' \vee pk^* \neq pk_{i'}$
req $\mathsf{Verify}_{\mathsf{mt}}(pp, pk_{i'}, pk', mt')$
req $\mathsf{Verify}_{\mathsf{jt}}(pp, pk_0, pk', jt')$
j++
$(pk_j, sk_j, jt_j) \leftarrow (pk', \bot, jt')$
$(par_j, rev_j, typ_j) \leftarrow (i', 0, \text{“Dec”})$
if $sk_{i'} \neq \bot$ **then**
  $(K, sk_j) \leftarrow \mathsf{Decaps}(pp, sk_{i'}, c', pk')$
  return K
return $\bot$

Helper create-honest-node(i)

req $pk_i \neq \bot$ // i-th node exists
j++
$(K, c, pk_j, mt, jt_j) \leftarrow \mathsf{Encaps}(pp, pk_i, jt_i)$
$(par_j, rev_j, typ_j) \leftarrow (i, 0, \text{“Enc”})$
if $sk_i \neq \bot$ **then** // i-th node is *full*
  $(*, sk_j) \leftarrow \mathsf{Decaps}(pp, sk_i, c, pk_j)$
else
  $sk_j \leftarrow \bot$
return (K, c, mt)

Fig. 10: Weak joiner CCA security for $\mathsf{UKEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps}, \mathsf{Verify}_{\mathsf{mt}}, \mathsf{Verify}_{\mathsf{jt}})$. By default, all variables are initialized to $\bot$. The difference to [AFM24] is highlighted in blue (see also Footnote 17).

8.2 Our Construction

We extend LuKEM (or more generally, any UKEM) to provide joiner security. At a high level, our construction JUKEM (see Fig. 11) combines a UKEM and a KEM similarly to how typical KEM combiners combine two KEMs. A public key consists of an evolving UKEM public key upk_i and a static KEM public key spk. Encapsulation runs the encapsulation algorithms of both UKEM and KEM and computes the output key K by hashing the two keys, two ciphertexts and two public keys. The UKEM encapsulation also updates upk_i to to upk_{i+1} and generates a member tag mt_{i+1}. The updated public key in our scheme is (upk_{i+1}, spk) and the member tag is mt_{i+1}. To verify the updated key against the root key, we simply check that the static KEM public key is the same. The idea is that if there are no corruptions at all, then the static KEM secret key ssk stored in all secret keys is never

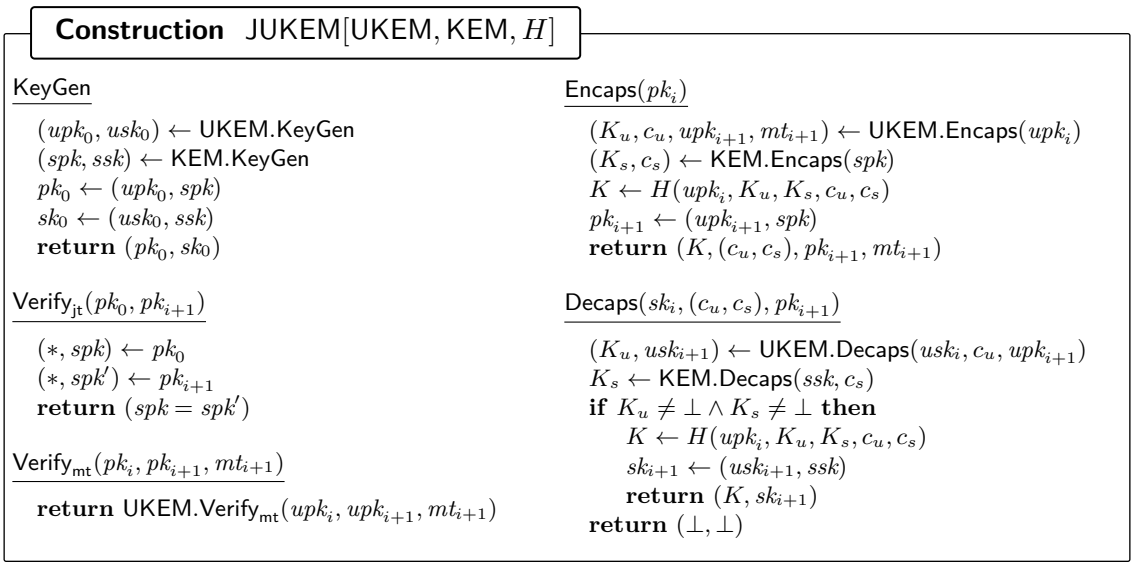


Fig. 11: UKEM construction with weak joiner security.

leaked. So, in this case, any "joiner" party that uses pk_{i+1} after verifying that it contains the same KEM public key spk as the root key pk_0 can rely on the security of the KEM for the security of encapsulated keys. We formalize this below, where we also show that JUKEM has member security as long as UKEM has member security.

8.3 Security of JUKEM

The following two theorems formalize the fact that our construction guarantees, respectively, member and joiner security.

Theorem 3. *If* KEM *is* IND-CCA *secure, then* JUKEM *defined in Fig. 11 is joiner-secure according to Definition 9 in the ROM. Specifically, for any adversary $\mathcal{A}$ there exists a reduction $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}J\text{-}CCA}}_{\mathsf{JUKEM}}(\mathcal{A}) \leq 2 \cdot (\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{B}) + q_{\mathrm{ro}} \cdot 2^{-\kappa}).$$

Theorem 4. *If* UKEM *is* IND-M-CCA *secure according to Definition 3, then* JUKEM *defined in Fig. 11 is also* IND-M-CCA *secure according to Definition 3 in the ROM. Specifically, for any adversary $\mathcal{A}$ there exists a reduction $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}M\text{-}CCA}}_{\mathsf{JUKEM}}(\mathcal{A}) \leq 2 \cdot (\mathsf{Adv}^{\mathsf{IND\text{-}M\text{-}CCA}}_{\mathsf{UKEM}}(\mathcal{B}) + q_{\mathrm{ro}} \cdot 2^{-\kappa}).$$

Proof (sketch, of Theorem 3). Observe that without corruptions and assuming that UKEM is insecure, the joiner security game with our scheme essentially collapses to the KEM IND-CCA security game. We can remove the Rev oracle, which has no effect, and the Enc oracle, which the adversary can emulate itself. The $\mathsf{Verify}_{\mathsf{jt}}$ checks in Dec and JChal make sure that the public key they use is the same as the public key generated by the challenger. Thus, they correspond to KEM IND-CCA challenge and decrypt oracles.

Observe further that the KEM in the above game is the standard KEM combiner where the secure KEM is KEM and UKEM is the insecure KEM. The work [GHP18] proves that this combiner is IND-CCA secure in the ROM. (We get the security loss by combining [GHP18, Theorem 1] with [GHP18, Example 3] and [GHP18, Lemma 6]) □

Proof (sketch, of Theorem 4). The proof is identical to the proof of [GHP18, Theorem 1]. □

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, as well as the anonymous reviewers for their helpful feedback, including pointing out a gap in a previous version of this paper.

References

[ABL25] Martin R. Albrecht, Benjamin Bencina, and Russell W. F. Lai. Hollow LWE: A new spin - unbounded updatable encryption from LWE and PCE. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025 , Part VIII*, volume 15608 of *LNCS*, pages 363–392. Springer, 2025.

[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.

[AJM22] Joël Alwen, Daniel Jost, and Marta Mularczyk. On the insider security of MLS. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 34–68. Springer, Cham, August 2022.

[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 2023, 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 *EuroS&P 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.

[CLMT24] Céline Chevalier, Guirec Lebrun, Ange Martinelli, and Abdul Rahman Taleb. Quarantined-TreeKEM: A continuous group key agreement for MLS, secure in presence of inactive users. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2400–2414. ACM Press, October 2024.

[DFV24] Max Duparc, Tako Boris Fouotsa, and Serge Vaudenay. SILBE: An updatable public key encryption scheme from lollipop attacks. In Maria Eichlseder and Sébastien Gambs, editors, *SAC 2024, Part I*, volume 15516 of *LNCS*, pages 151–177. Springer, Cham, August 2024.

[DHK+21] Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, and Gregor Seiler. Faster lattice-based KEMs via a generic Fujisaki-Okamoto transform using prefix hashing. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2722–2737. ACM Press, November 2021.

[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.

[FHT23] Elsie Mestl Fondevik, Britta Hale, and Xisen Tian. Guardianship in group key exchange for limited environments. Cryptology ePrint Archive, Report 2023/1761, 2023.

[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.

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 190–218. Springer, Cham, March 2018.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[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.

[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 *ACM CCS 2024*, CCS '24, page 3125–3137, New York, NY, USA, 2024. Association for Computing Machinery.

[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 Further Related Work and Background

Here, we discuss additional details on related work and its background. In particular, we discuss forward secrecy, forking attacks and updates in MLS, as well as the security definitions for UKEM and the construction from [ABL25].

A.1 (Non-)Requirements and Desired Properties for MLS

On MLS's forward secrecy. As mentioned in the introduction, the weakness of MLS regarding forward secrecy (of TreeKEM really) was first observed in [ACDT20]. The authors then show that, in TreeKEM, replacing (H)PKE with UPKE fixes the issue (at least in a simplified model, which is why that work was able to use a weaker notion of UPKE than considered here). To complete the picture, in [ACDT21], it is shown that rTreeKEM can be extended to an MLS-like group messaging protocol in almost exactly the same way that MLS is built from TreeKEM.

As discussed in [ACDT20], in MLS, after a compromised group member Updates (or Commits), confidentiality against *network adversaries* is restored, but not *forward secrecy*. This is because future corruptions could still leak one of the HPKE secret keys to which the commit secret s_{commit} in the healing Commit was encrypted. Once that happens, the adversary can recover the s_{commit} and combine it with the previously leaked (or derived) s_{init} to obtain the epoch secret s_{epoch} for the epoch after the healing Commit. In particular, this breaks confidentiality for all messages sent in the (supposedly "healed") epoch. Such attacks on MLS are reflected in TreeKEM's security predicate in [AJM22, Fig. 6].

On forking attacks. To see why forking attacks are relevant when UPKE is used in CGKA (e.g. inside MLS) note that a forking attack can be mounted whenever two or more parties have the same UPKE secret key secret key. This is a standard occurrence in MLS where (U)PKE secret keys are, by design, regularly distributed to multiple group members. An adversary can use such settings to mount a UPKE forking attack by, for example, delivering distinct (even honestly generated) Commits to parties with the same UPKE secret keys. This results in each receiving party updating their copy of the UPKE secret key with different coins from the respective update paths of the Commit they received resulting in more than one child UPKE keys for the original shared secret key.

On the (un)limited number of updates. As we argue in Section 1.3, while unlimited updates are clearly of interest in general, their practical value, especially in CGKA and

MLS, is not obvious. The reason is that many updates occur only if a UPKE key pair is used a lot of times. On the other hand, most applications require that (U)PKE key pairs are regularly rotated, i.e., replaced by fresh random (root) (U)PKE keys. This is necessary for PCS, i.e., restoring confidentiality in case the (U)PKE secret key is compromised.

A.2 Differences in Security Notions

The security notion used in this work is strictly stronger than prior notions, such as the ones from [DKW21, APS23], which do not account for forking attacks. However, it is slightly weaker than the one in [AFM24]. The main differences are that we disallow slightly more corruption queries and we do not consider full joiner security. At the same time, 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 "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.[18]

Weaker joiner security. We adapt 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 with "full" joiner security seems to require either a significant breakthrough in lattice-based NIZK technology or a fundamentally new approach to UPKE/UKEM constructions.

We note that practical protocols like MLS do include mechanisms aimed at providing fake group security (cf. "parent hash" [BBR+23]). Our weakened joiner security exactly aligns with those mechanisms.

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.

A.3 Comparison to ABL25

Recently, the concurrent work of [ABL25] presented a lattice-based UPKE scheme satisfying the IND-CR-CPA security defined in [APS23] (described in Section 6). Unlike our

[18] 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, their scheme supports an unbounded number of key updates. However, as discussed in [AFM24], the IND-CR-CPA security notion falls significantly short of providing the security guarantees required for use in CGKA-style protocols, and, as discussed above, an unbounded number of key updates is not strictly necessary for the application. At the same time, this property seems to make the scheme (or rather, its concrete parameters) resistant against the forking attack.

The scheme in [ABL25] is also less efficient than the one we propose. For example, their ciphertexts (and "update tokens", which in our UKEM syntax are included in the ciphertext) are about 50 times larger: 1500-3600 KiB (cf. Table 1 in [ABL25]) compared to 55-100 KiB in our scheme. Moreover, the only known method [APS23] for boosting IND-CR-CPA to a stronger notion allowing for malicious senders (a necessary but insufficient step towards being able to use UPKE in CGKA protocols) requires an additional NIZK, which would add further costs to their scheme.

B Forking Attack on the Parameters of Previous Work

We present efficient key-recovery attacks on prior schemes. In particular, we show that their proposed parameter sets cannot withstand the additional leakage introduced by forks, as explained in Section 1.2. This, in turn, justifies that our larger parameter sets are inherent to the stronger security level we target.

B.1 The APS Scheme

We give a Python script that simulates the simple attack on the parameters of APS. The script assumes a file `rfc.py` with the code copied from the Kyber RFC draft https://www.ietf.org/archive/id/draft-cfrg-schwabe-kyber-04.html#name-machine-readable-specificat. We keep the notation from that specification. In particular, we note that `k` below refers to the Kyber parameter (that we call n), which is different from variable k used in the main body to denote the maximum number of child nodes.

```
from rfc import sampleNoise, q
import random
import sys

def tryAttack(secretKey, eta1, k, numberOfForks):
    # Track leakage values for each coordinate - create empty frequency
    # dictionary for each secret key coefficient to count leaked values
    totalLeakage = [{} for _ in secretKey]

    # Simulate multiple key updates
    for _ in range(numberOfForks):
        newLeakage = upkeUpdateKey(secretKey, eta1, k)

        # Count occurrences of each leakage value
        for i in range(len(newLeakage)):
            l = newLeakage[i]
            totalLeakage[i][l] = totalLeakage[i].get(l, 0) + 1

    # Guess secret key from leakage statistics
    keyGuess = list(map(guessCoordinate, totalLeakage))
    return secretKey == keyGuess

def guessCoordinate(totalLeakageCoordinate: dict):
    # Sort leakage values by value
    l = sorted(totalLeakageCoordinate.items(), key=lambda x: x[0])
```

```
26
27      # Remove outliers for even-length lists
28      if len(l) % 2 == 0:
29          firstCount = l[0][1]
30          lastCount = l[-1][1]
31
32          if firstCount < lastCount: del l[0]
33          elif firstCount > lastCount: del l[-1]
34          else:
35              if random.randrange(2) == 0: del l[0]
36              else: del l[-1]
37
38      # Return middle value
39      return l[(len(l) - 1) // 2][0]
40
41  def upkeUpdateKey(secretKey, eta1, k):
42      # Generate noise and add to secret key
43      delta = sampleNoise(random32Bytes(), eta1, 0, k).ps
44      delta = flattenNoise(delta)
45      delta = normaliseNoise(delta)
46      return [s + d for s, d in zip(secretKey, delta)]
47
48  def random32Bytes():
49      return bytes([random.randrange(256) for _ in range(32)])
50
51  def flattenNoise(noise):
52      return [x for y in noise for x in y.cs]
53
54  def normaliseNoise(noise):
55      return [x - q if x > q / 2 else x for x in noise]
56
57  def main(numberOfForks, eta1, k):
58      # Generate target secret key
59      secretKey = sampleNoise(random32Bytes(), eta1, 0, k).ps
60      secretKey = flattenNoise(secretKey)
61      secretKey = normaliseNoise(secretKey)
62
63      # Run attack simulation
64      successes = 0
65      numberOfAttempts = 100000
66
67      for _ in range(numberOfAttempts):
68          if tryAttack(secretKey, eta1, k, numberOfForks):
69              successes += 1
70
71      # Display results
72      print("number of forks", numberOfForks)
73      success_rate = float(successes) / numberOfAttempts * 100
74      print("attack success rate", success_rate, "%")
75
76  if __name__ == "__main__":
77      # Parse command line arguments or get user input
78      if len(sys.argv) == 4:
79          numberOfForks = int(sys.argv[1])
80          eta1 = int(sys.argv[2])
81          k = int(sys.argv[3])
82      else:
83          numberOfForks = int(input("Enter number of forks: "))
84          eta1 = int(input("Enter eta1: "))
85          k = int(input("Enter k: "))
86      main(numberOfForks, eta1, k)
```

B.2 The DHW Scheme

We give a Python script that simulates the simple attack on the parameters of the DHW schemes [DHW23]. In DHW, both the DDH-based and LWE-based constructions use binary secret keys, so the attack applies to both. The attack simulation generates forks until it fully recovers the secret key and outputs the number of forks required. Its efficiency depends only on the key length of the key, which in turn depends on parameters. One could try to mitigate the attack by making the parameters larger, but that comes at a significant efficiency loss.

```
import random

# Depends on parameters, e.g. for the DDH-based construction
# DHW suggest to use 5 times the logarithm of the group size
keyLength = 512 * 5

def simulateAttack(secretKey):
    """
    Simulate a side-channel attack on UPKE key updates.

    Returns:
        tuple: (numberOfUpdatesNeeded, attackSuccessful)
    """
    # Track frequency of leaked values for each key coefficient
    leakageFrequencies = [{} for _ in secretKey]
    updateCount = 0

    # Continue until we can distinguish both possible values for each coefficient
    while True:
        leakedValues = updateKeyWithLeakage(secretKey)
        updateCount += 1
        coefficientsWithTwoValues = 0

        # Record leaked values and count frequencies
        for i, leakedValue in enumerate(leakedValues):
            freqDict = leakageFrequencies[i]
            freqDict[leakedValue] = freqDict.get(leakedValue, 0) + 1

            # Check if we've seen both possible values (0 and 1) for this coefficient
            if len(freqDict) == 2:
                coefficientsWithTwoValues += 1

        # Attack succeeds when all coefficients have leaked both possible values
        if coefficientsWithTwoValues == keyLength:
            break

    # Reconstruct secret key by taking minimum leaked value for each coefficient
    reconstructedKey = [min(freqDict.keys()) for freqDict in leakageFrequencies]
    attackSuccessful = reconstructedKey == secretKey

    return updateCount, attackSuccessful

def updateKeyWithLeakage(secretKey):
    """
    Simulate UPKE key update with side-channel leakage.
```

```
46
47      Returns:
48          list: Leaked values (secret + noise) for each coefficient
49      """
50      # Generate random noise (0 or 1) for each coefficient
51      noise = [random.randrange(2) for _ in range(keyLength)]
52
53      # Return leaked values (secret key + noise)
54      return [secret + noiseBit for secret, noiseBit in zip(secretKey, noise
        )]
55
56  def main():
57      """Run the UPKE attack simulation."""
58      # Generate random binary secret key
59      secretKey = [random.randrange(2) for _ in range(keyLength)]
60
61      print(f"Secret key length: {keyLength}")
62      print(f"Running attack simulation...")
63      avgUpdatesNeeded = 0
64
65      # Simulate the attack
66      for _ in range(1000):
67          updatesNeeded, _success = simulateAttack(secretKey)
68          avgUpdatesNeeded += updatesNeeded
69
70      print(f"Updates needed:", avgUpdatesNeeded / 1000)
71
72  if __name__ == "__main__":
73      main()
```

B.3 Attacking MLS with Existing UPKE Schemes

We argue that our attacks on the UPKE constructions of [DKW21, APS23] can be embedded in realistic executions of the application, and that they result in breaking the application's stated security goals. For this we assume some familiarity with MLS; the most important multi-party UPKE application.

Intuitively, the authors of [ACDT20] propose a protocol, call it R-MLS, that replace MLS's use of PKE by UPKE. The goal is to improve Post-Compromise Forward-Secrecy (PCFS) in certain attack scenarios.

Better PCFS of R-MLS. Recall that, in (R-)MLS the group maintains a binary ratchet tree, where each member is assigned to a leaf and each node has a (U)PKE key pair. We write P_i for the group member at leaf i. In particular, party P_i knows only the secret keys on the path from leaf i to the root.

Consider the following part of an execution for a group G with $2N + 2$ members. Assume, up to this point there have been no corruptions (and, for simplicity, that $2N + 2$ is a power of 2).

1. Member P_1 is corrupted.
2. Member P_2 removes P_1.
3. Member P_1 sends a message msg to the group.
4. Member P_{2N+2} receives the above message and then is corrupted.

By PCFS, we expect that the adversary learns nothing about msg, the corrupted P_1 was removed before msg was sent and P_{2N+2} was corrupted after having receiving it. However, this is only true for R-MLS and not for MLS. [ACDT20]

To explain the reason, that to decrypt and recover msg the adversary either needs to learn the decryption key or recompute it. For R-MLS, neither should be possible, but for MLS the decryption key can be recomputed. Indeed, for both protocols, when removing P_1 in Step 2, P_2 encrypts a fresh secret s_{commit} to the (U)PKE public key pk_{right} of the right child of the ratchet tree root. It then computes an epoch secret s_{epoch} by hashing s_{commit} together with an init secret s_{init} known to all members in G. The epoch secret s_{epoch} (unknown to P_1) is used to derive the decryption key used to protect msg and as soon as P_{2N+2} uses the key to recover msg, it deletes it. In other words, neither corruption directly leaks the key itself.

However, with MLS the adversary can actually re-derive the decryption key. For that it needs s_{init} which it learned by corrupting P_1 in Step 1. The adversary also needs s_{commit}. For MLS, the corruption in Step 4 reveals the PKE secret key sk_{right} that P_{2N+1} used to decrypt s_{commit} in Step 2. Thus, the adversary can itself decrypt s_{commit}.

On the other hand, with R-MLS, the attack no longer works as the corruption in Step 4 only reveals an *updated* UPKE key sk'_{right} that can no longer be used to decrypt s_{commit}.

PCFS with forks. Both MLS and R-MLS are designed to preserve PCFS even in the presence of network partitions. Consider the above scenario, but now with temporarily fragmented network.

1. Member P_1 is corrupted.
2. Next, for each $i \in [2, N+1]$ the following happens in parallel:
   (a) Member P_i removes member P_1. The resulting protocol message (i.e. a commit) is received by member P_{N+1+i}.
   (b) Member P_i sends a message msg_i.
   (c) Member P_{N+1+i} receives msg_i and is then corrupted.

With R-MLS we would expect the same PCFS as before. After all, corruptions are either of a party that never learns the decryption key (P_1) or after messages were received (and so, after the decryption key was already deleted). However, our attack shows that this is not true with APS or DKW UPKE schemes.

Say $N = 50$—MLS was designed to realistically, support up to $N = 50,000$—and R-MLS is instantiated with APS. We note that all members $N + 1 + i$ for $i \in [2, N + 1]$ have leaves in the right subtree. Thus, in Step 4, the adversary learns N independently updated secret keys of the right child of the root, $sk^{(1)}_{right}, sk^{(2)}_{right}, \ldots sk^{(N)}_{right}$. Using our attack the adversary can recover the original key sk_{right} (with probability $1/2$). Thus, just like with MLS, it can decrypt all messages.

C Additional Preliminaries

C.1 PKE Definitions

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

Definition 10 (Correctness). *Let* PKE *be a PKE scheme and let* $pp \leftarrow \mathsf{Setup}$ *such that* pp *defines a public key space* $\mathcal{PK}$ *and message space* $\mathcal{M}$*. We say that* PKE *is* δ*-correct if for* $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$ *and for all* $m \in \mathcal{M}$*,*

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

Game PKE IND-CPA Security

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

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

Fig. 12: The IND-CPA security game for PKE $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ and adversary $\mathcal{A}$.

Definition 11 (γ-spreadness [HHK17]). *Let $\gamma > 0$. We say that a (key-homomorphic) PKE* PKE *is γ-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* Encrypt.

Definition 12 (PKE IND-CPA). *Let game* IND-CPA *for* PKE *be as defined in Fig. 12. The advantage of an adversary $\mathcal{A}$ in that game is defined as*

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

We further recall the definition of a fuzzy PKE, originally defined for KEMs [BCP+23] and a strengthened variant of correctness that we will need for the NIZKPoK.

Definition 13 (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*

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

Definition 14 (Correctness under Adversarially Chosen Randomness). *Let* PKE *be a PKE scheme and let $pp \leftarrow \mathsf{Setup}$ such that pp defines a public key space $\mathcal{PK}$, message space $\mathcal{M}$ and randomness space $\mathcal{RS}$. We say that* PKE *is δ-correct under adversarially chosen randomness if for all $m \in \mathcal{M}$ and $r \in \mathcal{RS}$*

$$\Pr\left[m \neq m' \;\middle|\; \begin{array}{l}(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(pp)\\ c \leftarrow \mathsf{PKE.Encrypt}(pp, pk, m; r)\\ m' \leftarrow \mathsf{PKE.Decrypt}(pp, sk, c)\end{array}\right] < \delta.$$

C.2 Proof Systems

We repeat the syntax for NIPS from the main body. 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.
Prove. $\pi \leftarrow \mathsf{P}^H(crs, x, w)$ on input a statement x and a witness w, outputs a proof π that $(x, w) \in \mathcal{R}$.
Verification. $0/1 \leftarrow \mathsf{V}^H(crs, x, \pi)$ verifies a proof π of statement x, i.e., that $(x, w) \in \mathcal{R}$ for some w.

Game Complete **for Non-Interactive Proof Systems**

$\underline{\mathsf{Exp}^{\mathsf{Complete}}_{\Pi}(\mathcal{A})}$

$H[\cdot] \leftarrow \bot$
$crs \leftarrow \mathsf{Setup}^{H}$
$(x, w) \leftarrow \mathcal{A}^{H}(crs)$
if $(x, w) \notin \mathcal{R}$ **then return** 0
$\pi \leftarrow \mathsf{P}^{H}(crs, x, w)$
return $1 - \mathsf{V}^{H}(crs, x, \pi)$

$\underline{\textbf{Oracle } H(\alpha)}$

if $H[\alpha] = \bot$ **then**
$H[\alpha] \leftarrow \$$
return $H[\alpha]$

Fig. 13: Completeness game for NIPS $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ for NP relation $\mathcal{R}$.

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*.[19]

Definition 15 (NIPS Completeness). *Let Π be a NIPS and $\mathsf{Exp}^{\mathsf{Complete}}_{\Pi}$ be as defined in Fig. 13. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against completeness of Π 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) δ-complete if $\mathsf{Adv}^{\mathsf{Complete}}_{\Pi}(\mathcal{A}) \leq \delta$.

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 π for statement x.[20]

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

Definition 16 (NIZK Adaptive Simulation-Soundness). *Let Π be a NIZK and $\mathsf{Exp}^{\mathsf{SS}}_{\Pi}$ be as defined in Fig. 14. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against adaptive simulation-soundness of Π is defined as*

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

Definition 17 (NIZK Zero-Knowledge). *Let Π be a NIZK and $\mathsf{Exp}^{\mathsf{ZK\text{-}NIZK}}_{\Pi}$ be as defined in Fig. 15. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against zero-knowledge of Π is defined as*

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

[19] 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* $\bot$ *otherwise*. An adaptive adversary can break completeness with any statement $(crs, *)$.

[20] 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.

Game SS for Adaptive Simulation-Soundness

$\underline{\mathsf{Exp}^{\mathsf{SS}}_{\Pi}(\mathcal{A})}$

$H[\cdot], \pi[\cdot] \leftarrow \bot$
$crs \leftarrow \mathsf{Setup}^{H}$
$(x, \pi) \leftarrow \mathcal{A}^{H,\mathtt{Simulate}}(crs)$
return $\mathsf{V}^{H}(crs, x, \pi) \wedge ((x, \cdot) \notin \mathcal{R})$
$\wedge\ (\pi[x] = \bot)$

Oracle $\mathtt{Simulate}(x)$

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

Oracle $H(\alpha)$

if $H[\alpha] = \bot$ **then**
$H[\alpha] \leftarrow \$$
return $H[\alpha]$

Oracle $\mathcal{P}(\alpha, \beta)$

req $H[\alpha] \in \{\bot, \beta\}$
$H[\alpha] \leftarrow \beta$

Fig. 14: The adaptive simulation-soundness security game for NIZK $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V}, \mathsf{S})$ for NP relation $\mathcal{R}$.

Game ZK-NIZK and ZK-PoK for Zero Knowledge

$\underline{\mathsf{Exp}^{\mathsf{ZK\text{-}NIZK}}_{\Pi}(\mathcal{A}), \mathsf{Exp}^{\mathsf{ZK\text{-}PoK}}_{\Pi}(\mathcal{A})}$

$b \leftarrow\!\!\$\ \{0, 1\}$
$H[\cdot], \pi[\cdot] \leftarrow \bot$
$crs \leftarrow \mathsf{Setup}^{H}$ // ZK-NIZK
$(crs, td) \leftarrow \mathsf{SetupAlt}^{H,\mathcal{P}}$ // ZK-PoK
$b' \leftarrow \mathcal{A}^{H,\mathtt{Prove}}(crs)$
return $(b = b')$

Oracle $\mathtt{Prove}(x, w)$

req $(x, w) \in \mathcal{R}$
if $\pi[x] = \bot$ **then**
if $b = 0$ **then**
$\pi[x] \leftarrow \mathsf{P}^{H}(crs, x, w)$
else
// ZK-NIZK
$\pi[x] \leftarrow \mathsf{S}^{H,\mathcal{P}}(crs, x)$
// ZK-PoK
$\pi[x] \leftarrow \mathsf{S}^{H,\mathcal{P}}(td, x)$
return $\pi[x]$

Oracle $H(\alpha)$

if $H[\alpha] = \bot$ **then**
$H[\alpha] \leftarrow \$$
return $H[\alpha]$

Oracle $\mathcal{P}(\alpha, \beta)$

req $H[\alpha] \in \{\bot, \beta\}$
$H[\alpha] \leftarrow \beta$

Fig. 15: The adaptive zero knowledge security game for NIZK and NIZKPoK for NP relation $\mathcal{R}$.

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, all 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 π 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 π for x.

Remark 2. 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.[21]

In addition to NIZK properties, a NIZKPoK should provide the following: setup-indistinguishability and straightline simulation-extractability. We note that the latter

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

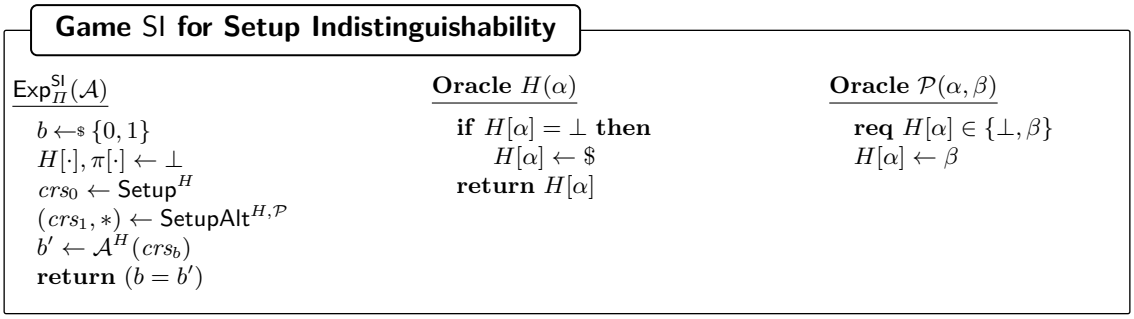


Fig. 16: The setup-indistinguishability game for NIZKPoK Π for NP relation $\mathcal{R}$ with setup Setup and trapdoor setup $\mathsf{SetupAlt}$.

implies soundness. We also modify the zero-knowledge definition by replacing Setup with $\mathsf{SetupAlt}$.

Definition 18 (NIZKPoK Setup Indistinguishability). *Let Π be a NIZKPoK and $\mathsf{Exp}^{\mathsf{SI}}_{\Pi}$ be as defined in Fig. 16. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against setup indistinguishability of Π 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 19 (NIZKPoK Zero-Knowledge). *Let Π be a NIZKPoK and $\mathsf{Exp}^{\mathsf{ZK\text{-}PoK}}_{\Pi}$ be as defined in Fig. 15. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against zero-knowledge of Π is defined as*

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

Definition 20 (NIZKPoK Straightline Simulation-Extractability). *Let Π be a NIZKPoK and $\mathsf{Exp}^{\mathsf{Ext}}_{\Pi}$ be as defined in Fig. 17. The advantage of a (possibly unbounded) adversary $\mathcal{A}$ against straightline simulation-extractability of Π is defined as*

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

D UKEM Correctness and Robustness

We first define correctness for a UKEM.

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

We say that UKEM is (ℓ, δ)-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.

Correctness 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

Game Ext for Straight-Line Simulation-Extractability

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

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

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

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

Fig. 17: 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.

Game Rob for Robustness

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

Oracle $\mathtt{Dec}(i, c, pk)$
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)

Oracle $\mathtt{Enc}(i)$
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. 18: Robustness game for $\mathsf{UKEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps}, \mathsf{Verify}_{\mathsf{mt}})$ and adversary $\mathcal{A}$.

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

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

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

D.1 Robustness of the LuKEM Scheme

Lemma 4. *Let* PKE *be key-homomorphic and* (ℓ, δ_1)*-correct and let* PoK *be* δ_2*-complete. Then* $\mathsf{LuKEM}[\mathsf{PKE}, \mathsf{PoK}]$ *is* (ℓ, δ')*-robust for* $\delta' \leq q_{\mathrm{enc}}(\delta_1 + \delta_2)$*, where* q_{enc} *is the number of* Enc *queries* $\mathcal{A}$ *makes.*

Proof. Let $\mathcal{A}$ be an adversary in the robustness game. Note that when 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 KeyGen (with the randomness sampled using the random oracle) which (c.f., the syntax) outputs keys in $\mathcal{SK}$. Since $\mathcal{A}$ can make ℓ 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. □

E Proof of Theorem 1

We prove Theorem 1 via Theorems 5 to 7. To this end, we define the two intermediate security notions IND-M-CPA$^+$ and IND-M-CCA$^-$ (Appendix E.1). First, we show that if PKE is Shifty-IND-CPA secure, then LuKEM$_{\text{no-pv}}$ is IND-M-CPA$^+$ secure (Appendix E.2). Second, we show that if LuKEM$_{\text{no-pv}}$ is IND-M-CPA$^+$ secure and PKE is correct and γ-spread, then LuKEM$_{\text{no-pv}}$ is also IND-M-CCA$^-$ secure (Appendix E.3). Finally, we show that if LuKEM$_{\text{no-pv}}$ is IND-M-CCA$^-$ secure and PoK is straightline simulation extractable (along with the other PoK properties), then LuKEM is IND-M-CCA secure (Appendix E.4).

E.1 Intermediate Security Notions

Consider an IND-M-CPA notion which differs from IND-M-CCA in that there is no Dec oracle. Our first intermediate notion, IND-M-CPA$^+$, strengthens IND-M-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-M-CPA$^+$ experiment are full, unless a correctness error occurs.

Our second intermediate notion, IND-M-CCA$^-$, strengthens IND-M-CPA$^+$ by adding a Dec$^-$ oracle that differs from Dec in that it does not take 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 3. We note that IND-M-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.

Our IND-M-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 Encapsrandomness 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-M-CPA$^+$ to IND-M-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] Encapsencrypts a random and independent $\overline{sk}$ using a PKE.

Definition 22 (IND-M-CPA$^+$ and IND-M-CCA$^-$ Security). *Let* ATK $\in \{\text{CPA}^+, \text{CCA}^-\}$. *Let* $\text{Exp}^{\text{IND-M-ATK}}_{\text{UKEM}}(\mathcal{A})$ *be as defined in Fig. 19. The advantage of an adversary* $\mathcal{A}$ *against*

The Upd and $\mathtt{Dec}^-$ Oracles for Intermediate Security Notions

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

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

Oracle $\mathtt{Dec}^-(i', c', pk')$

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-M-CPA$^+$ is the same as IND-M-CCA in Fig. 3 except that in IND-M-CPA$^+$ the Dec oracle is replaced by Upd. IND-M-CCA$^-$ is the same as IND-M-CPA$^+$ and additionally provides the $\mathtt{Dec}^-$ oracle.

the IND-M-ATK *security of a key-homomorphic* UKEM *scheme is defined as*

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

E.2 From Shifty-IND-CPA to IND-M-CPA$^+$

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

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

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

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

For a proof intuition, we refer to Section 5.

Proof of Theorem 5. 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 IND-M-CPA$^+$ experiment for $\mathsf{LuKEM}_{\text{no-pv}}$ and adversary $\mathcal{A}$. We have

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

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 $\mathsf{LuKEM}_{\text{no-pv}}$ and hence by the (k, δ)-correctness of PKE. We have

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

The "break" event. Let Brk be the event from the proof outline in Section 5. 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 $\texttt{MChal}$. 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 $\texttt{Enc}$ query (and not an $\texttt{Upd}$ query),
(3) there exists a j^* s.t. $j^* = i^*$ or j^* is reachable from i^* via only $\texttt{Dec}$ edges (these are created via $\texttt{Upd}$ 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 $\texttt{Enc}(j^*)$ or $\texttt{MChal}(j^*)$.

We will show that

$$2\Pr\left[\mathbf{Exp}_1(\mathcal{A})\right] - 1 \leq \Pr[\exists i^* : \mathsf{Brk}(i^*)]. \tag{5}$$

We then define the following event Brk^*: an index i^* is chosen (by the reduction) at random independently of $\mathcal{A}$'s view. 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.

$$\Pr[\mathsf{Brk}] \leq (q_{\text{enc}} + q_{\text{upd}}) \cdot \Pr[\mathsf{Brk}^*]. \tag{6}$$

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

$$\Pr[\mathsf{Brk}^*] \leq \mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{B}) + q_{\text{ro}} \cdot (q_{\text{enc}} + 1) \cdot 2^{-\kappa}. \tag{7}$$

Combining Eqs. (5) to (7), we get the bound from the theorem.

Proof of Eq. (5). The reason Eq. (5) holds is that if $\mathcal{A}$ does not trigger 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. (7)). 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 Brk^* occurs or until the guess becomes incorrect, in which case it outputs a random bit. The overall proof strategy follows the one of the FO transform [HHK17].

$\mathcal{B}$ 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 $\texttt{Enc}$ queries from $\mathcal{A}$ honestly by running Encaps until i^* is created. Assume the node i^* is created via an $\texttt{Enc}(p^*)$ 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 $pk_{i^*} := pk_{p^*} + pk^*$. For the ciphertext returned by $\texttt{Enc}(p^*)$, $\mathcal{B}$ encrypts a random message m. It also samples the key K returned by $\texttt{Enc}(p^*)$ at random. Observe that this simulation is identical to the $\mathsf{IND\text{-}M\text{-}CPA}^+$ 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 Brk occurs for a node closer to the root than i^*, so Brk^* can no longer happen and the experiment stops by outputting a random bit.

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 $\mathsf{IND\text{-}M\text{-}CPA}^+$ 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 $\texttt{Dec}$ 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 by outputting a random bit).

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 $j^* \in \mathsf{dec\text{-}closure}(\{i^*\})$. $\mathtt{Rev}(j^*)$ is not allowed and $\mathrm{Upd}(j^*)$ is answered honestly (recording $\overline{sk}$ provided by $\mathcal{A}$). The remaining queries are $\mathtt{Enc}(j^*)$ and $\mathtt{MChal}(j^*)$. For such a query creating a node $j' \notin \mathsf{dec\text{-}closure}(\{i^*\})$, $\mathcal{B}$ queries its $\mathtt{Chal}$ oracle.

If $j^* = i^*$ then $\mathcal{B}$ queries $\mathtt{Chal}$ 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 $\mathtt{Rev}$ queries of $\mathcal{A}$.

$\mathcal{B}$ proceeds similarly when there are $\mathtt{Upd}$ edges in between i^* and j^*. Since $\mathcal{A}$ provides $\overline{sk}$ whenever it queries the $\mathtt{Upd}$ oracle, $\mathcal{B}$ can query its $\mathtt{Chal}$ 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 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 Brk then $\mathcal{B}$ wins except with probability $q_{\mathrm{ro}} \cdot (q_{\mathrm{enc}} + 1) \cdot 2^{-\kappa}$.

Combining the above we get

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{Shifty\text{-}IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{B}) + q_{\mathrm{ro}} \cdot (q_{\mathrm{enc}} + 1) \cdot 2^{-\kappa} \;&\geq\; 2\Pr\left[b = b'\right] - 1. \\
&=\; 2\left(\Pr[\mathsf{Brk}^*] + \frac{1}{2} \cdot \Pr[\neg\mathsf{Brk}^*]\right) - 1 \\
&=\; 2\left(\Pr[\mathsf{Brk}^*] + \frac{1}{2} \cdot (1 - \Pr[\mathsf{Brk}^*])\right) - 1 \\
&=\; 2\left(\Pr[\mathsf{Brk}^*] + \frac{1}{2} \cdot (1 - \Pr[\mathsf{Brk}^*])\right) - 1 \\
&=\; \Pr[\mathsf{Brk}^*]\,,
\end{aligned}$$

which is exactly Eq. (7). □

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

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

Theorem 6. *If* $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ *is* $\mathsf{IND\text{-}M\text{-}CPA}^+$ *secure and* PKE *is* δ*-correct and* γ*-spread, then* $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ *is* $\mathsf{IND\text{-}M\text{-}CCA}^-$ *secure in the* non-programmable *ROM. More specifically, for any adversary* $\mathcal{A}$ *against* $\mathsf{IND\text{-}M\text{-}CCA}^-$ *security of* $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ *that issues* q_{ro} *random*

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

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

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

Proof Intuition. We extend the strategy from Theorem 5 to deal with $\mathcal{A}$'s $\mathtt{Dec}^{-}$ 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 $\mathsf{IND\text{-}M\text{-}CPA}^{+}$ game, meaning it has only access to an $\mathtt{Upd}$ 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 $\mathtt{Dec}^{-}$ queries in more detail. Since $\mathcal{B}$ is playing in the $\mathsf{IND\text{-}M\text{-}CPA}^{+}$ game for $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$, $\mathcal{B}$'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 $\mathtt{Dec}^{-}$ 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 $\mathtt{Dec}^{-}$ 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, K, r 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 $\mathtt{Dec}^{-}$ queries without searching through all random oracle queries. More specifically, if $\mathcal{A}$ queries $\mathtt{Dec}^{-}$ 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 $\mathtt{Upd}$ 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.

Proof of Theorem 6. We define a sequence of games to prove the theorem, using the strategy described above. In the following, we denote by $\Pr[\mathbf{Exp}_i(\mathcal{A})]$ the probability of game i returning 1.

Game 0. This is the $\mathsf{IND\text{-}M\text{-}CCA}^{-}$ experiment for $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}[\mathsf{PKE}]$ and adversary $\mathcal{A}$. We have

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

Game 1. Game 1 differs from Game 0 in that the challenger evaluates the $\mathtt{Dec}^{-}$ oracle without the secret key. This is done via observing random oracle queries and keeping them consistent with decryption queries. For this, the challenger uses the additional list $\mathcal{L}_D$ that is also described in the above intuition whose entries correspond to encryptions that $\mathcal{A}$ (may have) computed by querying the random oracles. More specifically, whenever $\mathcal{A}$ makes a random oracle query on a new input (pk, m), the challenger creates an entry in $\mathcal{L}_D$ which has the form $(pk, pk', \overline{sk}, m, c, K)$ such that $c = \mathsf{Encrypt}(pp_{\mathsf{PKE}}, M; r)$ and

$pk' = pk + \overline{pk}$, where $(\overline{pk}, \overline{sk}) = \mathsf{KeyGen}(d)$. Then, when $\mathcal{A}$ queries the $\mathtt{Dec}^-$ oracle on (i', c', pk'), the challenger 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 yes, then this is a valid ciphertext. The challenger creates a new node, computing the secret key as $sk_{i'} + \overline{sk}$ and returns K'. If there is no such entry, the challenger simply outputs $\perp$. Further, $\mathcal{A}$'s queries to the $\mathtt{Upd}$ oracle are handled just as described in Fig. 19.

To bound the difference between Game 0 and Game 1, we rely on robustness and spreadness. In particular, Games 0 and 1 are identical unless one of the following bad events occurs:

1. $\mathcal{A}$ *finds a correctness error.* Say $\mathcal{A}$ queries the RO on (pk_i, m') such that $\mathsf{Decrypt}$ fails on $c' = \mathsf{Encrypt}(pk_i, m'; H_2(pk', m'))$, and then queries $\mathtt{Dec}^-$ on (i, c', pk'). In Game 0, where the challenger runs $\mathsf{Decrypt}$, $\mathtt{Dec}^-$ outputs $\perp$, but in Game 1, where the challenger uses $\mathcal{L}_D$, it outputs $K' = H_1(pk_i, m')$. We denote this event by $\mathsf{BrkCorr}$.
2. $\mathcal{A}$ *encrypts without querying RO.* Say $\mathcal{A}$ queries $\mathtt{Dec}^-$ on (i, c', pk') such that $m' = \mathsf{Decrypt}(sk_i, c')$ is not $\perp$ and $c' = \mathsf{Encrypt}(pk_i, M'; H_2(pk', m'))$ but $\mathcal{A}$ never queried the RO on (pk', m'). In Game 0, where the re-encryption check succeeds, $\mathtt{Dec}^-$ outputs $K' = H_1(pk_i, m')$. But in Game 1 the challenger does not find a matching record in $\mathcal{L}_D$, so $\mathtt{Dec}^-$ outputs $\perp$. We denote this event by $\mathsf{BrkSprd}$.

Therefore,

$$|[\mathbf{Exp}_0(\mathcal{A})] - [\mathbf{Exp}_1(\mathcal{A})]| \leq \Pr[\mathsf{BrkCorr}] + \Pr[\mathsf{BrkSprd}] \ .$$

We will upper bound the two events as

$$\Pr[\mathsf{BrkCorr}] \leq q_{\mathrm{ro}} \cdot \delta \tag{8}$$

and

$$\Pr[\mathsf{BrkSprd}] \leq q_{\mathrm{dec}} \cdot 2^{-\gamma} \ . \tag{9}$$

To justify Eq. (9), recall that γ-spreadness means $\max_{c \in \{0,1\}^*} \Pr[\mathsf{Encrypt}(\mathsf{pk}(sk), m) = c] \leq 2^{-\gamma}$ for all m and $sk \in \overline{\mathcal{SK}}$ (i.e., sk after *any* number of updates). Therefore, when $\mathcal{A}$ has not queried the RO, a ciphertext that is not rejected in Game 0 will only be rejected in Game 1 with probability $2^{-\gamma}$. Union bound over all q_{dec} decryption queries yields Eq. (9).

To prove Eq. (8), observe that H_2 outputs independent random values and therefore each message queried to H_2 triggers a correctness error with probability at most δ. By the union bound over the number of random oracle queries q_{ro}, this event happens with probability at most $q_{\mathrm{ro}} \cdot \delta$.

Final reduction. We now construct a reduction $\mathcal{B}$ against $\mathsf{IND\text{-}M\text{-}CPA}^+$ that simulates Game 1 for $\mathcal{A}$. $\mathcal{B}$ forwards $\mathcal{A}$'s Enc, $\mathtt{MChal}$, Rev, $\mathtt{Upd}$ and random oracle queries to its own oracles. When $\mathcal{A}$ calls $\mathtt{Dec}^-$, $\mathcal{B}$ simulates without the secret keys as in Game 1. Note however, that $\mathcal{B}$ can extract the shifts $\overline{sk}$ as described in Game 1 which are stored in $\mathcal{L}_D$ and if decryption succeeds, $\mathcal{B}$ will also query $\mathtt{Upd}$ on that node with shift $\overline{sk}$. Due to the previous game hop, this simulation is perfect.

When $\mathcal{A}$ stops and outputs a bit b', $\mathcal{B}$ outputs the same bit to its own challenger. Note that if $\mathcal{A}$ is a valid adversary for game $\mathsf{IND\text{-}M\text{-}CCA}^-$ (i.e., it does not reveal any nodes on the challenge path) and outputs the correct bit, then $\mathcal{B}$ is also a valid for game $\mathsf{IND\text{-}M\text{-}CPA}^+$ and also outputs the correct bit. Hence,

$$2 \Pr\left[\mathbf{Exp}_1(\mathcal{A})\right] - 1 \leq \mathsf{Adv}^{\mathsf{IND\text{-}M\text{-}CPA}^+}_{\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}[\mathsf{PKE}]}(\mathcal{B}) \ ,$$

which concludes the proof. □

E.4 From IND-M-CCA$^-$ to IND-M-CCA using PoK

In the final step, we consider LuKEM shown in Fig. 6. It differs from $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$ 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-M-CCA secure if the underlying key-homomorphic UKEM is IND-M-CCA$^-$ secure and PoK is straightline simulation-extractable. For our final scheme, we instantiate UKEM with $\mathsf{LuKEM}_{\mathsf{no\text{-}pv}}$.

Theorem 7. *If* UKEM *is key-homomorphic and* IND-M-CCA$^-$ *secure and* PoK *is straightline simulation extractable, then* LuKEM *is* IND-M-CCA *secure. More specifically, for any adversary $\mathcal{A}$ against* IND-M-CCA *security of* LuKEM*, there exist adversaries $\mathcal{B}$, $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ against the* IND-M-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*

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

Proof Intuition. Let $\mathcal{A}$ be an adversary against the IND-M-CCA security of LuKEM. We construct a reduction $\mathcal{B}$ against IND-M-CCA$^-$ security of UKEM. In general, there will be a one-to-one correspondence between nodes in the IND-M-CCA experiment and nodes in the IND-M-CCA$^-$ experiment. Note that all nodes in the IND-M-CCA$^-$ game are full (up to robustness), so half-nodes in the IND-M-CCA experiment will correspond to full nodes in the IND-M-CCA$^-$ game. The reduction $\mathcal{B}$ keeps track itself of which nodes are half-nodes in the IND-M-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 $\texttt{Enc}(i)$ or $\texttt{MChal}(i)$, $\mathcal{B}$ also instructs the IND-M-CCA$^-$ challenger to create an Enc-node j by calling $\texttt{Enc}(i)$ or $\texttt{MChal}(i)$. 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 π 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 π using PoK.S. Finally, $\mathcal{B}$ marks the new node j as full if node i was full.

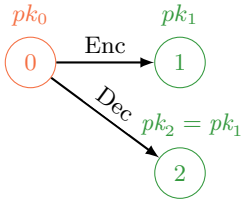
When $\mathcal{A}$ instructs its emulated challenger to create a Dec-node j by calling $\texttt{Dec}(i', c', pk', mt')$, $\mathcal{B}$ first checks whether mt' is a valid member-tag. Then it queries its restricted $\texttt{Dec}^-$ oracle to check whether c' successfully decrypts in which case it gets 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 $\mathcal{A}$.[22]

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

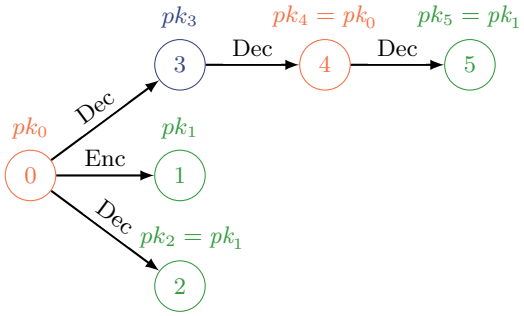
Extracting from adversarial proofs. So far, we assumed that whenever $\mathcal{A}$ calls Dec and decapsulation fails, $\mathcal{B}$ can extract $\overline{sk}$ from the PoK mt' sent by $\mathcal{A}$ to the Dec oracle. However, extraction does not work for statements (pk_p, pk_i) for which $\mathcal{B}$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

For example, say $\mathcal{A}$ queries $\texttt{Enc}(p)$, 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),

[22] 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.



(a) $\mathcal{A}$ queries $\texttt{Enc}(0)$, which creates node 1, and then queries $\texttt{Dec}(1, \bot, pk_1, mt_1)$ 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 $\texttt{Dec}(3, \bot, pk_0, mt)$. 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-M-CCA$^-$ game.

$\mathcal{B}$ 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-M-CCA$^-$ experiment corresponds to multiple nodes in the IND-M-CCA experiment: one Enc-node with public key pk_i and a set $\mathsf{copied}(i)$ 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-M-CCA experiment) and pk_i, and $\texttt{Dec}^-$ returns $\bot$ (and $\mathcal{A}$'s member tag is valid), then $\mathcal{B}$ simply creates a new half-node with pk_i and adds it to $\mathsf{copied}(i)$. If $\mathcal{A}$ queries `Enc` (or `MChal`) with a node in $\mathsf{copied}(i)$, $\mathcal{B}$ 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 $\mathsf{copied}(i)$ and some pk_j and mt. Fortunately, this only creates a half-node in IND-M-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 $\mathsf{copied}(i)$, 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-M-CCA$^-$. Otherwise, $\mathcal{B}$ creates a new half-node in $\mathsf{copied}(j)$.

Trivial wins with copied nodes. Assume $\mathcal{A}$ queries $\texttt{MChal}(i_c)$ for a node i_c such that $\mathcal{B}$, mapping i_c to the respective node i^* in its own game, queries $\texttt{MChal}(i^*)$. 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 $\mathsf{copied}(i^*)$ 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 $\texttt{Dec}^-$ 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-M-CCA, then $\mathcal{B}$'s corruptions do not violate the trivial-wins either.

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

Proof of Theorem 7. 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 $\mathsf{LuKEM}[\mathsf{UKEM}, \mathsf{PoK}]$, where UKEM is a key-homomorphic UKEM and the PoK statement is defined in Eq. (1). Hence, for an adversary $\mathcal{A}$ in that game we have

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{LuKEM}}(\mathcal{A}) := 2\Pr\left[\mathbf{Exp}_0(\mathcal{A})\right] - 1 \ . \tag{10}$$

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

$$|[\mathbf{Exp}_0(\mathcal{A})] - [\mathbf{Exp}_1(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{SI}}_{\mathsf{PoK}}(\mathcal{B}_1) \ . \tag{11}$$

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

More precisely, for each $\texttt{Enc}$ and the $\texttt{MChal}$ 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

$$|[\mathbf{Exp}_1(\mathcal{A})] - [\mathbf{Exp}_2(\mathcal{A})]| \leq \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{PoK}}(\mathcal{B}_2) \ . \tag{12}$$

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 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 $\texttt{Dec}$ 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 $\texttt{Dec}$ 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 $\mathsf{IND\text{-}M\text{-}CCA}^-$ 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'.[23] 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 $\mathsf{copied}(i)$.

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. 17). 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 $\pi[pk_{i'}, pk'] = \perp$ 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

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

Adversary $\mathcal{B}$ simulates Game 3 as follows. It gets as input pk_0 and has access to oracles Enc, Rev, `MChal`, `Upd` as well as the $\mathtt{Dec}^-$ 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 the set copied to map indices from the $\mathsf{IND\text{-}M\text{-}CCA}$ game to the $\mathsf{IND\text{-}M\text{-}CCA}^-$ 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 $\mathtt{Dec}^-$ 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 $\mathtt{Dec}^-$. If successful, $\mathcal{B}$'s challenger created a node and

[23] 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.

$\mathcal{B}$ 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 $\perp$.

For all other $\mathtt{Dec}$ queries, $\mathcal{B}$ first queries $\mathtt{Dec}^{-}$. 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 $\perp$.

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 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 7. □

F Proof of Lemma 2

To prove Lemma 2, we first consider the case where we shift the center from $\mathbf{c}_0$ to $\mathbf{0}$. We expand on the proof of Lemma 1, given as [BF11, Lemma 4.12].

Lemma 5. *Let $\mathbf{c}_0 \in \mathbb{Z}^n$, let $X \sim \mathcal{D}_{\mathbb{Z}^n,\sigma_0,\mathbf{0}}$ and, conditioned on $X = \mathbf{x}$, let $X^+ \sim \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{x}}$. Furthermore, let $Y^+ \sim \mathcal{D}_{\mathbb{Z}^n,\bar{\sigma},\mathbf{0}}$, with $\bar{\sigma} := \sqrt{\sigma_0^2+\sigma^2}$, and, conditioned on $Y^+ = \mathbf{z}$, let $Y \sim \mathcal{D}_{\mathbb{Z}^n,\sigma',c'(\mathbf{z})}$ with $\sigma' := \sigma_0\sigma/\bar{\sigma}$ and $c'(\mathbf{z}) := \sigma_0^2/\bar{\sigma}^2 \cdot \mathbf{z}$.*

If $\sigma' > \sqrt{(\ln(2n(1+1/\varepsilon)))/\pi}$, then the statistical distance between the joint distributions (X, X^+) and (Y, Y^+) is bounded by $\frac{4\varepsilon}{(1-\varepsilon)^2}$.

Proof. For all $\mathbf{x}, \mathbf{z} \in \mathbb{Z}^n$, with $w(\mathbf{x}) := \rho_{\sigma_0,\mathbf{0}}(\mathbb{Z}^n) \cdot \rho_{\sigma,\mathbf{x}}(\mathbb{Z}^n)$ and $w'(\mathbf{z}) := \rho_{\sigma',c'(\mathbf{z})}(\mathbb{Z}^n) \cdot \rho_{\bar{\sigma},\mathbf{0}}(\mathbb{Z}^n)$, we have

$$
\begin{aligned}
\Pr[Y = \mathbf{x} \wedge Y^+ = \mathbf{z}] &= \Pr[Y = \mathbf{x} \,|\, Y^+ = \mathbf{z}] \Pr[Y^+ = \mathbf{z}] \\
&= \rho_{\sigma',c'(\mathbf{z})}(\mathbf{x}) \cdot \rho_{\bar{\sigma},\mathbf{0}}(\mathbf{z}) / w'(\mathbf{z}) \\
&= \exp\Big(-\pi\Big(\underbrace{\|\mathbf{x} - \tfrac{\sigma_0^2}{\bar{\sigma}^2}\mathbf{z}\|^2}_{= \|\mathbf{x}\|^2 - 2\frac{\sigma_0^2}{\bar{\sigma}^2}\langle \mathbf{x}, \mathbf{z}\rangle + \frac{\sigma_0^4}{\bar{\sigma}^4}\|\mathbf{z}\|^2} \tfrac{\bar{\sigma}^2}{\sigma_0^2\sigma^2} + \|\mathbf{z}\|^2 \cdot \tfrac{1}{\bar{\sigma}^2}\Big)\Big) \Big/ w'(\mathbf{z}) \\
&= \exp\Big(-\tfrac{\pi}{\sigma_0^2\sigma^2}\Big(\bar{\sigma}^2\|\mathbf{x}\|^2 - 2\sigma_0^2\langle \mathbf{x}, \mathbf{z}\rangle + \tfrac{\sigma_0^4}{\bar{\sigma}^2}\|\mathbf{z}\|^2 + \tfrac{\sigma_0^2\sigma^2}{\bar{\sigma}^2}\|\mathbf{z}\|^2\Big)\Big) \Big/ w'(\mathbf{z}) \\
&= \exp\Big(-\tfrac{\pi}{\sigma_0^2\sigma^2}\Big((\sigma_0^2+\sigma^2)\|\mathbf{x}\|^2 - 2\sigma_0^2\langle \mathbf{x}, \mathbf{z}\rangle + \sigma_0^2\|\mathbf{z}\|^2\Big)\Big) \Big/ w'(\mathbf{z}) \\
&= \exp\Big(-\tfrac{\pi}{\sigma_0^2\sigma^2}\Big(\sigma^2\|\mathbf{x}\|^2 + \sigma_0^2\|\mathbf{z} - \mathbf{x}\|^2\Big)\Big) \Big/ w'(\mathbf{z}) \\
&= \rho_{\sigma_0,\mathbf{0}}(\mathbf{x}) \cdot \rho_{\sigma,\mathbf{x}}(\mathbf{z}) / w'(\mathbf{z}) \\
&= \Pr[X = \mathbf{x}] \Pr[X^+ = \mathbf{z} | X = \mathbf{x}] \cdot \tfrac{w(\mathbf{x})}{w'(\mathbf{z})} = \Pr[X = \mathbf{x} \wedge X^+ = \mathbf{z}] \cdot \tfrac{w(\mathbf{x})}{w'(\mathbf{z})}.
\end{aligned}
$$

We now use the fact [BF11, Lemma A.2] that for discrete random variables U, V, if for some $c \in \mathbb{R}$ and all $a \in \mathbb{Z}^m$: $\Pr[V = a] = c(1 - \delta(a)) \Pr[U = a]$ for $\delta(a) \in [0, \omega]$, then the statistical distance between U and V is at most $\omega/(1-\omega)$.

To apply this to the random variables (X, X^+) and (Y, Y^+), we thus need to show that

$$
\frac{w'(\mathbf{z})}{w(\mathbf{x})} = c(1 - \delta(\mathbf{x}, \mathbf{z})) \quad \text{with } \delta(\mathbf{x}, \mathbf{z}) \in [0, \omega] \tag{13}
$$

for some c and ω.

It was shown [GPV08] that for $\mathbf{c} \in \mathbb{Z}^n$ and $\tau > \sqrt{(\ln(2n(1+1/\varepsilon)))/\pi}$:

$$\frac{1-\varepsilon}{1+\varepsilon}\rho_{\tau,\mathbf{0}}(\mathbb{Z}^n) \leq \rho_{\tau,\mathbf{c}}(\mathbb{Z}^n) \leq \rho_{\tau,\mathbf{0}}(\mathbb{Z}^n) \ . \tag{14}$$

Since $\sigma \geq \sigma_0\sigma/\sqrt{\sigma_0^2+\sigma^2} = \sigma' > \sqrt{(\ln(2n(1+1/\varepsilon)))/\pi}$ by assumption, we have $\frac{1-\varepsilon}{1+\varepsilon} \cdot \frac{w'(\mathbf{0})}{w(\mathbf{0})} \leq \frac{w'(\mathbf{z})}{w(\mathbf{x})} \leq \frac{1+\varepsilon}{1-\varepsilon} \cdot \frac{w'(\mathbf{0})}{w(\mathbf{0})}$, which yields (13) for $c := \frac{1+\varepsilon}{1-\varepsilon} \cdot \frac{w'(\mathbf{0})}{w(\mathbf{0})}$ and $\omega := \frac{4\varepsilon}{(1+\varepsilon)^2}$. By [BF11, Lemma A.2], the statistical distance between (X, X^+) and (Y, Y^+) is thus at most $\omega/(1-\omega) = 4\varepsilon/(1-\varepsilon)^2$. □

Lemma 2 now follows by the following argument. Let us first rewrite X^+ as $X + \bar{X}$ with $\bar{X} \sim \mathcal{D}_{\mathbb{Z}^n,\sigma}$ and Y as $\sigma_0^2/\bar{\sigma}^2 Y^+ + \bar{Y}$ with $\bar{Y} \sim \mathcal{D}_{\mathbb{Z}^n,\sigma'}$. We now replace all distributions Z by $Z_1 := Z + \mathbf{c}_0$. We thus have $X_1 = X + \mathbf{c}_0$ and $X_1^+ = X_1 + \bar{X}$ as well as $Y_1^+ = Y^+ + \mathbf{c}_0$ and $Y_1 = \sigma_0^2/\bar{\sigma}^2(Y_1^+ - \mathbf{c}_0) + \bar{Y} + \mathbf{c}_0$. This shows Lemma 2.

G Proof of Theorem 2

We start with proving that our resampling strategy of the secret key and revealed key updates is statistically close to the original distribution.

Corollary 3. *Consider the following two distributions defined by $\sigma > 0$:*

- *sample* $\mathbf{r}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{0}}$
- *for* $i = 1, \ldots, k$, *sample* $\mathbf{r}_i^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{r}^*}$ *independently*

- *sample* $\mathbf{s}_1^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sqrt{2}\sigma,\mathbf{0}}$
- *for* $i = 2, \ldots k$, *sample* $\mathbf{s}_i^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_i,\mathbf{c}_i}$ *with* $\sigma_i := \sqrt{\frac{i+1}{i}}\sigma$ *and* $\mathbf{c}_i := \frac{1}{i}\sum_{\ell=1}^{i-1} \mathbf{s}_\ell^+$.
- *sample* $\mathbf{s}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_k^*,\mathbf{c}_{k+1}}$ *with* $\sigma_k^* = \frac{1}{\sqrt{k+1}}\sigma$.

Then we have that if $\frac{1}{\sqrt{k+1}}\sigma > \sqrt{(\ln(2n(1+1/\varepsilon)))/\pi}$, *the statistical distance between* $(\mathbf{r}^*, \mathbf{r}_1^+, \ldots, \mathbf{r}_k^+)$ *and* $(\mathbf{s}^*, \mathbf{s}_1^+, \ldots, \mathbf{s}_k^+)$ *is bounded by* $k \cdot \frac{4\varepsilon}{(1-\varepsilon)^2}$.

Proof. The statement is shown by defining a sequence of hybrid games where in H_0 we sample as in the first sequence $(\mathbf{r}^*, \ldots)$ and in H_k we sample as in the second sequence $(\mathbf{s}^*, \ldots)$. In particular, the sampling in H_j, for $0 \leq j \leq k$, is defined as follows:

- for $i = 1, \ldots, j$: $\mathbf{s}_i^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_i,\mathbf{c}_i}$
- $\mathbf{s}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_j^*,\mathbf{c}_{j+1}}$
- for $i = j+1, \ldots, k$: $\mathbf{r}_i^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{s}^*}$

This defines the values $(\mathbf{s}_1, \ldots, \mathbf{s}_j, \mathbf{r}_{j+1}, \ldots \mathbf{r}_k, \mathbf{s}^*)$. The only difference in sampling between games H_{j-1} and H_j is that in the former we sample

$$\mathbf{r}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_{j-1}^*,\mathbf{c}_j} \text{ and then}$$
$$\mathbf{r}_j^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{r}^*}$$

whereas in the latter we sample

$$\mathbf{s}_j^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_j,\mathbf{c}_j} \text{ and then}$$
$$\mathbf{s}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_j^*,\mathbf{c}_{j+1}}$$

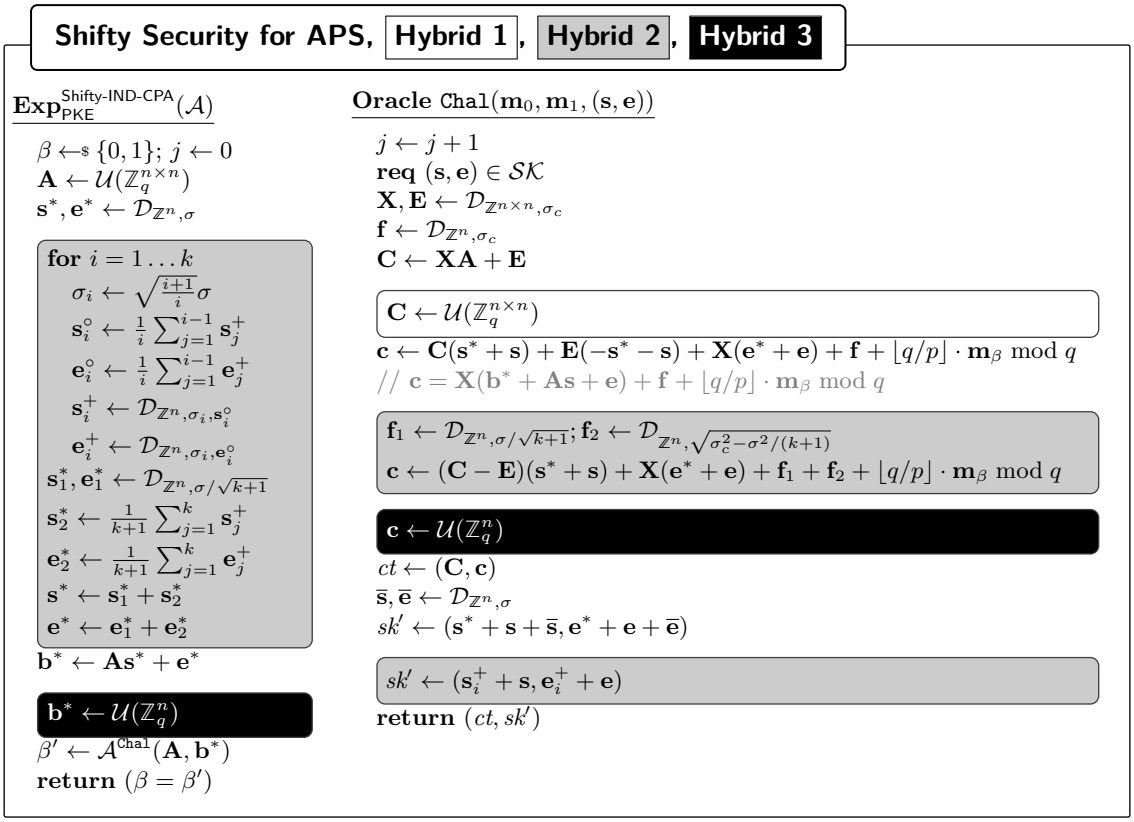


Fig. 21: The Shifty-IND-CPA security game for homomorphic PKE from [APS23] and hybrid games in the proof of Theorem 2: Hybrid 1 only includes the white box, Hybrid 2 additionally the gray boxes and Hybrid 3 contains all boxes.

Since $\sigma_j^* > \sqrt{(\ln(2n(1+1/\varepsilon)))/\pi}$, Lemma 2 yields that $(\mathbf{r}^*, \mathbf{r}_j^+)$ is $(\frac{4\varepsilon}{(1-\varepsilon)^2})$-close to $(\mathbf{t}^*, \mathbf{t}^+)$ where

$$\mathbf{t}^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\bar{\sigma},\mathbf{c}_j}, \text{ with } \bar{\sigma} := \sqrt{(\sigma_{j-1}^*)^2 + \sigma^2} = \sqrt{\tfrac{j+1}{j}}\sigma = \sigma_j \text{ and}$$
$$\mathbf{t}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma',\mathbf{c}'} \text{ with } \sigma' := \sigma_{j-1}^*\sigma/\bar{\sigma} \text{ and } \mathbf{c}' := \mathbf{c}_j + (\sigma_{j-1}^*)^2/\bar{\sigma}^2(\mathbf{t}^+ - \mathbf{c}_j)\ .$$

Since $\sigma' = (\frac{1}{\sqrt{j}}\sigma)\cdot\sigma/\sqrt{\frac{j+1}{j}}\sigma = \frac{1}{\sqrt{j+1}}\sigma = \sigma_j^*$ and $\mathbf{c}' = \mathbf{c}_j + \frac{1}{j+1}(\mathbf{t}^+ - \mathbf{c}_j) = \frac{1}{j+1}\mathbf{t}^+ + \frac{j}{j+1}\mathbf{c}_j = \frac{1}{j+1}\mathbf{t}^+ + \frac{j}{j+1}(\frac{1}{j}\sum_{\ell=1}^{j-1}\mathbf{s}_\ell^+) = \mathbf{c}_{j+1}$. We have thus showed that $(\mathbf{s}_j^+, \mathbf{s}^*)$ is $(\frac{4\varepsilon}{(1-\varepsilon)^2})$-close to $(\mathbf{t}^+, \mathbf{t}^*)$. Since there are k game hops, the overall statistical distance is $k \cdot \frac{4\varepsilon}{(1-\varepsilon)^2}$, which shows the lemma. □

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 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 $\texttt{Chal}$ queries), from G_0, which is Shifty-IND-CPA to

G_k, which is Hybrid 1. In 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 G_{i-1} to G_i replaces, in the i-th query, the component $\mathbf{C} = \mathbf{XA} + \mathbf{E}$ 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 $\mathbf{Xz}_0 + \mathbf{Ez}_1 + \mathbf{f}$ 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 G_{i-1} or G_i to adversary $\mathcal{A}$. W.l.o.g. we assume that $\mathcal{A}$ makes exactly k queries and uses correct values $(\mathbf{s}, \mathbf{e})$.

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

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

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

// $\mathbf{C} = \mathbf{XA} + \mathbf{E}$ or $\mathbf{C} \leftarrow \mathbb{Z}_q^{n\times n}$
// $\mathbf{h} = \mathbf{X}(\mathbf{e}^* + \mathbf{e}) + \mathbf{E}(-\mathbf{s}^* - \mathbf{s}) + \mathbf{f}$
$\mathbf{c} \leftarrow \mathbf{C}(\mathbf{s}^* + \mathbf{s}) + \mathbf{h} + \lfloor q/p \rfloor \cdot \mathbf{m}_\beta \bmod q$
$\overline{\mathbf{s}}, \overline{\mathbf{e}} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$
$sk' \leftarrow (\mathbf{s}^* + \mathbf{s} + \overline{\mathbf{s}}, \mathbf{e}^* + \mathbf{e} + \overline{\mathbf{e}})$
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** β'
return $(\beta = \beta')$

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

Note that $\mathcal{B}^{(i)}$'s query satisfies $\|\mathbf{z}_1\|_\infty \le \|\mathbf{s}^*\|_\infty + \|\mathbf{s}\|_\infty \le y\sigma + \ell y\sigma$ (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 $\|\mathbf{z}_0\|_\infty \le B$, and thus

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

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

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

Hybrid 1 → Hybrid 2. This statistical argument follows from Corollary 3. Instead of sampling $\mathbf{s}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$ and $\mathbf{s}^* + \overline{\mathbf{s}}_i =: \mathbf{s}_i^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{s}^*}$ in the i-th `Chal` query, we first sample $\mathbf{s}_i^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_i,\mathbf{s}_i^\circ}$ for all $1 \le i \le k$, where $\sigma_i := \sqrt{(i+1)/i} \cdot \sigma$ and $\mathbf{s}_i^\circ := \frac{1}{i}\sum_{j=1}^{i-1} \mathbf{s}_j^+$; and only then we sample $\mathbf{s}^* \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma/\sqrt{k+1},\mathbf{s}_2^*}$ with $\mathbf{s}_2^* := \frac{1}{k+1}\sum_{j=1}^{k} \mathbf{s}_j^+$.

Since, by assumption $\sigma/\sqrt{k+1} > \sqrt{(\ln(2n(1+1/\varepsilon)))/\pi}$, we can apply Corollary 3, which yields that the statistical distance from the previous distribution is bounded by $k \cdot \frac{4\varepsilon}{(1-\varepsilon)^2}$. To prepare for the next game hop, we sample $\mathbf{s}^*$ by first sampling $\mathbf{s}_1^*$ centered at $\mathbf{0}$ and then adding $\mathbf{s}_2^*$.

The exact same resampling is done to $\mathbf{e}^*$ and $\mathbf{e}_i^+ := \mathbf{e}^* + \overline{\mathbf{e}}_i$ for $1 \le i \le k$, where $\overline{\mathbf{e}}_i$ is sampled in the i-th `Chal` query. This results in another loss of $k \cdot \frac{4\varepsilon}{(1-\varepsilon)^2}$.

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

$\beta \leftarrow\!\!\$\ \{0,1\}$; $j \leftarrow 0$
parse $\overline{\mathbf{A}}$ as $[\mathbf{A}^\top \| \mathbf{D}_1^\top \| \ldots \| \mathbf{D}_k^\top]^\top$
parse $\overline{\mathbf{b}}$ as $[\mathbf{b}^\top \| \mathbf{d}_1^\top \| \ldots \| \mathbf{d}_k^\top]^\top$
// $\mathbf{b} = \mathbf{A}\mathbf{s}_1^* + \mathbf{e}_1^*$ or $\mathbf{b} \leftarrow \mathbb{Z}_q^n$
// $\mathbf{d}_i = \mathbf{D}_i\mathbf{s}_1^* + \mathbf{f}_{i,1}^*$ or $\mathbf{d}_i \leftarrow \mathbb{Z}_q^n$
for $i = 1 \ldots k$
  $\sigma_i \leftarrow \sqrt{\frac{i+1}{i}}\sigma$
  $\mathbf{s}_i^\circ \leftarrow \frac{1}{i}\sum_{j=1}^{i-1} \mathbf{s}_j^+$; $\mathbf{e}_i^\circ \leftarrow \frac{1}{i}\sum_{j=1}^{i-1} \mathbf{e}_j^+$
  $\mathbf{s}_i^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_i,\mathbf{s}_i^\circ}$; $\mathbf{e}_i^+ \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_i,\mathbf{e}_i^\circ}$
$\mathbf{s}_2^* \leftarrow \frac{1}{k+1}\sum_{j=1}^{k} \mathbf{s}_j^+$
$\mathbf{e}_2^* \leftarrow \frac{1}{k+1}\sum_{j=1}^{k} \mathbf{e}_j^+$
$\mathbf{b}^* \leftarrow \mathbf{b} + \mathbf{A}\mathbf{s}_2^* + \mathbf{e}_2^*$
$\beta' \leftarrow \mathcal{A}^{\texttt{Chal}}(\mathbf{A}, \mathbf{b}^*)$
return $(\beta = \beta')$

Oracle $\texttt{Chal}(\mathbf{m}_0, \mathbf{m}_1, (\mathbf{s}, \mathbf{e}))$

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

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

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 $\mathbf{f}_1 \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_1}$ and $\mathbf{f}_2 \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma_2}$, with $\sigma_1 := \sigma/\sqrt{k+1}$ and $\sigma_2 := \sqrt{\sigma_c^2 - \sigma^2/(k+1)}$. By Lemma 1, this is $(\frac{2\varepsilon}{1-\varepsilon})$-close to the distribution of $\mathbf{f}$ in Hybrid 1. (Note that from $\sigma_c > \sigma$, we have $\sigma_1\sigma_2/\sqrt{\sigma_1^2 + \sigma_2^2} \geq \sigma/\sqrt{k+1}$ and thus the premise of the lemma is again satisfied.) In total, we get:

$$\left|\mathsf{Adv}^{\text{Hybrid 2}}(\mathcal{A}) - \mathsf{Adv}^{\text{Hybrid 1}}(\mathcal{A})\right| \leq 5k \cdot \frac{2\varepsilon}{(1-\varepsilon)^2} \ . \tag{16}$$

Hybrid 2 → Hybrid 3. The last hop is by reduction to LWE with parameters $m := (k+1)n$ and standard deviation $\sigma/\sqrt{k+1}$, against which we construct the reduction $\mathcal{C}$ in Fig. 23:
We first analyze the case when $[\mathbf{b}^\top \| \mathbf{d}_1^\top \| \ldots \| \mathbf{d}_k^\top]^\top$ 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 $\mathbf{s}_1^*, \mathbf{e}_1^*, \mathbf{f}_{j,1} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma/\sqrt{k+1}}$. 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

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

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

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

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

$r \leftarrow\!\!\$\ \mathcal{RS}$
$pk \leftarrow G(s)$
$c \leftarrow \mathsf{Encrypt}(pp, pk, w; r)$
$\pi' \leftarrow \Pi.\mathsf{P}^H(crs', (x, pk, c), (r, w))$
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. 24: The NIZKPoK construction making black box use of a PKE scheme PKE with randomness space $\mathcal{RS}$, a NIZK Π 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. 25.

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. 25: The alternative setup, simulator and extraction algorithms for NIZKPoK construction $\mathsf{PoK}[\mathsf{PKE}, \Pi, G, H]$ in Fig. 24. We assume w has constant length ℓ.

H 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, $\mathcal{PK} = \mathbb{Z}_q^m$)[24] and randomness space $\mathcal{RS}$, a random oracle G with range $\mathcal{PK}$ and a NIZK (that uses a different random oracle H). The construction Π can be found in Fig. 24; in Fig. 25 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 (using random oracle H) that the ciphertext contains a witness for x.[25]

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. (1)), we get

$$\begin{aligned}&(\tilde{x} = ((pp_{\mathsf{PKE}}, pk_i, pk_{i+1}), pp, pk, c), \tilde{w} = (\overline{sk}, r)) \in \tilde{\mathcal{R}} \\ &\quad\iff c = \mathsf{Encrypt}(pp, pk, \overline{sk}; r) \ \wedge\ \mathsf{pk}(\overline{sk}) = pk_{i+1} - pk_i \ \wedge\ \overline{sk} \in \mathcal{SK} \ \wedge\ r \in \mathcal{RS}, \quad (18)\end{aligned}$$

where $\mathsf{pk}(\cdot)$ is implicitly parametrized by pp_{PKE}.

[24] Note that $\mathcal{PK}$ may contain values outside support of key generation.
[25] Technically, PoK is defined with one RO. We use two for exposition, without loss of generality.

Completeness. We show that completeness is preserved based on the underlying completeness property.

Lemma 6. *If NIZK is δ-complete, then NIZKPoK in Fig. 24 is δ-complete.*

Proof. Consider the completeness game with the NIZKPoK from Fig. 24. 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. (1). It samples randomness r from $\mathcal{RS}$ 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. By completeness of the NIZK, verification only fails with probability at most δ. □

Setup Indistinguishability. 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. We recall the definition in Appendix C.1. The following lemma follows from a straightforward reduction.

Lemma 7. *If PKE is fuzzy then NIZKPoK in Fig. 24 guarantees setup-indistinguishability (Definition 18 in Appendix C.2). In particular, for any adversary $\mathcal{A}$, there exists a reduction $\mathcal{B}$ such that*

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

Zero-Knowledge. 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 8. *If PKE is IND-CPA secure and NIZK is zero-knowledge, then NIZKPoK in Fig. 24 is zero-knowledge (Definition 19). In particular, for any adversary $\mathcal{A}$ there exist reductions $\mathcal{B}$, $\mathcal{B}'$ such that*

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

Proof (of Lemma 8). The proof proceeds in two steps. We start with the $\mathsf{ZK\text{-}PoK}$ game with the bit $b = 0$. In the first step we modify the honest prover to produce a simulated proof π'. The zero knowledge property of Π 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 $\mathsf{ZK\text{-}PoK}$ game when $b = 1$ which concludes the proof. □

Straightline Simulation-Extractability. The trapdoor td for the extractor contains the secret key sampled by the alternative setup. Thus, a witness can be extracted from a proof π by decrypting the ciphertext contained in π. If π 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 9. *If PKE is δ-correct under adversarially chosen randomness and NIZK is adaptively simulation-sound, then NIZKPoK in Fig. 24 is straightline simulation-extractable (Definition 20 in Appendix C.2). More precisely, if PKE is δ-correct under adversarially chosen randomness then for any adversary $\mathcal{A}$ there exist a reduction $\mathcal{B}$ such that*

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

Proof (of Lemma 9). 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 π in Condition 3 enforces that only well-formed ciphertexts c are passed to Decrypt, where well-formedness means that the adversary used randomness from $\mathcal{RS}$. We formalize this next using a game hop.

Game 0. This is the Ext experiment for PoK.

$$\mathsf{Adv}^{\mathsf{Ext}}_{\mathsf{PoK}}(\mathcal{A}) \coloneqq \Pr\left[\mathbf{Exp}_0(\mathcal{A})\right] \ . \tag{19}$$

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 π' 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

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

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^ℓ and queries its own simulate oracle to obtain a proof π'. If BrkSS occurs for x, c and π', then $\mathcal{B}$ breaks simulation soundness with statement $\tilde{x} = (x, pk, c)$ and proof π'. Indeed, Condition 3 of Win implies that NIZK.V accepts $\tilde{x}$ and π'. 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 Win occurs but BrkSS does not. So assume $\mathcal{A}$ wins with x, c, π'. Since 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, Decrypt outputs an incorrect witness w' (which may also be $\bot$). By correctness of PKE under adversarially chosen randomness, since $w \in \mathcal{M}$, $r \in \mathcal{RS}$, this can only happen with probability at most δ. □

H.1 Instantiation from Lattices

To give a lattice-based instantiation of NIZKPoK from Fig. 24, we instantiate 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 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 (18) 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:

$$\begin{aligned}&(\tilde{x} = ((\mathbf{A}, \mathbf{b}_i, \mathbf{b}_{i+1}), pk_{\mathsf{ky}}, c_{\mathsf{ky}}), \tilde{w} = ((\mathbf{s}, \mathbf{e}), r_{\mathsf{ky}})) \in \tilde{\mathcal{R}} \\ &\iff c_{\mathsf{ky}} = \mathsf{Kyber.Encrypt}(pk_{\mathsf{ky}}, \mathbf{s}; r_{\mathsf{ky}}) \ \wedge \ \|\mathbf{s}\|_2 \leq B \wedge \|\mathbf{b}_{i+1} - \mathbf{b}_i - \mathbf{As}\|_2 \leq B \\ &\iff c_{\mathsf{ky}} = (\mathbf{C}, \mathbf{c}) = (\mathbf{XA}' + \mathbf{E}, \mathbf{Xb}' + \mathbf{f} + \lfloor q/p \rfloor \cdot [\mathbf{s} \mid \mathbf{e}]) \\ &\qquad \wedge \|\mathbf{X}\|_2, \|\mathbf{E}\|_2, \|\mathbf{f}\|_2 \leq B' \ \wedge \ \|\mathbf{s}\|_2, \|\mathbf{b}_{i+1} - \mathbf{b}_i - \mathbf{As}\|_2 \leq B\end{aligned} \tag{20}$$

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].