

On the Limits of LWE and PCE based UPKE

Martin R. Albrecht¹, Benjamin Benčina², and Russell W. F. Lai^{3*}

¹ King’s College London and SandboxAQ
`martin.albrecht@kcl.ac.uk, sandboxaq.com`

² Royal Holloway, University of London
`benjamin.bencina.2022@live.rhul.ac.uk`

³ Aalto University
`russell.lai@aalto.fi`

Abstract Updatable Public-Key Encryption (UPKE) enables forward secrecy in asynchronous settings like secure group messaging. At EC’25, Albrecht, Benčina and Lai proposed the first plausibly post-quantum UPKE scheme supporting unlimited updates, based on permutation code equivalence (PCE) over finite fields and hollow lattice problems. However, it has impractically large public keys (4.8MiB) and ciphertexts (1.5MiB).

In this work, we first replace the information-theoretic Hollow LHL with a computational assumption secure under binary-secret binary-error LWE, reducing key sizes by an order of magnitude. Second, we show that generalising to Module-LWE is not beneficial: PCE instances over $\mathbb{F}_{q^d}$ admit a linearisation attack via the q -Frobenius structure, enabling recovery of secret isometries unless code rates are inflated by d , negating any size benefit from module lattices. Third, we establish IND-CCA security in the model of Alwen, Fuchsbauer and Mularczyk (EC’24), but do not achieve ‘joiner security’ like other post-quantum candidates.

Finally, we introduce a heuristic multi-key variant using matrix-valued secret keys, enabling efficient encryption of ℓ bits per ciphertext. We define the Hinted SPCE assumption to capture the main cryptanalytic challenge and show that the performance gain is likely too small to justify adoption, unless the new assumption can be avoided.

Overall, our ‘unlimited’ scheme has update sizes 3.5 to 10 times larger than lattice-based schemes limited to 2^{20} updates, depending on the aggressiveness of parameter choices.

1 Introduction

Updatable public-key encryption (UPKE), introduced in [JMM19,ACDT20], is a cryptographic primitive based on public-key encryption (PKE) that allows anyone to update a public key and generate an update token, which can be used by the secret key holder to update their secret key to one that corresponds to the new public key. The security guarantee is one of *forward secrecy* – ciphertexts encrypted under old public keys remain secure even if an updated secret key is compromised, as long as at least one update was performed honestly in between. Compared to the classical solution of using Forward-Secure PKE [CHK03] to achieve forward secrecy, UPKE schemes win on multiple fronts. By maintaining a strict chain of public-private key pairs, FS-PKE schemes typically suffer in efficiency, whereas relaxing this makes the overhead of the UPKE update mechanism over the underlying PKE comparatively small (below, we consider update mechanisms which amount to merely adding randomness to the key, or performing a linear operation).

1.1 UPKE from Lattices

The initial works [JMM19,ACDT20] are based on discrete logarithm problems, hardness assumptions known to be broken if a large enough quantum computer is ever built [Sho94]. The authors of [DKW21] constructed the

* The research of Albrecht, Benčina and Lai was supported by UKRI Grant EP/Y02432X/1, by UKRI Grant EP/S021817/1, and by the Research Council of Finland grant 358951 and the Future Makers Program of the Technology Industries of Finland Centennial Foundation via the project SPICED, respectively.

first UPKE scheme based on a post-quantum hardness assumption, namely Learning with Errors (LWE) [Reg05], and soon more followed suit. The authors of [HPS23] constructed another scheme based on LWE and its more efficient module version (MLWE), though they make an assumption without proof that security for the MLWE setting simply translates from that for the LWE setting. The authors of [AFMR26] constructed a further such scheme with similarly attractive parameters to those of [HPS23] and also proved it secure in a relaxed version of the stronger security model proposed in [AFM24].

All of the above lattice-based schemes have the disadvantage that they only support a fixed pre-determined number of updates before the secret key spoils unless they accept a super-polynomial LWE modulus. This is a consequence of their simple update mechanism, as these three schemes work by sampling some randomness (a fresh key) and *adding* it to the public key. This randomness is then encrypted into the update token, which is decrypted by the receiver and also *added* to the secret key in a way that makes the new key pair valid. Since secrets in LWE-based PKE schemes are (essentially) required to be short (in vector norm), and adding increases the norm, the secret keys gradually grow in norm over a number of updates until they eventually become too long to decrypt correctly. The parameters for these schemes are then chosen to accommodate a desired number of updates, typically not more than 2^{20} . However, we can imagine a multi-year multi-user session exceeding this limit.⁴

To solve this, a new UPKE scheme was proposed in [ABL25a] that used a lattice-based PKE scheme along with an update mechanism inspired by the Lattice Isomorphism Problem (LIP) [DvW22] and formally based on permutation code equivalence (PCE). Concretely, instead of adding randomness to the key pair, the keys are *rotated* by a random lattice isometry, that is an orthogonal matrix from $\mathcal{O}_n(\mathbb{R})$. To ensure that all key material stays integral, and because $\mathcal{O}_n(\mathbb{R})$ is an infinite group, the isometries are chosen from $\mathcal{O}_n(\mathbb{Z})$ instead, the finite group of signed permutations. The upshot is that isometries do not change vector norms, so after any number of updates, the updated key is exactly as long as the initial one. However, the scheme pays a price in complexity and efficiency, since the parameters have to be such that both LWE and PCE are hard simultaneously. The work [ABL25a] achieves this by considering *hollow lattice problems*, these are versions of lattice problems with the additional property that the matrices featured in the problems generate linear codes whose hull dimension is big enough. The resulting scheme, while admitting a super-polynomial number of updates, is not practical. For example, the public key in the first row of [ABL25a, Tab. 1] is 4.79 MiB in size, and a single ciphertext encrypting 128 bits is 1.49 MiB in size. In [ABL25a, Sec. 1.4], the authors list some potential future directions, all aiming towards a more efficient scheme with smaller public key and ciphertext sizes.⁵

1.2 Our Contributions

In this work we investigate potential improvements to the UPKE scheme from [ABL25a], some listed in their open problems section and some new. We present mostly positive results (1, 3 and 4), where (4) requires a new computational assumption, and a negative result (2). Table 1 contains proposed parameters.

1. Replacing the LHL with a computational assumption. One of the reasons why the public keys are so large in the UPKE scheme from [ABL25a] is the use of an information-theoretic game hop to argue that a dual-Regev public key $(\mathbf{A}, \mathbf{u})$, where $\mathbf{u}$ is the syndrome of a short vector $\mathbf{r}$, is indistinguishable from the case where $\mathbf{u} \in \mathbb{Z}_q^k$ is uniformly random, which necessitates $n > k \cdot \log(q)$. While this hop can be replaced by LWE for uniformly random $\mathbf{A} \in \mathbb{Z}_q^{n \times k}$ with $n < k \cdot \log(q)$ [LP11], the case where $\mathbf{A}$ is h -hollow for some hull dimension h was only proven in its information-theoretic form as the Hollow LHL [ABL25a, Lem. 9]. In Theorem 1, we give a computational version of this statement, called Hollow DSIS, and prove it is equivalent to Hollow LWE on the dual code, thus the same at rate $\frac{1}{2}$. Since the Hollow LHL requires $n > k \cdot \log(q) + k + h$ and typically $\log(q) \geq 12$, this reduces sizes by an order of magnitude.

⁴ However, *post-compromise security* concerns might rule out such long update chains.

⁵ We also note that follow-up work [BMS26] showed that the parameters proposed in [ABL25a, Tab. 1] were not secure, but this is solved by increasing the hull dimension $h \gg \sqrt{2n}$ without a change in sizes of the public keys or the ciphertexts (see Section 8).

2. *Generalisation to MLWE is pointless.* Another natural improvement is to consider MLWE instead of LWE, for module lattices over a ring of integers $\mathcal{R}$ of a cyclotomic field $\mathcal{K}$ with conductor f . To remain finite, the quotient ring $\mathcal{R}_q$ is normally considered for some rational prime q . We show in Section 6 that translating the UPKE scheme from [ABL25a] to MLWE leads to relying on the hardness of PCE over *finite extension fields*. Namely, we show that if q is inert in $\mathcal{R}$, then we are relying on the hardness of PCE over $\mathbb{F}_{q^d}$ for $d = \phi(f)$. We then show how to use the q -Frobenius structure of $\mathbb{F}_{q^d}$ to amplify every PCE sample into d samples sharing the same secret isometry, allowing us to mount the “Don’t Use It Twice!” linearisation attack [BCD⁺24b]. To counteract this, we are forced to increase the code rate by a factor of d , which negates any public key size reduction that comes from using MLWE. We then show that if q is not inert in $\mathcal{R}$, the splitting behaviour of q in $\mathcal{R}$ together with the inert case technique on each CRT slot still leads to obtaining d PCE samples from a single PCE sample.

3. *IND-CCA security of UKEM.* In Section 5 we give an updatable key encapsulation mechanism (UKEM) based on the UPKE scheme from [ABL25a], and claim it is secure in the IND-CCA model from [AFMR26].⁶ To achieve *member security* in the IND-CCA game, we introduce member tags for the scheme from [ABL25a] in Section 4, which are essentially just proofs of permutation equivalence similar to those from [PS23], except that we replace the FS transform [FS87] with Fischlin’s transform [Fis05] offering online extraction. To make sure the syndrome part of the dual-Regev public key was also correctly updated, we slightly modify the usual Graph Isomorphism-style Σ -protocol used in [PS23] to only verify successfully if the isometry fixes the last coordinate (wherein the syndrome lies). Like other post-quantum candidates, we neither target nor achieve ‘joiner security’ [AFM24]. The rest of the proof merely amounts to a slight relaxation and modularisation of the transformations and results from [AFMR26] that allow us to use their IND-CCA model with unbounded UPKE schemes, including the one from [ABL25a]. With the recently proposed isogeny-based unbounded UPKE scheme [BFK⁺26], we believe our generalised transformation is independently useful.

4. *A multi-key version of the scheme.* We give in Section 7 a multi-key version of the scheme from [ABL25a], allowing us to reuse a large part of the ciphertext to encrypt multiple bits, as in, e.g., [NAB⁺20]. Namely, instead of a single short vector $\mathbf{r} \in \{\pm 1\}^n$, the secret key is a matrix $\mathbf{R} \in \{\pm 1\}^{n \times \ell}$ for some small $\ell \in \mathbb{N}$. The public key now contains ℓ syndromes instead of just one, but this slight increase in the public key size is justified by a major decrease in ciphertext size when encrypting many bits (say, in a UKEM), since we can now encrypt ℓ times as many bits with only a marginally larger ciphertext. We argue the security of our multi-key scheme with a *new assumption* that we call Hinted SPCE. Using (3), the multi-key scheme can be compiled into an IND-CCA-secure UKEM, and we provide member tags for the corresponding UKEM with only a marginal overhead. We also initialise the cryptanalysis of Hinted SPCE by showing that [ABL25a, Lem. 11] implies that the hardness of the base case $\ell = 1$ follows from the hardness of Δ PCE and suggesting some future directions. Finally, in Table 1 we give parameters that were optimised for size given a target security level, and explain how they were generated in Section 8. As can be gleaned from this table, the Hinted SPCE assumption seems hardly ‘worth it’ for only giving a marginal improvement in size at the cost of introducing a completely new assumption; ultimately offering about a 20% decrease in the size of the output, that is $|\text{pk}| + |\text{ctxt}| + |\text{mt}|$. This is mostly due to the need to output a fresh public key with each output, which is large compared to a ciphertext.

1.3 Open Problems

The central open problem in this area of work remains to improve the performance of UPKE constructions in the spirit of [ABL25a]. A promising direction is to consider LIP [DvW22, BGPS23] directly instead of the specialisation to PCE. While [BFK⁺26] recently proposed an unbounded UPKE from isogenies with characteristically small sizes, doing so from lattice-based assumptions remains open. More generally, a post-quantum candidate solution for joiner security is a pressing topic for future research.

⁶ This is to follow recent works [AFM24, AFMR26, HPS23] that prefer to study UKEMs over UPKE, and it also allows for fairer size comparison.

Table 1: Parameters for the given λ and p . Total size is $|\text{pk}| + |\text{ctxt}| + |\text{mt}|$.

λ	p	n	k	$\log(q)$	$h \geq$	$ \text{pk} $	$ \text{ctxt} $	$ \text{mt} $	Total
Our scheme with 1 and 16 keys, respectively ($\sigma = 3.2$)									
128	2	1220	610	12	69	546.0 KiB	230.4 KiB	130.1 KiB	906.5 KiB
128	2	1220	610	12	84	559.4 KiB	14.4 KiB	131.5 KiB	705.3 KiB
Our scheme with 1 and 16 keys, respectively ($\sigma = 0.5$)									
128	2	980	490	9	65	264.3 KiB	140.8 KiB	99.9 KiB	505.0 KiB
128	2	980	490	9	80	272.4 KiB	8.8 KiB	101.2 KiB	382.4 KiB
[ABL25a] with 128-bit msg (new params. and mt, $\sigma = 3.2$)									
128	2	6825	420	13	131	4269.6 KiB	1382.4 KiB	762.2 KiB	6414.2 KiB
128	16	10600	530	16	159	10425.1 KiB	662.4 KiB	1257.9 KiB	12345.4 KiB
[ABL25a] with 128-bit msg (new params. and mt, $\sigma = 0.5$)									
128	2	6625	530	10	130	3943.9 KiB	1036.8 KiB	740.4 KiB	5721.1 KiB
[HPS23] and [AFMR26] with 2^{20} updates, respectively									
137	–	–	–	36	–	13.9 KiB	130.7 KiB	–	144.6 KiB
137	–	–	–	36	–	6.9 KiB	86.9 KiB	–	93.8 KiB

2 Preliminaries

We write $[\mathbf{A} \parallel \mathbf{B}]$ for vertical concatenation, i.e. $[\mathbf{A}^T \mid \mathbf{B}^T]^T$. All logarithms are base 2. Denote by $\mathcal{P}_n$ and $\mathcal{O}_n(\mathbb{Z})$ the group of $n \times n$ permutation and signed permutation matrices, respectively.

Let q be a prime, and $n \geq k > 0$ be positive integers. An $[n, k]$ -linear code $\mathcal{C}$ over $\mathbb{F}_q$ of length n and dimension k is a k -dimensional vector subspace of $\mathbb{F}_q^n$. The rate of $\mathcal{C}$ is $\frac{k}{n}$. Its vectors are called codewords, and it is typically represented by a full-rank generator matrix $\mathbf{G} \in \mathbb{F}_q^{n \times k}$ whose columns generate $\mathcal{C}$. We can always choose a different basis – any $\mathbf{G}' = \mathbf{G} \cdot \mathbf{U}$ for $\mathbf{U} \in \mathcal{GL}_k(\mathbb{F}_q)$ is also a generator matrix for $\mathcal{C}$. The *systematic form* of $\mathcal{C}$ is the unique generator matrix $\mathbf{S}$ for $\mathcal{C}$ that is column-reduced, and is thus the canonical choice of basis. For a generator matrix $\mathbf{G}$ of $\mathcal{C}$, we denote by $\text{SF}(\mathbf{G})$ the systematic form of $\mathcal{C}$ computed by column-reducing $\mathbf{G}$, and also output the transformation matrix when needed. A code may equivalently be represented as the kernel of its parity check matrix $\mathbf{H} \in \mathbb{F}_q^{n \times (n-k)}$, that is we have $\mathbf{v} \in \mathcal{C} \iff \mathbf{v}^T \cdot \mathbf{H} = \mathbf{0}$. The code generated by $\mathbf{H}$ is called the *dual code* to $\mathcal{C}$, i.e. $\mathcal{C}^\perp = \{\mathbf{y} \in \mathbb{F}_q^n \mid \forall \mathbf{x} \in \mathcal{C}: \mathbf{x}^T \cdot \mathbf{y} = 0\}$. The generator matrices of $\mathcal{C}^\perp$ are precisely the parity check matrices of $\mathcal{C}$, vice versa.

A code $\mathcal{C}$ is called self-orthogonal (also weakly self-dual) if $\mathcal{C} \subseteq \mathcal{C}^\perp$. The intersection of a code with its dual is called the *hull* of $\mathcal{C}$, and is denoted by $\text{hull}(\mathcal{C}) = \mathcal{C} \cap \mathcal{C}^\perp$. It follows that a code is self-orthogonal if and only if $\text{hull}(\mathcal{C}) = \mathcal{C}$. A vector $\mathbf{x} \in \mathbb{F}_q^n$ is called self-orthogonal if $\mathbf{x}^T \cdot \mathbf{x} = 0$.

Lemma 1 (cor. of [Sen97, Thm. 4.18], in [ABL25a, Lem. 1]). *The probability that a randomly sampled $[n, k]$ -linear code over $\mathbb{F}_q$ has a trivial hull is at least $1 - \frac{1}{q} - \frac{1}{q^2}$.*

2.1 Indistinguishable Distributions

Recently, the author of [Ben26] introduced an invariant for (S)PCE called the square class $\chi_{\mathcal{C}} \in \{\pm 1\}$ of a linear code $\mathcal{C}$, restricted the definition of (S)PCE and provided a sampler for such instances, which we use.

Definition 1 ((Signed) Permutation Code Equivalence [ABL25a, Ben26]). *Let $n, k, q \in \mathbb{Z}$ parametrised by λ with $k < n$ and q prime, and let $\mathcal{D}$ be a distribution on the subset of full-rank matrices in $\mathbb{F}_q^{n \times k}$. The advantages of any PPT adversary $\mathcal{A}$ against $\Delta\text{PCE}_{n,k,q}(\mathcal{D})$ and $\Delta\text{SPCE}_{n,k,q}(\mathcal{D})$ are defined as*

$$\text{Adv}_{\Delta(\text{S})\text{PCE}}(\mathcal{A}) = \left| \Pr \left[b = b' \mid \begin{array}{l} \mathbf{G} \leftarrow \$_\mathcal{D}; \mathbf{P} \leftarrow \$_\mathcal{G}; \mathbf{U} \leftarrow \$_{\mathcal{GL}_k(\mathbb{F}_q)} \\ \mathbf{H}_0 \leftarrow \$_\mathcal{D}; \mathbf{H}_1 := \mathbf{P} \cdot \mathbf{G} \cdot \mathbf{U} \\ b \leftarrow \$_{\{0,1\}}; b' \leftarrow \mathcal{A}(\mathbf{G}, \mathbf{H}_b) \end{array} \right] - \frac{1}{2} \right|,$$

KGen(1^λ)	Enc(pk := ($\mathbf{A}, \mathbf{u}$), msg $\in \mathcal{M}$)
$\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times k}; \mathbf{r} \leftarrow \{\pm 1\}^n; \mathbf{u}^\top := \mathbf{r}^\top \cdot \mathbf{A} \bmod q$	$\mathbf{x} \leftarrow \mathbb{Z}_q^k; \mathbf{e} \leftarrow \eta^n; e' \leftarrow \eta$
return (pk := ($\mathbf{A}, \mathbf{u}$), sk := $\mathbf{r}$)	$\mathbf{c}_0 := \mathbf{A} \cdot \mathbf{x} + \mathbf{e} \bmod q$
Dec (sk := $\mathbf{r}$, ctxt := ($\mathbf{c}_0, c_1$))	$c_1 := \langle \mathbf{u}, \mathbf{x} \rangle + e' + \left\lfloor \frac{q}{p} \right\rfloor \cdot \text{msg} \bmod q$
return $\left\lfloor \frac{p}{q} (c_1 - \langle \mathbf{r}, \mathbf{c}_0 \rangle \bmod q) \right\rfloor$	return ctxt := ($\mathbf{c}_0, c_1$)

Figure 1: Dual-Regev PKE scheme with $\mathbf{r} \in \{\pm 1\}^n$.

where $\mathcal{G}$ is $\mathcal{P}_n$ for ΔPCE and $\mathcal{O}_n(\mathbb{Z})$ for ΔSPCE , and assumed negligible [SS13] in λ if $\mathbf{D} = \mathbf{D}_{n,k,q}^{\chi,h}$ that samples uniformly random $[n, k]$ -linear h -hollow codes over $\mathbb{F}_q$ with square class χ , in which case we simply write $\Delta(\text{S})\text{PCE}_{n,k,q}^{\chi,h}$.

Definition 2 (Learning With Errors [Reg05]). Let $n, k, q \in \mathbb{Z}$ parametrised by λ , η an error distribution over $\mathbb{Z}$, and $\mathbf{D}$ a secret distribution over $\mathbb{Z}_q^k$. The advantage of any PPT adversary $\mathcal{A}$ against $\text{LWE}_{k,n,q,\eta}(\mathbf{D})$ is defined as

$$\text{Adv}_{\text{LWE}}(\mathcal{A}) = \left| \Pr \left[b = b' \mid \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times k}; \mathbf{s} \leftarrow \mathbf{D}; \mathbf{e} \leftarrow \eta^n \\ \mathbf{y}_0 \leftarrow \mathbb{Z}_q^n; \mathbf{y}_1 := \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q \\ b \leftarrow \{0, 1\}; b' \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{y}_b) \end{array} \right] - \frac{1}{2} \right|,$$

and assumed negligible in λ for $(\mathbf{D} = \mathcal{U}(\mathbb{Z}_q^k), \eta = \mathcal{D}_{\mathbb{Z},s})$.⁷

We rely on the signed-secret Dual-Regev PKE scheme, used in [ABL25a].

Definition 3 (Dual-Regev PKE [GPV08]). The signed-secret dual-Regev PKE scheme in Fig. 1 is a variant of dual-Regev PKE for $\mathcal{M} = \mathbb{Z} \cap [-p/2, p/2)$ where the long-term secret vector $\mathbf{r} \in \{\pm 1\}^n$ is a random signed vector.

2.2 Hollow Lattice Problems

Definition 4 (h -hollow matrices and codes). Let $n \geq k \geq h$ integers and q a prime power. We call a code $\mathfrak{C}$ h -hollow if $\dim \text{hull}(\mathfrak{C}) = h$. A matrix is called h -hollow if it is full-rank and generates a $[n, k]$ -linear h -hollow code over $\mathbb{F}_q$.⁸

Definition 5 (Hollow LWE [ABL25a, Def. 12]). Let $n, k, h, q \in \mathbb{Z}$ parametrised by λ with $0 \leq h \leq k \leq n$ and q prime, η an error distribution over $\mathbb{Z}$, and $\mathbf{D}_{n,k,q}^h$ the distribution of uniformly random h -hollow matrices in $\mathbb{F}_q^{n \times k}$. The advantage of any PPT adversary $\mathcal{A}$ against $\text{LWE}_{k,n,q,\eta}^h$ is defined as

$$\text{Adv}_{\text{LWE}^h}(\mathcal{A}) = \left| \Pr \left[b = b' \mid \begin{array}{l} \mathbf{A} \leftarrow \mathbf{D}_{n,k,q}^h; \mathbf{s} \leftarrow \mathbb{F}_q^k; \mathbf{e} \leftarrow \eta^n \\ \mathbf{y}_0 \leftarrow \mathbb{F}_q^n; \mathbf{y}_1 := \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q \\ b \leftarrow \{0, 1\}; b' \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{y}_b) \end{array} \right] - \frac{1}{2} \right|.$$

Recall from [ABL25a, Thm. 1] that $\text{LWE}_{k-h,n,q,\eta} \rightarrow \text{LWE}_{k,n,q,\eta}^h$. Note that the square class of the instance matrix $\mathbf{A}$ does not play a role here, as this reduction preserves it. Per [Ben26, Sec. 5.2] we only need to account for it when using ΔPCE .

Lemma 2 (Hollow LHL [ABL25a, Lem. 9]). Let $n \geq (1+c) \cdot k \cdot \log(q) + k + h$ for a positive real constant $c > 0$, let $h \leq \frac{k}{2}$, and q an odd prime. Let $\mathbf{A} \in \mathbb{F}_q^{n \times k}$ be a uniformly random h -hollow matrix, $\mathbf{r} \leftarrow \{\pm 1\}^n$, and $\mathbf{u} \leftarrow \mathbb{F}_q^k$. Then the pairs $(\mathbf{A}, \mathbf{r}^\top \cdot \mathbf{A})$ and $(\mathbf{A}, \mathbf{u}^\top)$ are statistically close in k .

⁷ We recall that, for any $x \in \mathbb{Z}$, $\mathcal{D}_{\mathbb{Z},s}(x) = \rho_s(x)/\rho_s(\mathbb{Z})$ where $\rho_s(x) = \exp(-\pi x^2/s^2)$.

⁸ When $n = 2k$, such codes are also sometimes called h -dual.

$\text{Game}_{\text{IND-CR-CPA}}^A(\lambda)$	$\text{OUpd}(\rho)$
$i := 0; b \leftarrow \{0, 1\}; (\text{pk}_0, \text{sk}_0) \leftarrow \text{KGen}(1^\lambda)$	$(\text{pk}_{i+1}, \text{up}_i) \leftarrow \text{UpdPk}(\text{pk}_i; \rho)$
$(\text{st}, \text{msg}_0, \text{msg}_1) \leftarrow \mathcal{A}^{\text{OUpd}}(\text{pk}_0); \text{ctxt} \leftarrow \text{Enc}(\text{pk}_i, \text{msg}_b)$	$\text{sk}_{i+1} \leftarrow \text{UpdSk}(\text{sk}_i, \text{up}_i)$
$\text{st} \leftarrow \mathcal{A}^{\text{OUpd}}(\text{ctxt}, \text{st}); (\text{pk}, \text{up}) \leftarrow \text{UpdPk}(\text{pk}_i)$	$i := i + 1$
$\text{sk} \leftarrow \text{UpdSk}(\text{sk}_i, \text{up}); b' \leftarrow \mathcal{A}(\text{pk}, \text{sk}, \text{up}, \text{st})$	
return $b = b'$	

Figure 2: The IND-CR-CPA game for a UPKE scheme.

2.3 Updatable Public-Key Encryption and Key Encapsulation

We recall the necessary definitions for updatable public-key encryption.

Definition 6 (UPKE [DKW21, Def. 3]). *An updatable public-key encryption (UPKE) scheme for a message space $\mathcal{M}$ is a collection of five PPT algorithms:*

- Key generation.** *Takes as input the security parameter 1^λ , and outputs the initial key-pair $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^\lambda)$.*
- Encryption.** *Takes as input a public key pk and a message $\text{msg} \in \mathcal{M}$, and outputs a ciphertext $\text{ctxt} \leftarrow \text{Enc}(\text{pk}, \text{msg})$.*
- Decryption.** *Takes as input a secret key sk and a ciphertext ctxt , and outputs a message or failure $\text{msg}/\perp \leftarrow \text{Dec}(\text{sk}, \text{ctxt})$.*
- Public key update.** *Takes as input a public key pk , and outputs $(\text{pk}', \text{up}) \leftarrow \text{UpdPk}(\text{pk})$ where pk' is a new public key and up is an update token.*
- Secret key update.** *Takes as input a secret key sk and an update token up , and outputs a new secret key or failure $\text{sk}'/\perp \leftarrow \text{UpdSk}(\text{sk}, \text{up})$.*

Definition 7 (UPKE correctness [DKW21, p. 264]). *A UPKE scheme $(\text{KGen}, \text{Enc}, \text{Dec}, \text{UpdPk}, \text{UpdSk})$ for a message space $\mathcal{M}$ is said to be correct if*

- $(\text{KGen}, \text{Enc}, \text{Dec})$ is a correct PKE scheme for message space $\mathcal{M}$, and
- For any $\lambda, t \in \mathbb{N}$, any $(\text{pk}_0, \text{sk}_0) \in \text{KGen}(1^\lambda)$, any $(\text{pk}_{i+1}, \text{up}_i) \in \text{UpdPk}(\text{pk}_i)$ and $\text{sk}_{i+1} \in \text{UpdSk}(\text{sk}_i, \text{up}_i)$ for $i \in [t]$, any message $\text{msg} \in \mathcal{M}$, it holds that

$$\Pr[\text{Dec}(\text{sk}_{i+1}, \text{Enc}(\text{pk}_{i+1}, \text{msg})) = \text{msg}] \geq 1 - \text{negl}(\lambda)$$

for any $i \in [t]$, where the probability is taken over the randomness of Enc .

Definition 8 (IND-CR-CPA security of UPKE [DKW21, Def. 4]). *Consider $\text{Game}_{\text{IND-CR-CPA}}^A$ in Fig. 2. A UPKE scheme admits indistinguishability under chosen randomness chosen plaintext attacks if for any PPT adversary $\mathcal{A}$ we have*

$$\text{Adv}_{\text{IND-CR-CPA}}^{\text{UPKE}}(\mathcal{A}) = \left| \Pr[1 \leftarrow \text{Game}_{\text{IND-CR-CPA}}^A(\lambda)] - \frac{1}{2} \right| \leq \text{negl}(\lambda).$$

Using ΔPCE for hollow codes, Hollow LWE , and Hollow LHL , the authors of [ABL25a] construct an IND-CR-CPA-secure UPKE scheme, which we recall in Fig. 3.

Proposition 1 ([ABL25a, Thm. 3]). *Let $n, k, h, q, \chi \in \mathbb{Z}$ parametrised by λ with $n \geq (1+c) \cdot k \cdot \log_2(q) + k + h$ for a real constant $c > 0$, $2h \leq k$, q odd prime and $\chi \in \{\pm 1\}$. If the advantage of any PPT adversary against $\text{LWE}_{k,n,q,\eta}^h$ and $\Delta\text{PCE}_{n,k,q}^{\chi,h}$ (fixed per [Ben26, Sec. 5.2]) is negligible in λ , the UPKE scheme from Fig. 3 is IND-CR-CPA secure.*

$\text{KGen}(1^\lambda)$ $\mathbf{A} \leftarrow \$ \mathcal{D}_{n,k,q}^{\chi,h} \quad // \text{ fix from [Ben26]}$ $\mathbf{r} \leftarrow \$ \{\pm 1\}^n; \text{ sk} := \mathbf{r}$ $\mathbf{u}^T := \mathbf{r}^T \cdot \mathbf{A} \bmod q; \text{ pk} := (\mathbf{A}, \mathbf{u})$ return (pk, sk)	$\text{Dec}(\text{sk}, \text{ctxt})$ parse ($\mathbf{r}, \mathbf{c}_0, c_1$) $\leftarrow$ (sk, ctxt) return $\left\lfloor \frac{p}{q} \cdot (c_1 - \langle \mathbf{r}, \mathbf{c}_0 \rangle \bmod q) \right\rfloor$
$\text{Enc}(\text{pk}, \text{msg} \in \mathbb{Z} \cap [-p/2, p/2))$ parse ($\mathbf{A}, \mathbf{u}$) $\leftarrow$ pk $\mathbf{x} \leftarrow \$ \mathbb{F}_q^k; \mathbf{e} \leftarrow \$ \eta^n; e' \leftarrow \$ \eta$ $\mathbf{c}_0 := \mathbf{A} \cdot \mathbf{x} + \mathbf{e} \bmod q$ $c_1 := \langle \mathbf{u}, \mathbf{x} \rangle + e' + \left\lfloor \frac{q}{p} \right\rfloor \cdot \text{msg} \bmod q$ return ctxt := ($\mathbf{c}_0, c_1$)	$\text{UpdPk}(\text{pk})$ parse ($\mathbf{A}, \mathbf{u}$) $\leftarrow$ pk; $\rho \leftarrow \$ \{0, 1\}^\lambda; \mathbf{O} \leftarrow \mathcal{H}(\rho)$ $(\mathbf{A}', \mathbf{U}) := \text{SF}(\mathbf{O} \cdot \mathbf{A}); \mathbf{u}'^T := \mathbf{u}^T \cdot \mathbf{U} \bmod q$ $\text{pk}' := (\mathbf{A}', \mathbf{u}'); \text{ up} \leftarrow \text{Enc}(\text{pk}, \rho); \text{ return } \text{pk}', \text{ up}$
	$\text{UpdSk}(\text{sk}, \text{up})$ parse $\mathbf{r} \leftarrow$ sk; $\rho \leftarrow \text{Dec}(\text{sk}, \text{up}); \mathbf{O} \leftarrow \mathcal{H}(\rho)$ $\mathbf{r}' := \mathbf{O} \cdot \mathbf{r}; \text{ return } \text{sk}' := \mathbf{r}'$

Figure 3: Construction of updatable encryption from [ABL25a, Fig. 4], using a random oracle $\mathcal{H}: \{0, 1\}^\lambda \rightarrow \mathcal{O}_n(\mathbb{Z})$. When encrypting more than $\log(p)$ bits, encrypt $\log(p)$ at a time and output a list of ciphertexts, and analogously for decryption.

We will rely on δ -correctness and γ -spreadness of (updatable) PKE.

Definition 9 (δ -correctness of PKE). *A PKE scheme $(\text{KGen}, \text{Enc}, \text{Dec})$ is δ -correct, if the following hold:*

- (i) $\Pr[\text{sk} \notin \mathcal{SK}; (\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^\lambda)] \leq 2^{-\delta}$,
- (ii) $\forall \text{msg} \in \mathcal{M} \forall \text{pk} \in \mathcal{PK} : \Pr[\text{ctxt} = \perp; \text{ctxt} \leftarrow \text{Enc}(\text{pk}, \text{msg})] \leq 2^{-\delta}$, and
- (iii) $\forall \text{msg} \in \mathcal{M} \forall \rho \in \{0, 1\}^\lambda : \text{if } (\text{pk}, \text{sk}) \leftarrow \text{KGen}(1^\lambda) \text{ with } \text{sk} \in \mathcal{SK} \text{ and } \text{ctxt} \leftarrow \text{Enc}(\text{pk}, \text{msg}; \rho) \text{ with } \text{ctxt} \neq \perp, \text{ then } \text{msg}' \leftarrow \text{Dec}(\text{sk}, \text{ctxt}) \text{ with } \text{msg}' = \text{msg}.$

A UPKE scheme is δ -correct, if it is correct after every update (cf. Definition 7).

Definition 10 (γ -spreadness of PKE [HHK17]). *A PKE scheme $(\text{KGen}, \text{Enc}, \text{Dec})$ is γ -spread, if for all $\text{msg} \in \mathcal{M}$, $\text{ctxt} \in \mathcal{C}$, and $(\text{pk}, \text{sk}) \in \text{Supp}(\text{KGen}(1^\lambda))$, we have that*

$$\Pr[\text{ctxt} = \text{Enc}(\text{pk}, \text{msg})] \leq 2^{-\gamma}$$

with probability taken over the randomness of Enc . A UPKE scheme is γ -spread, if it is spread after every update (cf. Definition 7).

Following prior work [AFM24, AFMR26], we are concerned with updatable key encapsulation mechanisms. We can generically construct a UKEM from UPKE as in Fig. 13.

Definition 11 (UKEM). *An updatable key encapsulation mechanism (UKEM) for a key space $\mathcal{K}$ is a collection of four PPT algorithms:*

Setup. *Takes as input the security parameter 1^λ , and outputs the public parameters $\text{pp} \leftarrow \text{Setup}(1^\lambda)$.*

Key generation. *Takes as input the public parameters pp , and outputs a key-pair $(\text{pk}, \text{sk}) \leftarrow \text{KGen}(\text{pp})$.*

Encapsulation. *Takes as input the public parameters pp and a public key pk , and outputs $(\text{kk}, \text{ctxt}, \text{pk}', \text{aux}) \leftarrow \text{Encaps}(\text{pp}, \text{pk})$ where $\text{kk} \in \mathcal{K}$ is the shared key, ctxt is a ciphertext, pk' is a public key, and aux stores auxiliary information (this can be, say, an update token, or a proof that pk' was correctly generated).*

Decapsulation. *Takes as input the public parameters pp , a secret key sk , a ciphertext ctxt , and auxiliary information aux , and outputs $(\text{kk}, \text{sk}')/\perp \leftarrow \text{Decaps}(\text{pp}, \text{sk}, \text{ctxt}, \text{aux})$ where kk is the shared key, and sk' is a secret key.*

Definition 12 (UKEM correctness). A UKEM = (Setup, KGen, Encaps, Decaps) is correct if for any $\text{pp} \leftarrow \text{Setup}(1^\lambda)$, $(\text{pk}_0, \text{sk}_0) \leftarrow \text{KGen}(\text{pp})$ and any $i \in \mathbb{N}$ we have that if $(\text{kk}_i, \text{ctxt}_i, \text{pk}_i, \text{aux}_i) \leftarrow \text{Encaps}(\text{pp}, \text{pk}_{i-1})$ and $(\text{kk}'_i, \text{sk}_i) \leftarrow \text{Decaps}(\text{pp}, \text{sk}_{i-1}, \text{ctxt}_i, \text{aux}_i)$ then $\Pr[\text{kk}_i = \text{kk}'_i] = 1 - \text{negl}(\lambda)$.

To establish IND-CCA security, we rely on intermediate notions called IND-CPA^+ and IND-CCA^- , which also appear in [AFMR26]. We remark that their definitions are stated for the setting where the underlying PKE scheme is key-homomorphic. This, however, is not necessary, and the security definitions can be stated in greater generality by considering the underlying UPKE scheme that is Fork-IND-CPA secure. We describe the three main security games we consider in Definition 13, and give them explicitly in Fig. 4.

Definition 13 (IND-CPA^+ , IND-CCA^- and IND-CCA security of UKEM [AFMR26]). Consider the games $\text{Game}_{\text{IND-CPA}^+}$, $\text{Game}_{\text{IND-CCA}^-}$, and $\text{Game}_{\text{IND-CCA}}$ in Fig. 4. All three games start with the root node (numbered 0), and build an experiment tree by constructing new nodes via oracle queries. Each node has an index i , a parent par_i (with $\text{par}_0 = \perp$), a revealed flag rev_i , a type typ_i (with $\text{typ}_0 = \perp$), and a key-pair $(\text{pk}_i, \text{sk}_i)$. A node is full, if it has a secret key, and a half-node if $\text{sk}_i = \perp$. Half-nodes only appear in $\text{Game}_{\text{IND-CCA}}$, the root node is always full (up to correctness), and all descendants of a half-node are themselves half-nodes.

The nodes are of two distinct types: **enc**-type nodes will correspond to an IND-CPA-style encapsulation oracle call followed by a compromise of an honestly updated key-pair, modelled by OEncaps , and **dec**-type nodes will correspond to maliciously inserted updates with a possible decapsulation oracle call, modelled by special oracles described below.

Each experiment is allowed (at most) one challenge query, which also generates an **enc**-type node. If the challenge oracle, OMChal , is never called, the advantage of any adversary is clearly 0. Thus from now on we assume every experiment has exactly one OMChal call. The adversary is also allowed to compromise any secret key via ORev , but automatically loses if the key is in the challenge set (computed by ChalSet), which would correspond to a trivial win.

The **dec**-type nodes are constructed as follows:

IND-CPA⁺. The adversary has access to a chosen update oracle OUpd . To the oracle it provides a target node i and (unencrypted) update material dup . The oracle then updates $(\text{pk}_i, \text{sk}_i)$ with dup and assigns them to a new child node of i of type **dec**.

IND-CCA⁻. The adversary has access to the chosen update oracle as in IND-CPA^+ , and a partial decapsulation oracle ODecaps^- . To this oracle it provides a target node i , a ciphertext ctxt , and a public key pk . Provided it was not queried on the challenge ciphertext ctxt^* or challenge key pk_{i^*} , the oracle tries to decapsulate the ciphertext $(\text{kk}, \text{sk}) \leftarrow \text{Decaps}(\text{pp}, \text{sk}_i, \text{ctxt}, \text{pk}_i, \text{pk})$. If decapsulation fails, the oracle returns a failure $\perp$, otherwise it creates a new **dec**-type child of i with the key-pair (pk, sk) , and returns kk . Notice only full nodes are created.

IND-CCA. The adversary has access to a decapsulation oracle ODecaps . Besides i , ctxt , and pk , the oracle also expects a member tag mt , that is a proof that pk is a valid update of pk_i . If mt fails to verify, the oracle immediately aborts with $\perp$. A half-node child of i is created with the public key pk . If i is a half-node itself, the oracle terminates, otherwise it tries to decapsulate $(\text{kk}, \text{sk}) \leftarrow \text{Decaps}(\text{pp}, \text{sk}_i, \text{ctxt}, \text{pk}_i, \text{pk})$. If decapsulation fails, the oracle again terminates with $\perp$, otherwise it upgrades the new node to a full node with the secret key sk , and returns kk .

2.4 Proofs of Knowledge

Definition 14 (Non-Interactive Zero-Knowledge Proof of Knowledge). A non-interactive zero-knowledge proof of knowledge for an NP relation $\mathcal{R}$ with a random oracle H is a collection of five algorithms:

Setup. Outputs public parameters $\text{pp} \leftarrow \text{Setup}^{\text{H}}(1^\lambda)$.

Prove. On input the public parameters pp and an instance-witness pair $(x, w) \in \mathcal{R}$, outputs a proof $\pi \leftarrow \text{Prove}^{\text{H}}(\text{pp}, x, w)$.

Verify. On input the public parameters pp , an instance x and a proof π , outputs a bit $b \leftarrow \text{Vf}^{\text{H}}(\text{pp}, x, \pi)$ that verifies there is a w such that $(x, w) \in \mathcal{R}$.

Game^A_{IND-CPA⁺, IND-CCA⁻, IND-CCA}(λ)	OEncaps(i)
$pp \leftarrow \text{Setup}(1^\lambda); (pk_0, sk_0) \leftarrow \text{KGen}(pp)$ $(par_0, rev_0, j, typ_0) := (\perp, 0, 0, \perp)$ $(i^*, ctxt^*) := (\perp, \perp); b \leftarrow \mathbb{S}\{0, 1\}$ // if IND-CPA ⁺ : $b' \leftarrow \mathcal{A}^{\text{OEncaps, ORev, OMChal, OUpd}}(pp, pk_0)$ // if IND-CCA ⁻ : $b' \leftarrow \mathcal{A}^{\text{OEncaps, ORev, OMChal, OUpd, ODecaps}^-}(pp, pk_0)$ // if IND-CCA: $b' \leftarrow \mathcal{A}^{\text{OEncaps, ORev, OMChal, ODecaps}}(pp, pk_0)$ $S := \text{ChalSet}(i^*)$ return $b = b'$ and $\forall j \in S: rev_j = 0$	$(kk, ctxt, up/\emptyset/mt) \leftarrow \text{HonestNode}(i)$ return $(kk, ctxt, pk_j, up/\emptyset/mt)$
HonestNode(i)	ORev(i)
assert $pk_i \neq \perp; j := j + 1$ $(kk, ctxt, pk_j, up/\emptyset/mt) \leftarrow \text{Encaps}(pp, pk_i)$ $(par_j, rev_j, typ_j) := (i, 0, \text{enc})$ if $sk_i \neq \perp$: $(\_, sk_j) \leftarrow \text{Decaps}(pp, sk_i, ctxt, pk_i, up/pk_j)$ else : $sk_j := \perp$ return $(kk, ctxt, up/\emptyset/mt)$	assert $sk_i \neq \perp; rev_i := 1$; return sk_i
OUpd(i, dup) (for IND-CPA ⁺ and IND-CCA ⁻)	OMChal(i)
assert dup valid and $pk_i \neq \perp; (pk, \_) \leftarrow \text{UPKE.UpdPk}(pk_i; \text{dup}); j := j + 1$ $sk \leftarrow \text{UPKE.UpdSk}(sk_i; \text{dup}); (pk_j, sk_j, par_j, rev_j, typ_j) := (pk, sk, i, 0, \text{dec});$ return $\perp$	assert $i^* = \perp; i^* := i$ $kk_0 \leftarrow \mathbb{S}\{0, 1\}^\lambda$ $(kk_1, ctxt^*, up/\emptyset/mt) \leftarrow \text{HonestNode}(i)$ return $(kk_b, ctxt^*, pk_j, up/\emptyset/mt)$
ODecaps⁻($i, ctxt, pk$) (for IND-CCA ⁻)	ChalSet(i^*)
assert $sk_i \neq \perp$; if $i^* \neq \perp$: assert $ctxt^* \neq ctxt$ or $pk_{i^*} \neq pk_i$ $(kk, sk) \leftarrow \text{Decaps}(pp, sk_i, ctxt, pk_i, pk)$; if $kk = \perp$: return $\perp$ $j := j + 1; (pk_j, sk_j, par_j, rev_j, typ_j) := (pk, sk, i, 0, \text{dec});$ return kk	if $i^* = \perp$: return $\emptyset$ $\text{base} :=$ node indices on direct path from node 0 to i^* $\text{ext\_base} := \{i' \mid \exists i \in \text{base}: pk_{i'} = pk_i\}$ $\text{dec\_cl} :=$ nodes reachable from some $i \in \text{ext\_base}$ via only dec -type nodes return dec_cl
ODecaps($i, ctxt, pk, mt$) (for IND-CCA)	
assert $pk_i \neq \perp$; if $i^* \neq \perp$: assert $ctxt^* \neq ctxt$ or $pk_{i^*} \neq pk_i$; assert $\forall f(pp, pk_i, pk, mt)$ $j := j + 1; (pk_j, sk_j, par_j, rev_j, typ_j) := (pk, \perp, i, 0, \text{dec});$ if $sk_i = \perp$: return $\perp$ $(kk, sk_j) \leftarrow \text{Decaps}(pp, sk_i, ctxt, pk_i, pk)$; if $kk \neq \perp$: return kk	

Figure 4: The IND-CPA⁺, IND-CCA⁻, and IND-CCA security games for various UKEM constructions. For IND-CPA⁺ the last output of OEncaps and OMChal is the update token up, for IND-CCA⁻ it is omitted and for IND-CCA the output is a member tag mt.

Simulate. On input the public parameters $\mathbf{pp}$ and an instance x , outputs a simulated proof $\pi \leftarrow \text{Sim}^H(\mathbf{pp}, x)$. This algorithm may program H .

Extract. On input the public parameters $\mathbf{pp}$, an instance x , and a proof π , outputs a witness $w \leftarrow \text{Ext}^H(\mathbf{pp}, x, \pi)$. This algorithm may program H . The extractor is straight-line (or online) if it does not rewind the prover.

Consider $\text{Game}_{\text{Compl}}$, Game_{SI} , Game_{ZK} , and Game_{Ext} from Fig. 5, respectively. We say that a NIZKPoK is complete, setup-indistinguishable, zero-knowledge, and straight-line simulation-extractable if the advantage of any adversary in winning $\text{Game}_{\text{Compl}}$, Game_{SI} , Game_{ZK} , and Game_{Ext} , respectively, is negligible.⁹

2.5 Proving Isometries

The standard way to prove two codes/lattices are isometric is with a Graph Isomorphism-style Σ -protocol (see e.g. [PS23] for codes and [DvW22] for lattices). For background on Σ -protocols, consult e.g. [Dam10]. Namely, to prove $\mathbf{C}_1 = \text{SF}(\mathbf{O} \cdot \mathbf{C}_0)$, the prover samples a new isometry $\mathbf{O}' \leftarrow \$\mathcal{O}_n(\mathbb{Z})$, and commits to it by sending $\mathbf{C} = \text{SF}(\mathbf{O}' \cdot \mathbf{C}_0)$ to the verifier. On input a challenge bit $b \in \{0, 1\}$, the prover then responds by showing an isometry from $\mathbf{C}_b$ to $\mathbf{C}$, that is it sends $\mathbf{O}'$ if $b = 0$ and $\mathbf{O}' \cdot \mathbf{O}^{-1}$ if $b = 1$ to the verifier. It is immediate that this protocol is complete, honest-verifier zero-knowledge, and has *knowledge error*, i.e. the probability of a cheating prover convincing the verifier is $\frac{1}{2}$. With λ parallel repetitions, this is amplified to $2^{-\lambda}$. There is a rewinding Ext that extracts $\mathbf{O}$ from two accepting transcripts that start with the same commitment but differ in the challenge bit, that is $\mathbf{O}$ can be easily computed from $\mathbf{O}'$ and $\mathbf{O}' \cdot \mathbf{O}^{-1}$ (a property called 2-special soundness). The basic interactive construction is presented in Fig. 6.

The standard way to make a Σ -protocol non-interactive is the Fiat-Shamir (FS) transform [FS87], where the challenge is derived from the commitment via a random oracle. However, this NIZKPoK has a rewinding Ext and is thus not straight-line simulation-extractable, which is required of NIZKPoKs used for member security in the model in Fig. 4. Several works [Pas03, Fis05, Unr15, Kat21] explored alternatives enabling straight-line simulation-extractability. All share the idea of modifying the FS transform so the prover must query the random oracle multiple times for the same commitment to produce a valid challenge chal and response resp . The extractor Ext then finds such a pair directly from the prover's oracle queries, avoiding rewinding. Fischlin's transform from [Fis05] uses a proof-of-work strategy, requiring that the hash of the response starts with a certain amount of 0's. For appropriate parameters this guarantees the list of random oracle queries contains two accepting transcripts from which we can extract the witness without rewinding the prover.¹⁰

Definition 15 (Fischlin's transform [Fis05]). Let (b, r, S, t) be integers denoting the number of proof-of-work bits, the number of repetitions, the maximal allowed sum, and the number of challenge bits, respectively. The parameters are valid if $S = \mathcal{O}(r)$, $b \cdot r = \omega(\log(\lambda))$, $2^{t-b} = \omega(\log(\lambda))$, and $b, r, t = \mathcal{O}(\log(\lambda))$ with $b \leq t$. Let (P_1, V_1, P_2, V_2) be a Σ -protocol. Consider the transform in Fig. 7 that, on input (Σ, b, r, S, t) , outputs a NIZKPoK with a straight-line extractor.

Remark 1. The parameter S represents the tolerance of the proof-of-work mechanism. We set it to 0 to simplify the analysis and minimise the soundness error per repetition [CL24]. The original transform by Fischlin included the transcript with the smallest hash value in the proof instead of aborting if proof-of-work was not satisfied, hence the sum check by the verifier.

The authors of [CL24] explored various parameter regimes and showed that the completeness error when $S = 0$ is $2^{-(2^{t-b} \cdot \log(e) - \log(r))}$, and the soundness error is $2^{-b \cdot r}$. This gives them a useful rule of thumb that

⁹ Straight-line simulation-extractability supersedes adaptive simulation-soundness.

¹⁰ We use Fischlin's transform [Fis05] instead of Unruh's transform [Unr15] because the former yields proofs of slightly smaller (expected) size. The latter also works in the QROM at a modest cost to proof size, but we will not need this property in our proofs later.

$\text{Game}_{\text{Compl}}^{\mathcal{A}}(\lambda)$	
$\text{pp} \leftarrow \text{Setup}^{\text{H}}(1^\lambda); (x, w) \leftarrow \mathcal{A}^{\text{H}}(\text{pp}); \pi \leftarrow \text{Prove}^{\text{H}}(\text{pp}, x, w)$ $b \leftarrow \text{Vf}^{\text{H}}(\text{pp}, x, \pi); \text{ return } (x, w) \in \mathcal{R} \text{ and } b = 0$	
.....	
$\text{Game}_{\text{SI}}^{\mathcal{A}}(\lambda)$	$\text{P}(a, b)$
$\text{pp}_0 \leftarrow \text{Setup}^{\text{H}}(1^\lambda); (\text{pp}_1, \text{td}) \leftarrow \text{SetupAlt}^{\text{H}, \text{P}}(1^\lambda)$ $b \leftarrow \$ \{0, 1\}; b' \leftarrow \mathcal{A}^{\text{H}}(\text{pp}_b); \text{ return } b' = b$	$\text{assert } \text{H}[a] \in \{\perp, b\}$ $\text{H}[a] := b$
.....	
$\text{Game}_{\text{ZK}}^{\mathcal{A}}(\lambda)$	$\text{OProve}(x, w)$
$b \leftarrow \$ \{0, 1\}$ $\pi_{\text{Sim}}[\cdot] := \emptyset$ $\text{pp} \leftarrow \text{Setup}^{\text{H}}(1^\lambda)$ $b' \leftarrow \mathcal{A}^{\text{H}, \text{OProve}}(\text{pp})$ $\text{ return } b' = b$	$\text{assert } (x, w) \in \mathcal{R}$ $\text{ if } \pi_{\text{Sim}}[x] = \perp :$ $\quad \text{ if } b = 0: \pi_{\text{Sim}}[x] \leftarrow \text{Prove}^{\text{H}}(\text{pp}, x, w)$ $\quad \text{ if } b = 1: \pi_{\text{Sim}}[x] \leftarrow \text{Sim}^{\text{H}, \text{P}}(\text{pp}, x)$ $\text{ return } \pi_{\text{Sim}}[x]$
.....	
$\text{Game}_{\text{Ext}}^{\mathcal{A}}(\lambda)$	
$\text{win} := 0; \pi_{\text{Sim}}[\cdot] := \emptyset; \text{pp} \leftarrow \text{Setup}^{\text{H}}(1^\lambda); \_ \leftarrow \mathcal{A}^{\text{H}, \text{OSim}, \text{OExt}}(\text{pp}); \text{ return win}$	
$\text{OExt}(x, \pi)$	$\text{OSim}(x)$
$\text{assert } \text{Vf}^{\text{H}}(\text{pp}, x, \pi); w \leftarrow \text{Ext}^{\text{H}, \text{P}}(\text{pp}, x, \pi)$ $\text{ if } \pi_{\text{Sim}}[x] = \perp \text{ and } (x, w) \notin \mathcal{R}: \text{ win} := 1$ $\text{ return } w$	$\text{ if } \pi_{\text{Sim}}[x] = \perp :$ $\quad \pi_{\text{Sim}}[x] \leftarrow \text{Sim}^{\text{H}, \text{P}}(\text{pp}, x)$ $\text{ return } \pi_{\text{Sim}}[x]$

Figure 5: NIZKPoK completeness game with $\text{Adv}_{\text{Compl}}(\mathcal{A}) = \Pr[1 \leftarrow \text{Game}_{\text{Compl}}^{\mathcal{A}}(\lambda)]$, setup indistinguishability game for the trapdoor alternative setup SetupAlt with $\text{Adv}_{\text{SI}}(\mathcal{A}) = \Pr[1 \leftarrow \text{Game}_{\text{SI}}^{\mathcal{A}}(\lambda)] - \frac{1}{2}$, zero-knowledge game with $\text{Adv}_{\text{ZK}}(\mathcal{A}) = \left| \Pr[1 \leftarrow \text{Game}_{\text{ZK}}^{\mathcal{A}}(\lambda)] - \frac{1}{2} \right|$, and straight-line simulation-extractability game with $\text{Adv}_{\text{Ext}}(\mathcal{A}) = \Pr[1 \leftarrow \text{Game}_{\text{Ext}}^{\mathcal{A}}(\lambda)]$.

$P_1((C_0, C_1), O)$	$P_2(st := O', (C_0, C_1), O, chal)$
$O' \leftarrow \$ \mathcal{O}_n(\mathbb{Z})$	if $chal = 0$: return $resp := O'$
$C, \_ := SF(O' \cdot C_0)$	else : return $resp := O' \cdot O^{-1}$
return $(com := C, st := O')$	$V_2((C_0, C_1), com, chal, resp)$
$V_1((C_0, C_1), com)$	parse $(C, b, \tilde{O}) \leftarrow (com, chal, resp)$
return $chal \leftarrow \$ \{0, 1\}$	return $C = SF(\tilde{O} \cdot C_b)$

Figure 6: One iteration of the standard isometry Σ -protocol (P_1, V_1, P_2, V_2) for proving knowledge of $w = O$ such that $C_1 = SF(O \cdot C_0)$ for a pair of code generator matrices $x = (C_0, C_1)$.

$P_{\text{Fis}}^H(x, w)$
for $i = 1, \dots, r$ do $(com_i, st_i) \leftarrow P_1(x, w)$ endfor ; $com := [com_1, \dots, com_r]$; $\pi := [\cdot]$
for $i = 1, \dots, r$ do for $j = 0, \dots, 2^t - 1$ do
$chal_i := j$ as a t -bit string; $resp_i \leftarrow P_2(st_i, x, w, chal_i)$
if $H(x, com, i, chal_i, resp_i) = 0^b$: $\pi[i] := (com_i, chal_i, resp_i)$ and break j -loop
if $\exists i: \pi[i] = \perp$: return $\perp$ else : return π
$V_{\text{Fis}}^H(x, \pi)$
for $i = 1, \dots, r$ if $V_2(x, com_i, chal_i, resp_i) = 0$: return 0
if $\sum_{i=1}^r H(x, com, i, chal_i, resp_i) > S$: return 0 else : return 1

Figure 7: Fischlin's transform [Fis05] with parameters $(b, r, S = 0, t)$ and a random oracle $H: \{0, 1\}^* \rightarrow \{0, 1\}^b$ for a Σ -protocol with challenges of t bits.

$t = b + 5$ when $r \leq 64$, and $t = b + 6$ when $r > 64$ for any reasonable r , to achieve a completeness error of about 2^{-40} (upon failure the protocol can just be repeated with fresh randomness). We want $b \cdot r = \lambda$, and will optimise the parameters for proof size.

The proof size is $|\pi| = \sum_{i=1}^r (|com_i| + t + |resp_i|)$. First we have $\sum_{i=1}^r |com_i| = \lambda$ by sending a hash of the list $[com_1, \dots, com_r]$ instead of the com_i themselves, which the verifier also computes. If the challenge bit is 0, then $|resp| = \lambda$, as we only need to send the seed used to sample the random isometry O' . If the challenge bit is 1, we instead need to send $O^{-1} \cdot O'$ where O is the witness, for which we do not know a seed, hence $C = |resp| = n + n \cdot \log(n)$, the bit-size of a signed permutation. We have

$$\mathbb{E}[|\pi|] = \lambda + r \cdot t + \frac{r \cdot t}{2} \cdot \lambda + \frac{r \cdot t}{2} \cdot C.$$

While reducing the number of rounds r noticeably reduces the proof size, this is offset by a significant increase in prover work. Since each proof essentially consists of computing at most $r \cdot 2^t = r \cdot 2^{b+5}$ single-bit proofs and then picking $r \cdot t$ of them based on proof-of-work, and $r \cdot b = \lambda$ is constant for a fixed λ , we get that halving r doubles b and thus multiplies the number of single-bit proof components that the prover is expected to compute to satisfy proof-of-work by 2^b per round. In the Σ -protocol in Fig. 6, computing com costs $\mathcal{O}(n^\omega)$, and computing $resp$ costs $\mathcal{O}(n)$ if $chal = 1$ (since we are multiplying permutations) and $\mathcal{O}(1)$ if $chal = 0$, thus expected prover work – assuming hashing costs $\mathcal{O}(1)$ – is

$$\mathbb{E}[\text{num. ops.}] = r \cdot 2^t \cdot \mathcal{O}(n^\omega) + \frac{r}{2} \cdot 2^t \cdot 2^b \cdot \mathcal{O}(n).$$

The NIZKPoK properties in Definition 14 are guaranteed by [Fis05, Thm. 2].

Fischlin’s transform requires the underlying Σ -protocol satisfies an additional property called *quasi-unique responses*: for any PPT algorithm $\mathcal{A}$ we have

$$\Pr \left[\begin{array}{l} \text{resp} \neq \text{resp}' \wedge V_2(x, \text{com}, \text{chal}, \text{resp}) \\ \wedge V_2(x, \text{com}, \text{chal}, \text{resp}') \end{array} \middle| (x, \text{com}, \text{chal}, \text{resp}, \text{resp}') \leftarrow \mathcal{A} \right] \leq \text{negl}(\lambda).$$

In short, a prover that can produce multiple responses for the same commitment and challenge has a higher probability of satisfying proof-of-work without querying the random oracle on two transcripts with different challenges, which we need to extract the witness. This is a rather strong property that is not satisfied for many commonly used Σ -protocols, including the protocol in Fig. 6 and our modification of it in Section 4 – for any accepted response $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$ we have that $-\mathbf{O}$ is also accepted. This can be circumvented by considering a stronger notion of soundness [Ks22]. However, for Σ -protocols based on code equivalence, the standard approach [Bor23, FPST26] is to bound from above the number of distinct responses an adversary can produce and then increase b to account for its advantage – in the above example, $-\mathbf{O}$ is often the only other accepted response, so incrementing b suffices. We discuss this more concretely in Section 4.4.

2.6 Splitting Behaviour of Rational Primes in Cyclotomic Rings

Denote by $\phi(\cdot)$ Euler’s totient function and by Φ_f the f -th cyclotomic polynomial. An integer a is called a *primitive root* modulo n , if a generates the multiplicative group $\mathbb{Z}_n^*$, or equivalently if the multiplicative order of a in $\mathbb{Z}_n$ is $\phi(n)$. Let $\mathcal{K} \cong \mathbb{Q}[X]/\Phi_f(X)$ be the cyclotomic field of conductor $f = p^r$ for a rational prime p , and denote by $\mathcal{R}$ its ring of integers.

Proposition 2 ([Was82, Thm. 2.13]). *Let q be a rational prime coprime to f with multiplicative order m in $\mathbb{Z}_f^*$. Then $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ is isomorphic (via the CRT decomposition) to a product of $\frac{\phi(f)}{m}$ isomorphic copies of $\mathbb{F}_{q^m}$.*

In particular, $\mathcal{R}_q$ is a field if and only if q is a primitive root modulo f , and moreover such *inert* rational primes q always exist, since primitive roots exist if and only if $f \in \{1, 2, 4, p^r, 2p^r \mid r \in \mathbb{N}\}$ [Vin54, VI.2].

3 Hollow Decision SIS

In this section, we introduce a new hollow lattice problem, a computational variant of the Hollow LHL distinguishing problem (Lemma 2) introduced in [ABL25a]. We refer to it as the *Hollow Decision SIS* problem, to distinguish it from the Hollow LWE problem (Definition 5) introduced in [ABL25a, Sec. 4] and proven computationally hard under standard LWE [ABL25a, Thm. 1]. Recall that the Hollow LWE asks LWE to be hard when multiplying a tall hollow matrix by a secret vector from the right. Here we are asking for an LWE-like problem to be hard when multiplying a tall hollow matrix by a short secret vector from the left.

Definition 16 (Hollow DSIS). *Let $n, k, h, q \in \mathbb{Z}$ with $0 \leq h \leq k \leq n$, q prime, let $\mathcal{D}_{n,k,q}^h$ denote the distribution of uniformly random h -hollow matrices in $\mathbb{F}_q^{n \times k}$, and let $\mathcal{D}$ be a secret distribution. The advantage of any PPT adversary $\mathcal{A}$ against $\text{DSIS}_{k,n,q}^h(\mathcal{D})$ is defined as*

$$\text{Adv}_{\text{DSIS}^h}(\mathcal{A}) = \left| \Pr \left[b = b' \middle| \begin{array}{l} \mathbf{A} \leftarrow \mathcal{D}_{n,k,q}^h; \mathbf{r} \leftarrow \mathcal{D}; b \leftarrow \{0, 1\}; \mathbf{u}_0 \leftarrow \mathbb{F}_q^k \\ \mathbf{u}_1^\top := \mathbf{r}^\top \cdot \mathbf{A} \bmod q; b' \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{u}_b) \end{array} \right] - \frac{1}{2} \right|.$$

We show Hollow DSIS is nothing but Hollow LWE on the matrix generating the dual code. Thus for rate $\frac{1}{2}$ the two problems are the same.¹¹

¹¹ We would like to thank an anonymous reviewer at AC’26 for suggesting the following reduction. The previous version of this manuscript used Appendix A to obtain worse parameters and only conjectured that Hollow DSIS is hard for rate $\frac{1}{2}$.

Theorem 1 ($\text{LWE}_{k,n,q,\eta}^h = \text{DSIS}_{n-k,n,q}^h(\eta^n)$). *Let $n, k, h, q \in \mathbb{Z}$ with $0 \leq h \leq k \leq n - h$, q prime, and η a distribution over $\mathbb{Z}$. If there exists a (t, ε) -algorithm against $\text{LWE}_{k,n,q,\eta}^h$ then there exists a $(t + \mathcal{O}(n^\omega), \varepsilon)$ -algorithm against $\text{DSIS}_{n-k,n,q}^h(\eta^n)$, and vice versa.*

Proof. We give an explicit reduction in both directions.

Hollow LWE $\rightarrow$ Hollow DSIS. Let $(\mathbf{A}, \mathbf{b})$ be a $\text{LWE}_{k,n,q,\eta}^h$ sample with $\mathbf{A} \leftarrow \mathcal{D}_{n,k,q}^h$ and either $\mathbf{b} \leftarrow \mathbb{F}_q^n$ if $b = 0$ or $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ for $\mathbf{s} \leftarrow \mathbb{F}_q^k$ and $\mathbf{e} \leftarrow \eta^n$ if $b = 1$. The reduction computes a uniformly random generator $\mathbf{A}^\perp \in \mathbb{F}_q^{n \times (n-k)}$ of the code $\text{Span}(\mathbf{A})^\perp$ and then the vector $\mathbf{u}^\top = \mathbf{b}^\top \cdot \mathbf{A}^\perp$. It sends the pair $(\mathbf{A}^\perp, \mathbf{u})$ to its distinguisher $\mathcal{A}$ for $\text{DSIS}_{n-k,n,q}^h(\eta^n)$ and outputs its bit. Since $[n, k]$ -linear h -hollow codes over $\mathbb{F}_q$ are in bijective correspondence with $[n, n-k]$ -linear h -hollow codes over $\mathbb{F}_q$ via the map $\mathcal{C} \mapsto \mathcal{C}^\perp$ which preserves the hull, we have that $\mathbf{A}^\perp$ is distributed according to $\mathcal{D}_{n,n-k,q}^h$. If $b = 0$ and $\mathbf{b}$ is uniform then so is $\mathbf{u}$. Otherwise

$$\mathbf{u}^\top = \mathbf{s}^\top \cdot \mathbf{A}^\top \cdot \mathbf{A}^\perp + \mathbf{e}^\top \cdot \mathbf{A}^\perp = \mathbf{e}^\top \cdot \mathbf{A}^\perp,$$

hence $\mathbf{u}$ is correctly distributed for the secret distribution $\mathbf{D} = \eta^n$. The reduction clearly succeeds if and only if $\mathcal{A}$ succeeds, and its computational overhead amounts to linear algebra.

Hollow DSIS $\rightarrow$ Hollow LWE. Let $(\mathbf{A}, \mathbf{u})$ be a $\text{DSIS}_{n-k,n,q}^h(\eta^n)$ sample with $\mathbf{A} \leftarrow \mathcal{D}_{n,n-k,q}^h$ and either $\mathbf{u}^\top \leftarrow \mathbb{F}_q^{n-k}$ if $b = 0$ or $\mathbf{u}^\top = \mathbf{r}^\top \cdot \mathbf{A}$ for $\mathbf{r} \leftarrow \eta^n$ if $b = 1$. The reduction computes a uniformly random generator $\mathbf{A}^\perp \in \mathbb{F}_q^{n \times k}$ of the code $\text{Span}(\mathbf{A})^\perp$ and then a uniformly random vector $\mathbf{y}$ such that $\mathbf{y}^\top \cdot \mathbf{A} = \mathbf{u}^\top$ (that is, any solution plus a random element of $\ker(\mathbf{A}^\top) = \text{Span}(\mathbf{A}^\perp)$). $\mathbf{A}^\perp$ is correctly distributed by the argument in the previous direction. If $b = 0$ and $\mathbf{u}$ is uniform, then so is $\mathbf{y}$. Otherwise $\mathbf{y}$ is distributed identically to the sum $\mathbf{x} + \mathbf{r}$ where $\mathbf{x}$ is a uniformly random element in $\text{Span}(\mathbf{A})^\perp$, that is, $\mathbf{y} = \mathbf{A}^\perp \cdot \mathbf{s} + \mathbf{r}$ for $\mathbf{s} \leftarrow \mathbb{F}_q^k$, hence $(\mathbf{A}^\perp, \mathbf{y})$ is distributed as a $\text{LWE}_{k,n,q,\eta}^h$ sample. $\square$

Let $\mathbf{D}_\times = \mathcal{U}(\{\pm 1\})$; the reduction from [ABL25a, Thm. 1] then implies that signed-error LWE reduces to signed-secret Hollow DSIS that we require for our scheme.

Corollary 1 ($\text{LWE}_{k-h,n,q,\mathbf{D}_\times} \rightarrow \text{DSIS}_{n-k,n,q}^h(\mathbf{D}_\times^n)$). *Let $n, k, h, q \in \mathbb{Z}$ with $0 \leq h \leq k \leq n - h$ and q odd prime. If there exists a (t, ε) -algorithm against $\text{LWE}_{k-h,n,q,\mathbf{D}_\times}$ then there exists a $(t + \text{poly}(k, q), \varepsilon \cdot (1 - 1/q - 1/q^2) \cdot (1 - \text{negl}(k)))$ -algorithm against $\text{DSIS}_{n-k,n,q}^h(\mathbf{D}_\times^n)$.*

We give a correctness and security statement for the signed-secret dual-Regev PKE for the parameters described in Theorem 1 and extend Proposition 1.

Proposition 3. *Let $k \geq \Omega(\lambda)$ and $\varepsilon \leq \text{negl}(\lambda)$. If $\eta = \mathcal{D}_{\mathbb{Z},s}$ is the discrete Gaussian distribution over $\mathbb{Z}$ with parameter $s \geq \sqrt{2}\eta_\epsilon(\mathbb{Z})$ for $\eta_\epsilon(\mathbb{Z})$ the smoothing parameter [MR04] of $\mathbb{Z}$, so that the convolution theorem of [MP13a, MP13b, Thm. 3.3] applies, and $q > p^2/2 + p \cdot s \cdot \sqrt{(\lambda + 1) \cdot (n + 1)}$, then the signed-secret dual-Regev PKE from Definition 3 is correct with overwhelming probability in λ . If q is prime and $n = 2k$, and if there exists a PPT algorithm $\mathcal{A}$ against the IND-CPA security of the signed-secret dual-Regev PKE with h -hollow matrices that makes q_{Enc} calls to the encryption (challenge) oracle, then there exist PPT algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ against $\text{DSIS}_{k,n,q}^h$ and $\text{LWE}_{k,n,q,\eta}^h$, respectively, such that*

$$\text{Adv}_{\text{IND-CPA}}^{\text{PKE}}(\mathcal{A}) \leq \text{Adv}_{\text{DSIS}^h}(\mathcal{B}_1) + q_{\text{Enc}} \cdot \text{Adv}_{\text{LWE}^h}(\mathcal{B}_2).$$

Proof (Sketch). The proof is standard. First replace the public key with a random pair using Hollow DSIS, then replace the ciphertexts using Hollow LWE.

Corollary 2 (update of [ABL25a, Thm. 3], assuming DSIS). *Let $n, k, h, q, \chi \in \mathbb{Z}$ be parametrised by λ with $n = 2k$, $2h \leq k$, $\chi \in \{\pm 1\}$ and q as in Proposition 3. If the advantage of any PPT adversary against $\text{LWE}_{k,n,q,\eta}^h$, $\text{DSIS}_{k,n,q}^h$ and $\Delta\text{PCE}_{n,k,q}^{\chi,h}$ is negligible in λ , the UPKE scheme from Fig. 3 is IND-CR-CPA secure. Concretely, for any PPT algorithm $\mathcal{A}$ against the IND-CR-CPA game from Fig. 2 for UPKE from Fig. 3 with the above parameters, there are PPT algorithms $\mathcal{B}_1, \mathcal{B}_2$ such that*

$$\text{Adv}_{\text{IND-CR-CPA}}^{\text{UPKE}}(\mathcal{A}) \leq 2\text{Adv}_{\text{IND-CPA}}^{\text{PKE}}(\mathcal{B}_1) + \text{Adv}_{\Delta\text{PCE}}(\mathcal{B}_2).$$

Proof (Sketch). The proof is exactly the same as that of [ABL25a, Thm. 3] (Proposition 1), replacing Hollow LHL with Hollow DSIS and using Proposition 3. We do not pursue this further as we prove a stronger result in Section 5.

From now on, let UPKE denote the scheme from [ABL25a], presented in Fig. 3 with the above updated parameters in Corollary 2.

4 Proving Isometries of Codes and Syndromes

We construct *member tags* for the UPKE construction in Fig. 3; these are proofs that the new public key is indeed a valid update of the old one, guaranteeing that a malicious sender has not tried to get a public key adopted as a properly updated public key for which they know the secret key. The corresponding security notion is called *member security* [AFM24] and is part of IND-CCA security (Section 5). We formalise member tags in Section 4.1 under the condition that the UPKE scheme is *CU-friendly*, which is satisfied by Fig. 3 and, to the best of our knowledge, by all existing UPKE schemes.

4.1 Formalising Member Tags

Denote by pk and pk' two UPKE public keys where pk' was obtained from pk by updating. Any party holding the secret key sk corresponding to pk can update their secret key to sk' and check that it indeed corresponds to pk' , making the new key-pair valid. However, parties without sk may also want to check that pk' is indeed a valid update of pk , otherwise a malicious user $\mathcal{A}$ could try to convince them that a fresh public key $\tilde{\text{pk}}$, for which $\mathcal{A}$ knows the secret key $\tilde{\text{sk}}$, is the key of the next epoch. This is mainly an issue if all holders of sk are offline and thus cannot alert other parties to such behaviour, while another party would like to adopt pk' so that it can send another message to the same receiver.

Notice, however, that the witness to the update $\text{pk} \rightarrow \text{pk}'$ need not be a seed for the update (like in the chosen randomness security game in Fig. 2), but rather valid update material not necessarily generated using a seed (like in the chosen update security game in [DKW22, Sec. 7.5]), for example $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$ in the case of Fig. 3. In order to define the update relation for which we construct a NIZKPoK, we first define UPKE schemes that admit it.

Definition 17 (CU-friendly UPKE). Let $\mathcal{UP}$ be the set of update material, $\mathcal{PK}, \mathcal{SK}$ the sets of public and secret keys for a PKE scheme, all parametrised by λ , and $H: \{0, 1\}^\lambda \rightarrow \mathcal{UP}$ a random oracle. Define the following pairs of algorithms.

CU-update mechanism. Deterministic algorithms $(\text{UpdPk}_{\text{CU}}, \text{UpdSk}_{\text{CU}})$, where $\text{pk}' := \text{UpdPk}_{\text{CU}}(\text{pk}, \text{dup})$ on input $\text{pk} \in \mathcal{PK}$ and $\text{dup} \in \mathcal{UP}$ outputs $\text{pk}' \in \mathcal{PK}$, and $\text{sk}' := \text{UpdSk}_{\text{CU}}(\text{sk}, \text{dup})$ on input $\text{sk} \in \mathcal{SK}$ and $\text{dup} \in \mathcal{UP}$ outputs $\text{sk}' \in \mathcal{SK}$ or aborts with $\perp$. This mechanism is correct if for every $\text{dup} \in \mathcal{UP}$ and valid key-pair $(\text{pk}, \text{sk}) \in \mathcal{PK} \times \mathcal{SK}$ we have that $(\text{UpdPk}_{\text{CU}}(\text{pk}, \text{dup}), \text{UpdSk}_{\text{CU}}(\text{sk}, \text{dup}))$ is a valid key-pair.

CR-update mechanism. Deterministic algorithms $(\text{UpdPk}_{\text{CR}}, \text{UpdSk}_{\text{CR}})$, where $\text{pk}' := \text{UpdPk}_{\text{CR}}(\text{pk}, \rho)$ on input $\text{pk} \in \mathcal{PK}$ and $\rho \in \{0, 1\}^\lambda$ returns $\text{pk}' := \text{UpdPk}_{\text{CU}}(\text{pk}, H(\rho))$, and $\text{sk}' := \text{UpdSk}_{\text{CR}}(\text{sk}, \rho)$ on input $\text{sk} \in \mathcal{SK}$ and $\rho \in \{0, 1\}^\lambda$ returns $\text{sk}' := \text{UpdSk}_{\text{CU}}(\text{sk}, H(\rho))$.

CU-friendly update mechanism. Probabilistic algorithms $(\text{UpdPk}, \text{UpdSk})$, where $(\text{pk}', \text{up}) \leftarrow \text{UpdPk}(\text{pk})$ on input $\text{pk} \in \mathcal{PK}$ samples $\rho \leftarrow \$ \{0, 1\}^\lambda$, runs $\text{pk}' := \text{UpdPk}_{\text{CR}}(\text{pk}, \rho)$, computes $\text{up} \leftarrow \text{PKE.Enc}(\text{pk}, \rho)$ and returns (pk', up) . Then $\text{sk}' \leftarrow \text{UpdSk}(\text{sk}, \text{up})$ on input $\text{sk} \in \mathcal{SK}$ and a PKE ciphertext up runs $\rho \leftarrow \text{PKE.Dec}(\text{sk}, \text{up})$ and aborts if it fails, then computes $\text{sk}' := \text{UpdSk}_{\text{CR}}(\text{sk}, \rho)$ and returns sk' .

A UPKE scheme over PKE is CU-friendly, if there exist deterministic algorithms $(\text{UpdPk}_{\text{CU}}, \text{UpdSk}_{\text{CU}})$ such that $(\text{UPKE.UpdPk}, \text{UPKE.UpdSk})$ is a CU-friendly update mechanism derived from $(\text{UpdPk}_{\text{CU}}, \text{UpdSk}_{\text{CU}})$.

We note that all UPKE schemes we are aware of are CU-friendly.

$P_1((\mathbf{C}_0, \mathbf{S}_0, \mathbf{C}_1, \mathbf{S}_1), (\mathbf{O}, \mathbf{P}))$	$P_2(\text{st} := \mathbf{O}', (\mathbf{C}_0, \mathbf{S}_0, \mathbf{C}_1, \mathbf{S}_1), \mathbf{O}, \text{chal})$
$\mathbf{O}_0 \leftarrow_{\$} \mathcal{O}_n(\mathbb{Z}); \mathbf{O}_1 \leftarrow_{\$} \mathcal{O}_\ell(\mathbb{Z})$	if $\text{chal} = 0$: return $\text{resp} := \mathbf{O}'$
$\mathbf{O}' := \text{diag}(\mathbf{O}_0, \mathbf{O}_1)$	else : return $\text{resp} := \mathbf{O}' \cdot \text{diag}(\mathbf{O}, \mathbf{P})^{-1}$
$[\mathbf{C} \parallel \mathbf{S}], \_ := \text{SF}(\mathbf{O}' \cdot [\mathbf{C}_0 \parallel \mathbf{S}_0])$	$V_2((\mathbf{C}_0, \mathbf{S}_0, \mathbf{C}_1, \mathbf{S}_1), \text{com}, \text{chal}, \text{resp})$
return $(\text{com} := [\mathbf{C} \parallel \mathbf{S}], \text{st} := \mathbf{O}')$	parse $(\tilde{\mathbf{C}}, b, \tilde{\mathbf{O}}) \leftarrow (\text{com}, \text{chal}, \text{resp})$
$V_1((\mathbf{C}_0, \mathbf{S}_0, \mathbf{C}_1, \mathbf{S}_1), \text{com})$	if $\tilde{\mathbf{O}} \notin \text{diag}(\mathcal{O}_n(\mathbb{Z}), \mathcal{O}_\ell(\mathbb{Z}))$: return 0
return $\text{chal} \leftarrow_{\$} \{0, 1\}$	return $\tilde{\mathbf{C}} = \text{SF}(\tilde{\mathbf{O}} \cdot [\mathbf{C}_b \parallel \mathbf{S}_b])$

Figure 8: One iteration of our form-enforcing Σ -protocol (P_1, V_1, P_2, V_2) for the following relation. A statement is given by $x = (\mathbf{C}_0, \mathbf{S}_0, \mathbf{C}_1, \mathbf{S}_1)$ where $(\mathbf{C}_0, \mathbf{C}_1)$ is a pair of code generator matrices and $(\mathbf{S}_0, \mathbf{S}_1)$ is a pair of (syndrome) matrices. A witness is given by $w = (\mathbf{O}, \mathbf{P})$ such that $\mathbf{C}_1 \cdot \mathbf{U} = \text{SF}(\mathbf{O} \cdot \mathbf{C}_0)$ and $\mathbf{S}_1 = \mathbf{P} \cdot \mathbf{S}_0 \cdot \mathbf{U}$.

Definition 18 (Update relation). Let $(\text{KGen}, \text{Enc}, \text{Dec}, \text{UpdPk}, \text{UpdSk})$ be a CU-friendly UPKE scheme. The update relation for UPKE is defined by instances $(\text{pk}, \text{pk}') \in \mathcal{PK}^2$ where there exists a witness $\text{dup} \in \mathcal{UP}$ such that $\text{pk}' = \text{UpdPk}_{\text{CU}}(\text{pk}, \text{dup})$.

Definition 19 (Member tags). A member tag scheme MT for a CU-friendly UPKE is a complete, zero-knowledge and straight-line simulation-extractable NIZKPoK (Definition 14) for the update relation for UPKE.

Remark 2. It may be possible in Definition 19 to use a NIZKPoK that extracts a relaxed witness from some larger set $\mathcal{UP}' \supseteq \mathcal{UP}$, as discussed in [AL21]. Currently this is prohibited by the syntax of Definition 18, since UpdPk_{CU} simply aborts on malformed input not in $\mathcal{PK} \times \mathcal{UP}$. In this case, the proof of Lemma 8 would likely have to be amended to account for the lossy extractor. We do not pursue this further since in this work we extract the exact witness used by the prover.

4.2 Member Tags for Our UPKE

Denote $\text{pk} = (\mathbf{A}, \mathbf{u})$ and $\text{pk}' = (\mathbf{B}, \mathbf{v})$ two UPKE public keys from Fig. 3 where pk' was obtained from pk by updating. At their core, our member tags are Graph Isomorphism-style Σ -protocol proofs of linear isometry from [PS23], where instead of using the FS transform, we use Fischlin's transform [Fis05] that offers straight-line simulation-extractability (which will be required later in Section 5). Notice, however, that proving $\mathbf{A}$ and $\mathbf{B}$ generate isometric codes is not enough to prove that pk being well-formed implies the same for pk' . Indeed, any pair $\tilde{\text{pk}} = (\mathbf{B}, \mathbf{w} \in \mathbb{F}_q^k)$ is also accepted by a verifier without the corresponding secret key, which, if adopted, produces ciphertexts that cannot be decrypted. Hence we modify the Σ -protocol in Fig. 6 to enforce that the last coordinate of the code is left alone by the isometry, and send a proof that $[\mathbf{A} \parallel \mathbf{u}^T]$ is isometric to $[\mathbf{B} \parallel \mathbf{v}^T]$. We establish a slightly more general result that allows for more than one syndrome, which we use in Section 7 to construct member tags for the multi-key scheme that are only marginally larger than those for the single-key variant.

Definition 20. Let $\mathbf{C}_0, \mathbf{C}_1 \in \mathbb{F}_q^{n \times k}$ generate $[n, k]$ -linear codes over $\mathbb{F}_q$ and let $\mathbf{S}_0, \mathbf{S}_1 \in \mathbb{F}_q^{\ell \times k}$. Consider the Σ -protocol in Fig. 8 that proves knowledge of a pair of signed permutations $(\mathbf{O}, \mathbf{P}) \in \mathcal{O}_n(\mathbb{Z}) \times \mathcal{O}_\ell(\mathbb{Z})$ such that $[\mathbf{C}_1 \parallel \mathbf{S}_1] = \text{SF}(\text{diag}(\mathbf{O}, \mathbf{P}) \cdot [\mathbf{C}_0 \parallel \mathbf{S}_0])$.

Notice that the new square class invariant [Ben26] does not play a role here as it of course matches for equivalent codes. Completeness and honest-verifier zero-knowledge of the Σ -protocol in Fig. 8 are standard. We prove it is also special sound, in particular implying our form-enforcing condition is correct.

Lemma 3. Assume the notation of Definition 20. The Σ -protocol in Fig. 8 admits 2-special soundness.

$\text{MTag}(\text{pp}, \text{pk}, \text{pk}', \text{dup})$	$\text{Setup}(1^\lambda)$
$\text{parse } ((\mathbf{A}, \mathbf{u}), (\mathbf{B}, \mathbf{v})) \leftarrow (\text{pk}, \text{pk}')$	$\text{return } \text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda)$
$\text{parse } \mathbf{O} \leftarrow \text{dup}; (\mathbf{B}', \mathbf{U}) := \text{SF}(\mathbf{O} \cdot \mathbf{A})$	
$\text{assert } \mathbf{B} = \mathbf{B}' \text{ and } \mathbf{v} = \mathbf{U}^\top \cdot \mathbf{u}$	$\text{Vf}(\text{pp}, \text{pk}, \text{pk}', \text{mt})$
$\pi \leftarrow \Pi.\text{Prove}(\text{pp}, (\mathbf{A}, \mathbf{u}^\top, \mathbf{B}, \mathbf{v}^\top), (\mathbf{O}, 1))$	$\text{parse } ((\mathbf{A}, \mathbf{u}), (\mathbf{B}, \mathbf{v}), \pi) \leftarrow (\text{pk}, \text{pk}', \text{mt})$
$\text{return } \text{mt} := \pi$	$\text{return } \Pi.\text{Vf}(\text{pp}, (\mathbf{A}, \mathbf{u}^\top, \mathbf{B}, \mathbf{v}^\top), \pi)$

Figure 9: Our explicit member tags MT for the scheme in Fig. 3 using the proof of isometry NIZKPoK scheme $\Pi = (\text{Setup}, \text{Prove}, \text{Vf}, \text{Sim}, \text{Ext})$ that is the Σ -protocol in Fig. 8 compiled with Fischlin's transform in Fig. 7 (or Fig. 10).

Proof. The proof is standard. Let $(\mathbf{C}_0, \mathbf{S}_0, \mathbf{C}_1, \mathbf{S}_1)$ be an instance, and let $(\text{com}_0, \text{chal}_0, \text{resp}_0)$ and $(\text{com}_1, \text{chal}_1, \text{resp}_1)$ be two accepting transcripts with $\text{com}_0 = \text{com}_1 = \tilde{\mathbf{C}}$. Wlog. we have $\text{chal}_b = b$ for $b \in \{0, 1\}$. Denote $\text{resp}_0 = \tilde{\mathbf{O}}_0 = \text{diag}(\mathbf{O}_0, \mathbf{O}_1)$, since $(\text{com}_0, \text{chal}_0, \text{resp}_0)$ was accepted by $\mathbf{V}_2$. Analogously, denote $\text{resp}_1 = \tilde{\mathbf{O}}_1 = \text{diag}(\mathbf{O}_2, \mathbf{O}_3)$. Then

$$\mathbf{O}' = \tilde{\mathbf{O}}_1^{-1} \cdot \tilde{\mathbf{O}}_0 = \text{diag}(\mathbf{O}_2^{-1} \cdot \mathbf{O}_0, \mathbf{O}_3^{-1} \cdot \mathbf{O}_1) = \text{diag}(\mathbf{O}, \mathbf{P}) \in \text{diag}(\mathcal{O}_n(\mathbb{Z}), \mathcal{O}_\ell(\mathbb{Z}))$$

gives a map with the correct form from $[\mathbf{C}_0 \parallel \mathbf{S}_0]$ to $[\mathbf{C}_1 \parallel \mathbf{S}_1]$, since

$$\text{SF}(\mathbf{O}' \cdot [\mathbf{C}_0 \parallel \mathbf{S}_0]) = \text{SF}(\tilde{\mathbf{O}}_1^{-1} \cdot \tilde{\mathbf{C}}) = \text{SF}([\mathbf{C}_1 \parallel \mathbf{S}_1]). \quad \square$$

Remark 3. The verifier's form check can be made more precise if $\ell = 1$. The verifier can enforce the response is of the form $\text{diag}(\mathbf{O}, 1)$ instead of allowing ± 1 in the last coordinate. The honest prover now sets $\mathbf{O}_1 := 1$ to comply. This means we can use the Σ -protocol from Fig. 8 with this stricter check to precisely prove that if $((\mathbf{A}, \mathbf{u}), \mathbf{r})$ is a key-pair for the UPKE scheme in Fig. 3 and $(\mathbf{B}, \mathbf{v})$ is a public key that passes this proof, then there is an isometry $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$ such that $\mathbf{B}, \mathbf{U} = \text{SF}(\mathbf{O} \cdot \mathbf{A})$ and precisely $\mathbf{v}^\top = \mathbf{u}^\top \cdot \mathbf{U}$ (instead of up to a sign).

Theorem 2. *Let Π be the NIZKPoK obtained by applying Fischlin's transform with parameters $(b, r, 0, t)$ from Definition 15 to the Σ -protocol in Fig. 8 (and Remark 3). Then Π is a member tag scheme for UPKE from Fig. 3.*

Proof. The UPKE scheme from Fig. 3 is clearly CU-friendly by defining valid update material as $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$. Since our member tag is now a single proof of isometry with one syndrome from Fig. 8, it clearly inherits its NIZKPoK properties from the underlying Σ -protocol (Lemma 3) and the properties of Fischlin's transform from Section 2.5. $\square$

We endow our member tag construction with a more explicit syntax in Fig. 9.

4.3 Fischlin's Transform with Fixed-Weight Challenges

A common size optimisation technique in constructing signatures from Σ -protocols compiled with the FS transform is to consider *fixed-weight challenges* instead of uniform ones (see e.g. [BKV19, BBPS21, PS23]). Recall that the commitment randomness is typically sampled using a seed shorter in length than the actual randomness, so if the challenge bit is 0, the proof has smaller expected size if we send the seed instead of the commitment randomness, from which the verifier can reconstruct it (whereas if the challenge bit is 1 we must send the randomness composed with our witness for which we do not know a seed). The main idea is thus to bias the challenge towards 0 and make it appropriately longer, with the hope that the total proof size is smaller due to having to send more seeds rather than commitment randomness. This distribution is precisely

$\text{P}_{\text{Fis}}^{\text{H}}(x, w)$
for $i = 1, \dots, r$ **do** $(\text{com}_i, \text{st}_i) \leftarrow \text{P}_1(x, w)$ **endfor** ; $\text{com} := [\text{com}_1, \dots, \text{com}_r]$; $\pi := [\cdot]$
for $i = 1, \dots, r$ **do for** $\text{chal}_i \in \{\text{strings of length } \ell \text{ and weight } w\}$ **do**
 $\text{resp}_i \leftarrow \text{P}_2(\text{st}_i, x, w, \text{chal}_i)$
if $\text{H}(x, \text{com}, i, \text{chal}_i, \text{resp}_i) = 0^b$: $\pi[i] := (\text{com}_i, \text{chal}_i, \text{resp}_i)$ **and break** chal_i -loop
if $\exists i: \pi[i] = \perp$: **return** $\perp$ **else** : **return** π

$\text{V}_{\text{Fis}}^{\text{H}}(x, \pi)$
for $i := 1, \dots, r$ **if** $w(\text{chal}_i) \neq w$ **or** $\text{V}_2(x, \text{com}_i, \text{chal}_i, \text{resp}_i) = 0$: **return** 0
if $\sum_{i=1}^r \text{H}(x, \text{com}, i, \text{chal}_i, \text{resp}_i) > S$: **return** 0 **else** : **return** 1

Figure 10: Fischlin's transform from Fig. 7 with fixed-weight challenges.

the Bernoulli distribution. It is common to consider fixed-weight challenges for a precise soundness analysis, lest we deal with moments of the Bernoulli distribution. Thus we pick challenges of length ℓ and weight w such that $\binom{\ell}{w} \geq 2^\lambda$ to achieve soundness error below $2^{-\lambda}$.

This approach works for the FS transform since we can treat the λ uniform challenge bits as one challenge string of length λ , and can thus bias it appropriately as long as the probability of guessing it stays below $2^{-\lambda}$. However, this is not the case for Fischlin's transform, where each round already has multiple-bit challenges that have to individually satisfy proof-of-work constraints, and the prover has to iterate over them. We use the fact that each round already has multiple bits to apply the fixed-weight optimisation to *each round separately*: instead of searching through the 2^t bit-strings of length t to find one that satisfies proof-of-work, we search through $\binom{\ell}{w}$ bit-strings of length ℓ and Hamming weight w such that $\binom{\ell}{w} \geq 2^t$ to find one that satisfies proof-of-work, but $w \ll \frac{t}{2}$ and ℓ appropriately large. Since proof-of-work is done with a random oracle, it is immediate that this comes with no cost to completeness or simulation-extractability. The final challenge string then has length $r \cdot \ell$ and weight $r \cdot w$ for r the number of rounds, and we have

$$|\pi| = \lambda + r \cdot (\ell + (\ell - w) \cdot \lambda + w \cdot C),$$

where λ is the seed length and C is the response length when $\text{chal} = 1$, that is, $n + n \cdot \lceil \log(n) \rceil$ for a signed permutation. This is much smaller compared to using uniform challenges (cf. Section 2.5) as soon as $w \ll \frac{t}{2}$, especially for large n . Fischlin's transform with the round-wise fixed-weight challenges is presented in Fig. 10.

4.4 Quasi-Unique Responses

Recall from Section 2.5 that we need our Σ -protocol in Definition 20 to have quasi-unique responses: it should be hard for any PPT adversary to find an accepting transcript with more than one valid response. We note that this is not true even for the standard Σ -protocol based on code equivalence in Fig. 6, since for any response $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$, we have that $-\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$ is also valid. More generally, denote by $\text{Aut}(\mathfrak{C}) = \{\mathbf{O} \in \mathcal{O}_n(\mathbb{Z}) \mid \mathfrak{C} = \mathbf{O} \cdot \mathfrak{C}\}$ the *orthogonal automorphism group* of the linear code $\mathfrak{C}$. If $\mathfrak{C}_1 = \mathbf{O} \cdot \mathfrak{C}_0$, then all other isometries that take $\mathfrak{C}_0$ to $\mathfrak{C}_1$ are exactly $\text{Iso}(\mathfrak{C}_0, \mathfrak{C}_1) = \{\mathbf{O} \cdot \mathbf{O}' \mid \mathbf{O}' \in \text{Aut}(\mathfrak{C}_0)\}$. Clearly $\{\pm \mathbf{I}\} \leq \text{Aut}(\mathfrak{C})$ for any $\mathfrak{C}$, thus the standard assumption is that no PPT algorithm can, given $\mathfrak{C}$ from some distribution, efficiently compute $\mathbf{O} \in \text{Aut}(\mathfrak{C}) \setminus \{\pm \mathbf{I}\}$ [BCD⁺24a]. In group action cryptography this is known as the Stabilizer Computation Problem (SCP) and is normally assumed hard either worst-case or for codes sampled from $\mathcal{U}_{n,k,q}$. To make our construction work, we assume that SCP is hard also for hollow codes.

Definition 21 (Hollow SCP). *Let $n, k, h, q, \chi \in \mathbb{Z}$ parametrised by λ with $0 \leq h \leq k \leq n - h$, $\chi \in \{\pm 1\}$ and q odd prime. The advantage of any algorithm $\mathcal{A}$ against Hollow SCP is*

$$\text{Adv}_{\text{SCP}}(\mathcal{A}) = \Pr \left[\text{SF}(\mathbf{A}) = \text{SF}(\mathbf{A} \cdot \mathbf{O}) \wedge \mathbf{O} \in \mathcal{O}_n(\mathbb{Z}) \setminus \{\pm \mathbf{I}\} \mid \mathbf{A} \leftarrow \mathcal{D}_{n,k,q}^{\chi,h}, \mathbf{O} \leftarrow \mathcal{A}(\mathbf{A}) \right].$$

$\text{Game}_{\text{Fork-IND-CPA}}^A(\lambda)$	$\text{OChal}(\text{msg}_0, \text{msg}_1, (\text{dup}_i)_{i=1}^T)$
$b \leftarrow \{0, 1\}$	assert $T \leq t$ and $\forall i: \text{dup}_i$ valid
$\text{pp} \leftarrow \text{Setup}(1^\lambda)$	for $i = 1, \dots, T$ do
$(\text{pk}_0, \text{sk}_0) \leftarrow \text{KGen}(\text{pp})$	$\text{pk}_i \leftarrow \text{UpdPk}_{\text{CU}}(\text{pk}_{i-1}, \text{dup}_i); \text{sk}_i \leftarrow \text{UpdSk}_{\text{CU}}(\text{sk}_{i-1}, \text{dup}_i)$
$b' \leftarrow \mathcal{A}^{\text{OChal}}(\text{pp}, \text{pk}_0)$	$\text{ctxt} \leftarrow \text{Enc}(\text{pk}_T, \text{msg}_b); (\text{pk}', \text{up}) \leftarrow \text{UpdPk}(\text{pk}_T)$
return $b = b'$	$\text{sk}' \leftarrow \text{UpdSk}(\text{sk}_T, \text{up});$ return $(\text{ctxt}, \text{pk}', \text{sk}', \text{up})$

Figure 11: The Fork-IND-CPA game for a UPKE scheme. dup_i are (unencrypted) update material used in order to update the key-pair, for example $\mathbf{O}_i \in \mathcal{O}_n(\mathbb{Z})$ in the case of Fig. 3, or linear shifts in [AFMR26].

Assumption 1 Assume the notation of Definition 21. For any PPT algorithm $\mathcal{A}$ we have that $\text{Adv}_{\text{SCP}}(\mathcal{A}) \leq \text{negl}(\lambda)$.

SCP is of course vacuous if $\text{Aut}(\mathfrak{C}) = \{\pm \mathbf{I}\}$ with overwhelming probability over the randomness of $\mathfrak{C}$. The authors of [LWZ26] show this is the case for codes sampled from $\mathcal{U}_{n,k,q}$ with overwhelming probability in k for our parameters of interest (namely $n = 2k$), however no such result exists for codes sampled using $\mathcal{D}_{n,k,q}^{\chi,h}$, hence we rely on Assumption 1 instead.

Our scheme in Definition 20 has an additional problem: because valid isometries lie in $\mathcal{O}_n(\mathbb{Z}) \times \mathcal{O}_\ell(\mathbb{Z})$, we have that $\{\text{diag}(\pm \mathbf{I}_n, \pm \mathbf{I}_\ell)\}$ is always a subgroup of the automorphism group of any (stacked) instance. However, using the notation of Fig. 8, recall that the systematic form transform of $[\mathbf{C}, \mathbf{S}]$ relies solely on $\mathbf{C}$, not on $\mathbf{S}$. Hence we can introduce an additional check by the verifier that when sampling the commitment $\text{diag}(\mathbf{O}_0, \mathbf{O}_1)$ in Fig. 8, we only allow the one $\pm \mathbf{O}_1$ such that the top-left entry s of the commitment syndrome $\mathbf{S}$ satisfies $0 \leq s < \frac{q-1}{2}$. The honest prover of course complies with this. Now we are again in the setting where the only known allowed automorphisms are $\{\text{diag}(\pm \mathbf{I}_n, \mathbf{I}_\ell)\}$. This is the multi-key analogue to Remark 3. We may account for the known automorphisms by setting the proof-of-work parameter from b to $b + 1$ in Section 8.

5 IND-CCA Secure UKEM

We give a transformation that compiles an unbounded UPKE scheme (and member tags for it) satisfying a forking version of chosen update security into an IND-CCA-secure UKEM (Definition 13), which we apply to our UPKE scheme in Fig. 3. The formal proofs, despite their length, merely amount to a modularisation of the transformations and a relaxation of the requirements regarding the IND-CCA model for UKEM in Definition 13 from [AFMR26]. We believe Theorem 3 is generic enough to be usable with other schemes such as [BFK⁺26].

5.1 Forking Security and Update Freshness

Security under adversarially chosen updates (rather than randomness) was already studied in [DKW22, HPS23]. We give a forking version akin to Shifty-IND-CPA from [AFMR26].

Definition 22 (Fork-IND-CPA security of UPKE). Let $t = \text{poly}(\lambda)$. Consider the game $\text{Game}_{\text{Fork-IND-CPA}}$ in Fig. 11. A CU-friendly UPKE scheme is Fork-IND-CPA secure, if for any PPT adversary $\mathcal{A}$ we have

$$\text{Adv}_{\text{Fork-IND-CPA}}(\mathcal{A}) = \left| \Pr \left[1 \leftarrow \text{Game}_{\text{Fork-IND-CPA}}^A(\lambda) \right] - \frac{1}{2} \right| \leq \text{negl}(\lambda).$$

The property of *update freshness* captures the naturally desirable behaviour of a UPKE scheme that a fresh pair of keys should be indistinguishable from an honestly updated pair, even if it has already been updated

$\text{Game}_{\text{UF}}^{\mathcal{A}}(\lambda)$	$\text{OChal}((\text{dup}_i)_{i=1}^{t_{j+1}})$
$i := 0$	assert $\forall \text{dup}_i \in \mathcal{UP}; j := j + 1; (\text{pk}, \text{sk}) := (\text{pk}_0, \text{sk}_0)$
$(\text{pk}_0, \text{sk}_0) \leftarrow \text{KGen}(1^\lambda)$	for $i := 1, \dots, t_j$ do
$b \leftarrow_{\$} \{0, 1\}$	$\text{pk} := \text{UpdPk}_{\text{CU}}(\text{pk}, \text{dup}_i); \text{sk} := \text{UpdSk}_{\text{CU}}(\text{sk}, \text{dup}_i)$
$b' \leftarrow \mathcal{A}(\text{pk}_0, \text{sk}_0)$	$(\text{pk}^{(1)}, \text{up}) \leftarrow \text{UpdPk}(\text{pk}); \text{sk}^{(1)} \leftarrow \text{UpdSk}(\text{sk}, \text{up})$
return $b = b'$	$(\text{pk}^{(0)}, \text{sk}^{(0)}) \leftarrow \text{KGen}(1^\lambda); \text{return } (\text{pk}^{(b)}, \text{sk}^{(b)})$

Figure 12: The update freshness game for CU-friendly UPKE.

many times with adversarially chosen update material. To the best of our knowledge it has not been defined before in the literature. Update freshness is naturally attained by UPKE schemes based on discrete logarithm, but not by any lattice-based scheme, except notably the scheme from [ABL25a], as we later prove.

Definition 23 (Update Freshness). Consider Game_{UF} from Fig. 12. A CU-friendly UPKE is called update-fresh, if for any PPT algorithm $\mathcal{A}$ we have

$$\text{Adv}_{\text{UF}}(\mathcal{A}) = \left| \Pr \left[1 \leftarrow \text{Game}_{\text{UF}}^{\mathcal{A}}(\lambda) \right] - \frac{1}{2} \right| \leq \text{negl}(\lambda).$$

The UPKE scheme in Fig. 3 satisfies both properties.

Lemma 4. For any PPT algorithm $\mathcal{A}$ against Game_{UF} for the UPKE scheme in Fig. 3 there is a PPT algorithm $\mathcal{B}$ such that

$$\text{Adv}_{\text{UF}}(\mathcal{A}) \leq \text{Adv}_{\Delta\text{PCE}}(\mathcal{B}).$$

Proof (Sketch). This is proven in [ABL25a, Lem. 11] (and [Ben26, Lem. 3]).

Lemma 5 (Fork-IND-CPA). For the parameters under which ΔPCE , LWE^h and DSIS^h are assumed hard, the UPKE scheme from Fig. 3 is Fork-IND-CPA secure. Concretely, for any PPT algorithm $\mathcal{A}$ against UPKE that makes q_{Chal} calls to the OChal oracle, there are PPT algorithms $\mathcal{B}_1, \mathcal{B}_2$ such that

$$\text{Adv}_{\text{Fork-IND-CPA}}^{\text{UPKE}}(\mathcal{A}) \leq \text{Adv}_{\text{UF}}^{\text{UPKE}}(\mathcal{B}_1) + \text{Adv}_{\text{IND-CPA}}^{\text{PKE}}(\mathcal{B}_2) + \frac{q_{\text{RO}}}{2^\lambda},$$

where $\mathcal{B}_1$ makes q_{Chal} calls to OChal and $\mathcal{B}_2$ makes q_{Chal} calls to OEnc .

Proof (Sketch). The full proof is in Section 5.4 and closely follows the strategy of the proof of IND-CR-CPA security in [ABL25a, Thm. 3]. Game_0 is the Fork-IND-CPA game for UPKE. In Game_1 and Game_2 , we first flag correct update randomness guesses as bad and then abort if this bad event occurs. In Game_3 , we use the ROM to decouple the update material from the encrypted randomness. In Game_4 , we replace the leaked keys with fresh pairs using update freshness, and are left to deal with the IND-CPA game with q_{Chal} challenges in Game_5 . We then bound the probability of the bad event.

5.2 A Chain of Transformations

We modularise the transformation from [AFMR26] that compiles a UPKE scheme into an IND-CCA-secure UKEM into three simpler transformations that compile into intermediate security notions (Definition 13). First we establish that the transformation in Fig. 13 compiles a Fork-IND-CPA-secure UPKE scheme into an IND-CPA⁺-secure UKEM (Definition 13).

<u>Setup(1^λ)</u>	<u>Encaps(pp, pk)</u>
return pp := H_{kk}	msg $\leftarrow$ $\{0, 1\}^\lambda$; kk $\leftarrow H_{kk}(\text{pk}, \text{msg})$
<u>KGen(pp)</u>	ctxt $\leftarrow \text{UPKE.Enc}(\text{pk}, \text{msg})$
(pk, sk) $\leftarrow \text{UPKE.KGen}(1^\lambda)$; return (pk, sk)	(pk', up) $\leftarrow \text{UPKE.UpdPk}(\text{pk})$
<u>Decaps(pp, sk, ctxt, pk, up)</u>	return (kk, ctxt, pk', up)
msg $\leftarrow \text{UPKE.Dec}(\text{sk}, \text{ctxt})$; kk $\leftarrow H_{kk}(\text{pk}, \text{msg})$	<u>$H_{kk}(\text{msg})$</u>
sk' $\leftarrow \text{UPKE.UpdSk}(\text{sk}, \text{up})$; return (kk, sk')	if msg $\notin \mathcal{Q}$: $\mathcal{Q}[\text{msg}] \leftarrow \{0, 1\}^\lambda$
	return $\mathcal{Q}[\text{msg}]$

Figure 13: Construction of UKEM from UPKE = (KGen, Enc, Dec, UpdPk, UpdSk).

Lemma 6 (Fork-IND-CPA $\rightarrow$ IND-CPA⁺ adapted from [AFMR25, Thm. 5]). *Let UPKE be a CU-friendly, δ -correct and update fresh scheme. If UPKE is Fork-IND-CPA secure then the resulting UKEM from Fig. 13 is correct and IND-CPA⁺ secure. Concretely, for any PPT algorithm $\mathcal{A}$ against IND-CPA⁺ for UKEM that creates t nodes with q_{Encaps} queries to OEncaps and $\text{poly}(\lambda)$ queries to OUpd, and queries its random oracles q_{RO} times, there are PPT algorithms $\mathcal{B}_1, \mathcal{B}_2$ such that*

$$\text{Adv}_{\text{IND-CPA}^+}^{\text{UKEM}}(\mathcal{A}) \leq \frac{4q_{\text{Encaps}} + 5}{2^\delta} + t \cdot \left(\text{Adv}_{\text{UF}}^{\text{UPKE}}(\mathcal{B}_1) + \text{Adv}_{\text{Fork-IND-CPA}}^{\text{UPKE}}(\mathcal{B}_2) + \frac{2q_{\text{RO}}}{2^\lambda} \right),$$

where $\mathcal{B}_2$ makes at most q_{Encaps} queries to OChal in the Fork-IND-CPA game and $\mathcal{B}_1$ makes one query to OChal in the update freshness game.

Proof (Sketch). The full proof is in Section 5.5. We turn the IND-CPA⁺ challenger against $\mathcal{A}$ for UKEM into an adversary for Fork-IND-CPA of UPKE. **Game**₀ is the IND-CPA⁺ game for UKEM and in **Game**₁ we account for potential correctness errors, giving the $2^{-\delta}$ term. In **Game**₂ we guess a favourable ancestor of the challenge node with probability $\frac{1}{t}$, in **Game**₃ we track if the adversary has queried the seed for the update material of the challenge node. In **Game**₄, we plant our own key-pair into the guessed node, relying on update freshness. Finally we may take input from a Fork-IND-CPA challenger and use the provided OChal oracle to simulate OEncaps(i) and OMChal(i) when it does not know sk_i .

Next we establish that the transformation in Fig. 14 compiles an IND-CPA⁺-secure UKEM into a IND-CCA⁻-secure UKEM (Definition 13).

Lemma 7 (IND-CPA⁺ $\rightarrow$ IND-CCA⁻ adapted from [AFMR25, Thm. 6]). *Suppose the underlying UPKE scheme is δ -correct and γ -spread. If the UKEM scheme in Fig. 13 obtained from UPKE is IND-CPA⁺ secure, then UKEM_{FO} constructed in Fig. 14 is correct and IND-CCA⁻ secure. Concretely, for any PPT algorithm $\mathcal{A}$ against UKEM_{FO} that makes q_{RO} queries to its random oracles, q_{Encaps} queries to OEncaps, and q_{Decaps} queries to ODecaps⁻, there is a PPT algorithm $\mathcal{B}$ against UKEM that makes at most q_{RO} queries to its random oracles, q_{Encaps} queries to its OEncaps, and q_{Decaps} queries to OUpd, such that*

$$\text{Adv}_{\text{IND-CCA}^-}^{\text{UKEM}_{\text{FO}}}(\mathcal{A}) \leq \frac{6t + q_{\text{RO}}}{2^\delta} + \frac{q_{\text{Decaps}}}{2^\gamma} + \frac{q_{\text{RO}} + q_{\text{Encaps}}}{2^\lambda} + \text{Adv}_{\text{IND-CPA}^+}^{\text{UKEM}}(\mathcal{B}).$$

Proof (Sketch). The full proof is in Section 5.6, and is a typical FO-style proof. We turn the IND-CCA⁻ challenger against $\mathcal{A}$ for UKEM_{FO} into an adversary for IND-CPA⁺ for UKEM. **Game**₀ is the IND-CCA⁻ game for UKEM_{FO} and in **Game**₁ we account for potential correctness errors, giving the $2^{-\delta}$ term. In **Game**₂ we start silently recording all $H_i(\text{pk}, \text{msg})$ queries for $i \in \{\text{kk}, \text{ctxt}, \text{up}\}$, always calling all of them and storing (pk, pk', dup, msg, ctxt, kk), where the pk', dup and ctxt are computed as in the scheme. In **Game**₃ we flag as bad all ODecaps⁻($i, \text{ctxt}', \text{pk}'$) queries where we have not previously recorded a corresponding

$\text{for } i \in \{\text{kk}, \text{ctxt}, \text{up}\} : H_i(\text{msg})$	$\text{Setup}(1^\lambda)$
$\text{if } \text{msg} \notin \mathcal{Q}_i : \mathcal{Q}_i[\text{msg}] \leftarrow \$ \{0, 1\}^\lambda$	$\text{return } \text{pp} := (H_{\text{kk}}, H_{\text{ctxt}}, H_{\text{up}})$
$\text{return } \mathcal{Q}_i[\text{msg}]$	$\text{Decaps}(\text{pp}, \text{sk}, \text{ctxt}, \text{pk}, \text{pk}')$
$\text{KGen}(\text{pp})$	$\text{msg} \leftarrow \text{UPKE.Dec}(\text{sk}, \text{ctxt})$
$(\text{pk}, \text{sk}) \leftarrow \text{UPKE.KGen}(1^\lambda); \text{return } (\text{pk}, \text{sk})$	$\text{kk} \leftarrow H_{\text{kk}}(\text{pk}, \text{msg})$
$\text{Encaps}(\text{pp}, \text{pk})$	$\rho_{\text{ctxt}} \leftarrow H_{\text{ctxt}}(\text{pk}, \text{msg})$
$\text{msg} \leftarrow \$ \{0, 1\}^\lambda; \text{kk} \leftarrow H_{\text{kk}}(\text{pk}, \text{msg})$	$\text{ctxt}' \leftarrow \text{UPKE.Enc}(\text{pk}, \text{msg}; \rho_{\text{ctxt}})$
$\rho_{\text{ctxt}} \leftarrow H_{\text{ctxt}}(\text{pk}, \text{msg})$	$\text{assert } \text{ctxt} = \text{ctxt}'$
$\text{ctxt} \leftarrow \text{UPKE.Enc}(\text{pk}, \text{msg}; \rho_{\text{ctxt}})$	$\rho_{\text{up}} \leftarrow H_{\text{up}}(\text{pk}, \text{msg})$
$\rho_{\text{up}} \leftarrow H_{\text{up}}(\text{pk}, \text{msg})$	$(\tilde{\text{pk}}, \text{up}) \leftarrow \text{UPKE.UpdPk}(\text{pk}; \rho_{\text{up}})$
$(\text{pk}', \_) \leftarrow \text{UPKE.UpdPk}(\text{pk}; \rho_{\text{up}})$	$\text{assert } \text{pk}' = \tilde{\text{pk}}$
$\text{return } (\text{kk}, \text{ctxt}, \text{pk}', \text{up})$	$\text{sk}' \leftarrow \text{UPKE.UpdSk}(\text{sk}, \text{up})$
$\text{return } (\text{kk}, \text{ctxt}, \text{pk}', \text{up})$	$\text{return } (\text{kk}, \text{sk}')$

Figure 14: An FO-type transformation (white-box) yielding UKEM_{FO} from $\text{UKEM} = (\text{Setup}, \text{KGen}, \text{Encaps}, \text{Decaps})$ in Fig. 13. All modifications highlighted.

$\text{Setup}(1^\lambda)$	$\text{Encaps}(\text{pp}, \text{pk})$
$\dots; \text{pp}_{\text{mt}} \leftarrow \text{MT.Setup}(1^\lambda)$	$\dots; \text{mt} \leftarrow \text{MT.MTag}(\text{pp}_{\text{mt}}, \text{pk}, \text{pk}', \text{UPKE.H}(\rho_{\text{up}}))$
$\text{return } \text{pp} := (\text{pp}_{\text{mt}}, H_{\text{kk}}, H_{\text{ctxt}}, H_{\text{up}})$	$\text{return } (\text{kk}, \text{ctxt}, \text{pk}', \text{mt})$
$\text{Vf}(\text{pp}, \text{pk}, \text{pk}', \text{mt})$	
$\text{return } \text{MT.Vf}(\text{pp}_{\text{mt}}, \text{pk}, \text{pk}', \text{mt})$	

Figure 15: A transformation joining a member tag scheme $\text{MT} = (\text{Setup}, \text{MTag}, \text{Vf})$ with UKEM_{FO} , yielding $\text{UKEM}_{\text{FOMO}}$. Only modifications shown and highlighted.

$(\text{pk}_i, \text{pk}', \text{dup}, \text{msg}, \text{ctxt}', \text{kk})$, and in Game_4 and Game_5 we abort on bad queries, giving the $2^{-\gamma}$ term. Game_6 returns values from its records instead of decapsulating, and then we may finally take input from an IND-CPA^+ challenger, using the provided OUpd to simulate decapsulation nodes. We then notice the advantages in the IND-CPA^+ game for the randomised and the de-randomised schemes can differ only if the adversary guesses a random oracle value.

Finally we state that the transformation in Fig. 15 compiles an IND-CCA^- -secure UKEM into a IND-CCA -secure UKEM (Definition 13).

Lemma 8 ($\text{IND-CCA}^- \rightarrow \text{IND-CCA}$ adapted from [AFMR25, Thm. 7]). *If UKEM_{FO} from Fig. 14 is IND-CCA^- secure, the underlying UPKE is CU-friendly, and MT is a member tag scheme for UPKE, then $\text{UKEM}_{\text{FOMO}}$ from Fig. 15 is IND-CCA secure. Concretely, for any PPT algorithm $\mathcal{A}$ against the IND-CCA game for $\text{UKEM}_{\text{FOMO}}$, there are PPT algorithms $\mathcal{B}, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, such that*

$$\text{Adv}_{\text{IND-CCA}}^{\text{UKEM}_{\text{FOMO}}}(\mathcal{A}) \leq \text{Adv}_{\text{IND-CCA}^-}^{\text{UKEM}_{\text{FO}}}(\mathcal{B}) + \text{Adv}_{\text{SI}}^{\text{MT}}(\mathcal{B}_1) + \text{Adv}_{\text{ZK}}^{\text{MT}}(\mathcal{B}_2) + \text{Adv}_{\text{Ext}}^{\text{MT}}(\mathcal{B}_3).$$

Proof (Sketch). The full proof is in Section 5.7 and is almost identical to [AFMR25, Thm. 7]. We turn the IND-CCA challenger against $\mathcal{A}$ for $\text{UKEM}_{\text{FOMO}}$ into an adversary for IND-CCA^- for UKEM_{FO} . We first account with Game_1 for the alternative setup required in some NIZKPoK (but not ours). In Game_2 we replace all member tags with simulated proofs, relying on zero-knowledge. Game_3 starts bookkeeping when half-nodes are introduced (since there should be none in IND-CCA^-), and extracts the update material from the mt

provided to ODecaps when there is no record of mt having been simulated. We may then take input from an IND-CCA^- challenger, using the provided oracles to simulate ODecaps .

5.3 Composing the Transformations

We compose the above chain of transformations.

Theorem 3. *Let UPKE be a δ -correct, γ -spread and update-fresh scheme that is CU -friendly with member tags MT . If UPKE is Fork-IND-CPA secure then the composition $\text{T}[\text{UPKE}, \text{MT}]$ of transformations from Figs. 13 to 15 produces a correct and IND-CCA-secure $\text{UKEM}_{\text{FOMO}}$ in the ROM. Concretely, for any PPT algorithm $\mathcal{A}$ against the IND-CCA game for $\text{UKEM}_{\text{FOMO}}$ that creates t nodes through q_{Encaps} queries to OEncaps and q_{Decaps} queries to ODecaps , and makes q_{RO} queries to its random oracles, there are PPT algorithms $\mathcal{B}_i$ for $i = 1, \dots, 5$ such that*

$$\begin{aligned} \text{Adv}_{\text{IND-CCA}}^{\text{UKEM}_{\text{FOMO}}}(\mathcal{A}) &\leq t \cdot \text{Adv}_{\text{Fork-IND-CPA}}^{\text{UPKE}}(\mathcal{B}_1) + t \cdot \text{Adv}_{\text{UF}}^{\text{UPKE}}(\mathcal{B}_2) + \frac{6t + q_{\text{RO}}}{2^\delta} + \frac{q_{\text{Decaps}}}{2^\gamma} \\ &\quad + \frac{q_{\text{Encaps}} + q_{\text{RO}} + 2t \cdot q_{\text{RO}}}{2^\lambda} + \text{Adv}_{\text{SI}}^{\text{MT}}(\mathcal{B}_3) + \text{Adv}_{\text{ZK}}^{\text{MT}}(\mathcal{B}_4) + \text{Adv}_{\text{Ext}}^{\text{MT}}(\mathcal{B}_5). \end{aligned}$$

Proof. Apply Lemmas 6 to 8 in reverse order to bound $\text{Adv}_{\text{IND-CCA}}^{\text{UKEM}_{\text{FOMO}}}(\mathcal{A})$. Notice additionally that when the correctness errors are dealt with by using the reduction from Lemma 7, they do not appear later on, hence we do not need to count them again.

Immediately, the UPKE scheme from [ABL25a] in Fig. 3, along with our improvements, can be compiled into an IND-CCA-secure UKEM, which we present in Fig. 16.

Corollary 3. *For the parameters under which ΔPCE , LWE^h and DSIS^h are assumed hard, the $\text{UKEM}_{\text{FOMO}}$ obtained from the UPKE scheme in Fig. 3 by applying the transformations from Figs. 13 to 15 in order, the latter with MT from Fig. 9, is correct and IND-CCA-secure in the ROM. Concretely, for any PPT algorithm $\mathcal{A}$ against the IND-CCA game for $\text{UKEM}_{\text{FOMO}}$ that creates t nodes through q_{Encaps} queries to OEncaps and q_{Decaps} queries to ODecaps , and makes q_{RO} queries to its random oracles, there are PPT algorithms $\mathcal{B}_i$ for $i = 1, \dots, 5$ such that*

$$\begin{aligned} \text{Adv}_{\text{IND-CCA}}^{\text{UKEM}_{\text{FOMO}}}(\mathcal{A}) &\leq 2t^2 \cdot \text{Adv}_{\Delta\text{PCE}}(\mathcal{B}_1) + t^2 \cdot \text{Adv}_{\text{LWE}^h}(\mathcal{B}_2) + t \cdot \text{Adv}_{\text{DSIS}^h}(\mathcal{B}_3) \\ &\quad + \frac{6t + q_{\text{RO}}}{2^\delta} + \frac{q_{\text{Decaps}}}{2^\gamma} + \frac{q_{\text{Encaps}} + (3t + 1)q_{\text{RO}}}{2^\lambda} + \text{Adv}_{\text{ZK}}^{\text{MT}}(\mathcal{B}_4) + \text{Adv}_{\text{Ext}}^{\text{MT}}(\mathcal{B}_5), \end{aligned}$$

where $\delta, \gamma \in \mathbb{R}$ are the largest such that the dual-Regev PKE from Fig. 1 is δ -correct and γ -spread.

Proof. Use Theorem 3 with Lemmas 4 and 5 and Proposition 3. There is no alternative setup in MT, hence $\text{Adv}_{\text{SI}}^{\text{MT}}(\mathcal{B}) = 0$.

Remark 4. It was shown for example in [HPS23, App. B.2 & B.3] that $\delta(\lambda)$ and $\gamma(\lambda)$ are such that $2^{-\delta}$ and $2^{-\gamma}$ are negligible for LWE-based PKE constructions parametrised by λ .

5.4 Proof of Lemma 5

Proof. We use the ROM to decouple up from the honest update material. Then we replace the leaked updated keys with fresh pairs, and we are left with the IND-CPA game. All algorithms have access to $\text{UPKE} = (\text{KGen}, \text{Enc}, \text{Dec}, \text{UpdPk}, \text{UpdSk})$ and its parameters.

$\text{Game}_0(\lambda)$: This game is the Fork-IND-CPA game in Fig. 11.

KGen (pp) return (pk, sk) $\leftarrow$ UPKE.KGen(1^λ) Encaps (pp, pk) msg $\leftarrow$ $\$ \{0, 1\}^\lambda$; kk $\leftarrow$ H _{kk} (pk, msg) $\rho_{\text{ctxt}} \leftarrow$ H _{ctxt} (pk, msg); $\rho_{\text{up}} \leftarrow$ H _{up} (pk, msg) ctxt $\leftarrow$ UPKE.Enc(pk, msg; ρ_{ctxt}) (pk', $\_$) $\leftarrow$ UPKE.UpdPk(pk; ρ_{up}) mt $\leftarrow$ MT.MTag(pp _{mt} , pk, pk', UPKE.H(ρ_{up})) return (kk, ctxt, pk', mt) Vf (pp, pk, pk', mt) return MT.Vf(pp _{mt} , pk, pk', mt)	Setup (1^λ) pp _{mt} $\leftarrow$ MT.Setup(1^λ) return pp := (pp _{mt} , H _{kk} , H _{ctxt} , H _{up}) Decaps (pp, sk, ctxt, pk, pk') msg $\leftarrow$ UPKE.Dec(sk, ctxt) kk $\leftarrow$ H _{kk} (pk, msg) $\rho_{\text{ctxt}} \leftarrow$ H _{ctxt} (pk, msg); $\rho_{\text{up}} \leftarrow$ H _{up} (pk, msg) ctxt' $\leftarrow$ UPKE.Enc(pk, msg; ρ_{ctxt}) ($\tilde{\text{pk}}$, up) $\leftarrow$ UPKE.UpdPk(pk; ρ_{up}) assert ctxt = ctxt' and pk' = $\tilde{\text{pk}}$ sk' $\leftarrow$ UPKE.UpdSk(sk, up) return (kk, sk')
---	---

Figure 16: Our construction of an updatable key encapsulation mechanism based on the UPKE scheme in Fig. 3 and MT from Fig. 9, using the random oracles H: $\{0, 1\}^\lambda \rightarrow \mathcal{O}_n(\mathbb{Z})$ implicit in UPKE and H_{kk}, H_{ctxt}, H_{up}: $\{0, 1\}^* \rightarrow \{0, 1\}^\lambda$.

OChal(msg ₀ , msg ₁ , (dup _i) _{i=1} ^T) assert $T \leq t$ and $\forall i$: dup _i valid for $i = 1, \dots, T$ do pk _i $\leftarrow$ UpdPk _{CU} (pk _{i-1} , dup _i); sk _i $\leftarrow$ UpdSk _{CU} (sk _{i-1} , dup _i) ctxt $\leftarrow$ Enc(pk _T , msg _b); (pk, sk) $\leftarrow$ KGen(1^λ) up $\leftarrow$ Enc(pk _T , $\rho \leftarrow \$ \{0, 1\}^\lambda$); return (ctxt, pk', sk', up)

Figure 17: Proof of Lemma 5: Game₄.

Game₁(λ): Recall that UPKE has its own random oracle UPKE.H: $\{0, 1\}^* \rightarrow \mathcal{O}_n(\mathbb{Z})$ also simulated by the game, which maps seeds to uniformly random update material. This game is the same as **Game₀**, except that if $\mathcal{A}$ ever queries any seed $(\rho_i)_{i=1}^{q_{\text{Chal}}}$ to UPKE.H used in the q_{Chal} honest updates at the end of each OChal oracle call, the game sets a **bad** flag. These two games are the same from $\mathcal{A}$'s perspective, as there is no change in behaviour, and

$$\text{Adv}_{\text{Game}_1}(\mathcal{A}) = \text{Adv}_{\text{Game}_0}(\mathcal{A}).$$

Game₂(λ): This game is the same as **Game₁**, except that if **bad** is ever set, this game aborts with a free win for $\mathcal{A}$. The advantage of $\mathcal{A}$ cannot decrease and we have

$$\text{Adv}_{\text{Game}_1}(\mathcal{A}) \leq \text{Adv}_{\text{Game}_2}(\mathcal{A}).$$

From here on, we track the event that **bad** := 1.

Game₃(λ): This game is the same as **Game₂**, except that the update material for the honest update in the i -th OChal oracle call is no longer derived from a seed ρ_i by using UPKE.H, but sampled uniformly at random as $\mathbf{O}_i \leftarrow \$ \mathcal{O}_n(\mathbb{Z})$ instead. If **bad** is never set, then these two games are indistinguishable to $\mathcal{A}$ in the ROM, since ρ_i and $\mathbf{O}_i$ are independent. If **bad** was set, then $\mathcal{A}$ wins in both games, hence

$$\text{Adv}_{\text{Game}_3}(\mathcal{A}) = \text{Adv}_{\text{Game}_2}(\mathcal{A}).$$

Game₄(λ): This game in Fig. 17 is the same as **Game₃**, except that in each **OChal** oracle call we have $(pk', sk') \leftarrow \text{KGen}(1^\lambda)$ instead of updating honestly. If **bad** was set, then $\mathcal{A}$ wins in both games. Otherwise there exists a PPT algorithm $\mathcal{B}_1$ such that

$$|\text{Adv}_{\text{Game}_4}(\mathcal{A}) - \text{Adv}_{\text{Game}_3}(\mathcal{A})| \leq \text{Adv}_{\text{UF}}^{\text{UPKE}}(\mathcal{B}_1).$$

Indeed, the reduction $\mathcal{B}_1$ runs $\text{pp} \leftarrow \text{Setup}(1^\lambda)$, samples its own challenge bit b and then sets pk_0 to the public key it gets as input. When **OChal** is called, $\mathcal{B}_1$ updates pk_0 with the provided material and records it, then encrypts msg_b with the new key and calls its **OChal** oracle. It receives back (pk, sk) , which it forwards to $\mathcal{A}$ together with up , which is an encryption of a random bit-string under the public key updated with the provided update material. If **bad** is set, then $\mathcal{A}$ wins in both games and so $\mathcal{B}_1$ returns a uniformly random bit. Otherwise, $\mathcal{B}_1$ simulates either **Game₄** or **Game₃**, depending on whether its challenger's bit was 0 or 1, respectively.¹²

Game₅(λ): From this game on, perform a variant of the usual IND-CPA reduction for the hollow dual-Regev PKE with q_{Chal} encryption oracle calls by replacing (pk_0, sk_0) with random values using Hollow DSIS and replacing all ciphertexts ctxt with random values using Hollow LWE. If **bad** was set, then $\mathcal{A}$ wins in both games. Otherwise we have a PPT algorithm $\mathcal{B}_2$, the above reduction, such that

$$|\text{Adv}_{\text{Game}_5}(\mathcal{A}) - \text{Adv}_{\text{Game}_4}(\mathcal{A})| \leq \text{Adv}_{\text{IND-CPA}}^{\text{PKE}}(\mathcal{B}_2).$$

Since **Game₅** is independent of the challenge bit, the advantage of $\mathcal{A}$ in winning it is bounded by the probability of **bad** being set, which in **Game₅** is bounded from above by $q_{\text{RO}} \cdot 2^{-\lambda}$ using the union bound. $\square$

5.5 Proof of Lemma 6

Proof. Correctness is immediate from UPKE correctness. We turn the IND-CPA^+ challenger for our UKEM into an adversary against $\text{Game}_{\text{Fork-IND-CPA}}^{\text{UPKE}}$. All algorithms have access to $\text{UPKE} = (\text{KGen}, \text{Enc}, \text{Dec}, \text{UpdPk}, \text{UpdSk})$ and its parameters.

Game₀(λ): This game is the IND-CPA^+ game in Fig. 4.

Game₁(λ): This game is the same as **Game₀**, except that the game aborts if there is a correctness error. This is to ensure that all nodes in the experiment remain full nodes. There can be 1 error due to **KGen** failing, and $4q_{\text{Encaps}} + 4$ errors due to encapsulation failing. Indeed, in both the **OEncaps** calls and the **OMChal** call, the challenger calls **HonestNode**, which first runs **Encaps** and then runs **Decaps** (as there are no half-nodes). **Encaps** may fail by calling UPKE.Enc or UPKE.UpdPk and **Decaps** may fail by calling UPKE.Dec or UPKE.UpdSk . Observe that there are no correctness errors due to **OUpd** calls, since the challenger by CU-friendliness uses $\text{UPKE.UpdPk}_{\text{CU}}$ and $\text{UPKE.UpdSk}_{\text{CU}}$ to apply chosen updates. Thus, by δ -correctness of UPKE and the union bound, the probability of a correctness error occurring in the experiment is

$$|\text{Adv}_{\text{Game}_1}(\mathcal{A}) - \text{Adv}_{\text{Game}_0}(\mathcal{A})| \leq \Pr[\text{cor. error}] \leq \frac{4q_{\text{Encaps}} + 5}{2^\delta},$$

By the fundamental lemma of game playing, we assume there are no correctness errors from now on. For simpler notation, we omit this in pseudocode.

Game₂(λ): This game in Fig. 18 is the same as **Game₁**, except that the game samples $\hat{i}$ uniformly at random from $[0, t - 1]$ where t is the allowed number of nodes, and aborts if $\hat{i}$ is not the last node on the path from the root to the challenge node i^* not created by an **OUpd** oracle query. That is, the game tries to guess either the challenge node (if it is the root or of type **enc**) or the first ancestor of the challenge node that is of type **enc** (all its descendants including the challenge node are of type **dec**) in advance, and aborts if it fails. We have

$$\text{Adv}_{\text{Game}_1}(\mathcal{A}) \leq t \cdot \text{Adv}_{\text{Game}_2}(\mathcal{A}).$$

We assume from now on the game has correctly guessed $\hat{i}$.

¹² Observe that the game never uses the provided sk_0 , hence our reduction also works for *weak update freshness*, where sk_0 is never provided. Our multi-key scheme only satisfies this weaker property, but will nevertheless be Fork-IND-CPA secure.

Game ₂ (λ)	OMChal(i)
$\hat{i} \leftarrow \$ [0, t-1]; \text{pp} \leftarrow \text{Setup}(1^\lambda)$	if $i = 0$ or $\text{typ}_i = \text{enc}$: assert $i = \hat{i}$
$(\text{pk}_0, \text{sk}_0) \leftarrow \text{KGen}(\text{pp})$	else : assert i descends from $\hat{i}$
$(\text{par}_0, \text{rev}_0, j, \text{typ}_0) := (\perp, 0, 0, \perp)$	and $\hat{i}$ first ancestor with $\text{typ}_{\hat{i}} = \text{enc}$
$(i^*, \text{ctxt}^*) := (\perp, \perp); b \leftarrow \$ \{0, 1\}$	assert $i^* = \perp; i^* := i$
$b' \leftarrow \mathcal{A}^{\text{OEncaps}, \text{ORev}, \text{OMChal}, \text{OUpd}}(\text{pp}, \text{pk}_0)$	$\text{kk}_0 \leftarrow \$ \{0, 1\}^\lambda$
$S := \text{ChalSet}(i^*)$	$(\text{kk}_1, \text{ctxt}^*, \text{up}) \leftarrow \text{HonestNode}(i)$
return $b = b'$ and $\forall j \in S: \text{rev}_j = 0$	return $(\text{kk}_b, \text{ctxt}^*, \text{pk}_j, \text{up})$

Figure 18: Proof of Lemma 6: Game₂.

HonestNode(i)
assert $\text{pk}_i \neq \perp; j := j + 1; (\text{kk}, \text{ctxt}, \text{pk}_j, \text{up}) \leftarrow \text{Encaps}(\text{pp}, \text{pk}_i)$
$(\text{par}_j, \text{rev}_j, \text{typ}_j) := (i, 0, \text{enc}); (\_, \text{sk}_j) \leftarrow \text{Decaps}(\text{pp}, \text{sk}_i, \text{ctxt}, \text{pk}_i, \text{up})$
if $j = \hat{i}$: $(\text{pk}_j, \text{sk}_j) \leftarrow \text{KGen}(\text{pp});$ return $(\text{kk}, \text{ctxt}, \text{up})$

Figure 19: Proof of Lemma 6: Game₅.

Game₃(λ): This game is the same as Game₂ if $\hat{i} = 0$. If $\hat{i} \neq 0$, then it sets a **bad** flag and aborts with a free win if $\mathcal{A}$ ever queries UPKE.H on the randomness ρ encrypted in **up** in node i^* . For simpler notation, we omit this in pseudocode. The advantage cannot decrease and we have

$$\text{Adv}_{\text{Game}_2}(\mathcal{A}) \leq \text{Adv}_{\text{Game}_3}(\mathcal{A}).$$

Game₄(λ): This game is the same as Game₃, except that this **up** in node i^* encrypts a uniformly random bit-string that is unrelated to **dup** used to update the keys to $(\text{pk}_{i^*}, \text{sk}_{i^*})$ and $\text{dup} \leftarrow \$ \mathcal{UP}$ instead. If **bad** is never set, then Game₃ is indistinguishable from Game₄ in the ROM, whereas if **bad** is set, then $\mathcal{A}$ wins in both games. Hence

$$\text{Adv}_{\text{Game}_3}(\mathcal{A}) = \text{Adv}_{\text{Game}_4}(\mathcal{A}).$$

Game₅(λ): This game in Fig. 19 is the same as Game₄, except that when node $\hat{i}$ is about to be created (either as the root or by an OEncaps oracle query), $(\text{pk}_{\hat{i}}, \text{sk}_{\hat{i}})$ are freshly sampled with KGen. If $\hat{i} = 0$ is the root node, this game is identical to Game₂, otherwise we directly obtain a PPT algorithm $\mathcal{B}_1$ such that

$$|\text{Adv}_{\text{Game}_5}(\mathcal{A}) - \text{Adv}_{\text{Game}_4}(\mathcal{A})| \leq \text{Adv}_{\text{UF}}^{\text{UPKE}}(\mathcal{B}_1) + q_{\text{RO}} \cdot 2^{-\lambda}.$$

Indeed, $\mathcal{B}_1$ sets $(\text{pk}_0, \text{sk}_0)$ to the key-pair it receives as input, and then plays the game as $\mathcal{A}$ expects, but records all adversarially chosen and honest update material. When the node i^* is created, $\mathcal{B}_1$ queries its OChal oracle with the update material $(\text{dup}_i)_{0 \rightarrow i^*}$ on the direct path from the root node to node i^* , receives back a key-pair and sets it as $(\text{pk}_{i^*}, \text{sk}_{i^*})$. If the update freshness challenge bit was $b = 1$, then $\mathcal{B}_1$ simulates Game₄, otherwise it simulates Game₅.¹³ If **bad** is never set, then **up** in i^* is independent of $(\text{pk}_{i^*}, \text{sk}_{i^*})$, hence $\mathcal{B}_1$ simulates both games perfectly depending on its challenger's bit and $\mathcal{A}$ distinguishing Game₄ from Game₅ leads to $\mathcal{B}_1$ winning the update freshness game. If **bad** is set, then $\mathcal{A}$ wins in both games, so $\mathcal{B}_1$ returns a uniformly random bit. The probability of **bad** being set in Game₅ is bounded from above by $q_{\text{RO}} \cdot 2^{-\lambda}$ using the union bound.

¹³ This is indeed where weak update freshness proves insufficient. To simulate secret-key compromise outside of the sub-tree rooted at i^* , the reduction needs sk_0 . We believe this limitation may be bypassed by proving this step from the underlying hardness assumptions of the concrete scheme.

$\mathcal{B}_2(\text{pp}, \text{pk}_i)$ $\hat{i} \leftarrow \{0, t-1\}; (\text{pk}_0, \text{sk}_0) \leftarrow \text{KGen}(\text{pp})$ if $\hat{i} = 0$: $(\text{pk}_0, \text{sk}_0) := (\text{pk}_i, \perp)$ $(\text{par}_0, \text{rev}_0, j, \text{typ}_0) := (\perp, 0, 0, \perp)$ $(i^*, \text{ctxt}^*, \text{sk}_i) := (\perp, \perp, \perp); b' \leftarrow \{0, 1\}$ $\_ \leftarrow \mathcal{A}_5^{\text{OEncaps}, \text{ORev}, \text{OMChal}, \text{OUpd}}(\text{pp}, \text{pk}_0)$ assert $\forall j \in \text{ChalSet}(i^*) : \text{rev}_j = 0$ return b'	$\text{OEncaps}(i)$ if $\text{sk}_i = \perp$: $\text{kk} \leftarrow \{0, 1\}^\lambda; j := j+1; \rho_{0,1}^{(i)} \leftarrow \{0, 1\}^{ \text{pk} +\lambda}$ $(\text{ctxt}, \text{pk}', \text{sk}', \text{up}) \leftarrow \text{OChal}(\rho_0^{(i)}, \rho_1^{(i)}, (\text{dup}_{i \rightarrow i^*}))$ $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}', \text{sk}', i, 0, \text{enc})$ else : $(\text{kk}, \text{ctxt}, \text{up}) \leftarrow \text{HonestNode}(i)$ if $j = i$: $(\text{pk}_j, \text{sk}_j) := (\text{pk}_i, \perp)$ return $(\text{kk}, \text{ctxt}, \text{pk}_j, \text{up})$
$\text{OUpd}(i, \text{dup})$ assert dup valid and $\text{pk}_i \neq \perp$ $(\text{pk}', \_) \leftarrow \text{UPKE.UpdPk}_{\text{CU}}(\text{pk}_i, \text{dup})$ if $\text{sk}_i = \perp$: $\text{sk}' := \perp$ else : $\text{sk}' \leftarrow \text{UPKE.UpdSk}_{\text{CU}}(\text{sk}_i, \text{dup})$ $j := j+1$ $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}', \text{sk}', i, 0, \text{dec})$ return $\perp$	$\text{OMChal}(i)$ assert i as in $\text{Game}_2; i^* := i; \rho_0^{(i)} \leftarrow \{0, 1\}^{ \text{pk} +\lambda}$ $\rho_1^{(i)} := (\text{pk}_{i^*}, \text{msg}_{i^*} \leftarrow \{0, 1\}^\lambda)$ $(\text{ctxt}^*, \text{pk}', \text{sk}', \text{up}) \leftarrow \text{OChal}(\rho_0^{(i)}, \rho_1^{(i)}, (\text{dup}_{i \rightarrow i^*}))$ $\text{kk}^* \leftarrow \{0, 1\}^\lambda; j := j+1$ $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}', \text{sk}', i^*, 0, \text{enc})$ return $(\text{kk}^*, \text{ctxt}^*, \text{pk}_j, \text{up})$
$\text{ORev}(i)$ assert $\text{sk}_i \neq \perp; \text{rev}_i := 1; \text{return}$ sk_i	$\mathcal{A}_5.\text{H}_{\text{kk}}(\text{msg})$ for $i \geq \hat{i}, b \in \{0, 1\}$ if $\text{msg} = \rho_b^{(i)} : b' := b; \text{abort}$ return $\text{H}_{\text{kk}}(\text{msg})$

Figure 20: Proof of Lemma 6: reduction from Fork-IND-CPA to Game_5 .

We conclude by providing a reduction $\mathcal{B}_2$ in Fig. 20 that, given any algorithm $\mathcal{A}_5$ playing Game_5 , wins the Fork-IND-CPA security game in Definition 22 with advantage at least that of $\mathcal{A}_5$. Let win be the event that $\mathcal{A}_5$ ever queries the random oracle H_{kk} on the randomness $\rho = (\text{pk}_i, \text{msg}_i)$ used to produce any ctxt through an OEncaps or OMChal call on either $\hat{i}$ or any of its **dec**-type children. Observe that if win never happens, then the challenge transcript $(\text{kk}', \text{ctxt}, \text{pk}', \text{up})$ is independent of the challenge bit in the ROM. Conversely, if win happens, then $\mathcal{A}_5$ has either learned the challenge bit or correctly guessed a random seed. Our reduction will thus, in short, plant its Fork-IND-CPA challenges into Game_5 such that win would allow it to learn its challenge bit, observe $\mathcal{A}_5$'s calls to H_{kk} and simply output the bit corresponding to $\mathcal{A}_5$'s query if win happens, and a random bit otherwise. For this reason the win event can be regarded as granting a 'free win' to $\mathcal{A}_5$, though of course $\mathcal{B}_2$ halts as soon as it happens.

Concretely, $\mathcal{B}_2$ plays the game as $\mathcal{A}_5$ expects, except that it takes as input (pp, pk_i) from its Fork-IND-CPA challenger, sets $\text{sk}_i := \perp$, and uses the oracles provided by its Fork-IND-CPA challenger to simulate the oracle calls of $\mathcal{A}_5$ whenever it does not know sk_i (necessarily $i \geq \hat{i}$). Notably, $\mathcal{B}_2$ does not run OMChal when queried by $\mathcal{A}_5$, but instead queries its OChal oracle on $(\rho_0^{(i^*)}, \rho_1^{(i^*)}, (\text{dup}_{i \rightarrow i^*}))$, where $\rho_0^{(i^*)}$ is the random seed of kk_0 , $\rho_1^{(i^*)} = (\text{pk}_{i^*}, \text{msg}_{i^*})$ is the seed of kk_1 , and $(\text{dup}_{i \rightarrow i^*})$ is the (possibly empty) list of the update material provided by $\mathcal{A}_5$ through its OUpd oracle calls on the direct path from $\hat{i}$ to i^* in the experiment tree. It receives back ctxt^* and immediately calls OLeak on the same session with an empty update set, denoted by $\text{OLeak}(\_, \emptyset)$, to get $(\text{pk}', \text{sk}', \text{up})$. It then creates a new child of i^* with this material, samples a random $\text{kk}' \leftarrow \{0, 1\}^\lambda$ and returns $(\text{kk}', \text{ctxt}^*, \text{pk}', \text{up})$ to $\mathcal{A}_5$.

Observe that $\mathcal{A}_5$ is not allowed to reveal sk_i or any secret keys obtained from sk_i via a chain of OUpd calls, otherwise the game aborts with a random bit. These secret keys are all unknown to $\mathcal{B}_2$ and hence recorded as $\perp$, and these are the only secret keys recorded as $\perp$. Thus the first OEncaps query by $\mathcal{A}_5$ that creates a child of $\hat{i}$ or any such chain of **dec**-type nodes is translated by $\mathcal{B}_2$ into a OChal call to its own Fork-IND-CPA challenger, which yields to $\mathcal{B}_2$ a new secret key for that node. The reduction then uses this secret key and its updates to answer ORev queries that $\mathcal{A}_5$ may make thereafter.

```

for  $i \in \{\text{kk}, \text{ctxt}, \text{up}\}$ :  $\mathcal{A}.\text{H}_i(\text{msg}')$ 


---


parse  $(\text{pk}, \text{msg}) \leftarrow \text{msg}'$ ; for  $j \in \{\text{kk}, \text{ctxt}, \text{up}\}$  if  $\text{msg}' \notin \mathcal{Q}_j$ :  $\mathcal{Q}_j[\text{msg}'] \leftarrow \$ \{0, 1\}^\lambda$ 
 $\text{ctxt} \leftarrow \text{UPKE}.\text{Enc}(\text{pk}, \text{msg}; \mathcal{Q}_{\text{ctxt}}[\text{msg}'])$ ;  $(\text{pk}', \_) \leftarrow \text{UPKE}.\text{UpdPk}(\text{pk}; \mathcal{Q}_{\text{up}}[\text{msg}'])$ 
 $\text{dup} \leftarrow \text{UPKE}.\text{H}(\mathcal{Q}_{\text{up}}[\text{msg}'])$ ;  $\mathcal{Q}^* := \mathcal{Q}^* \cup \{(\text{pk}, \text{pk}', \text{dup}, \text{msg}, \text{ctxt}, \mathcal{Q}_{\text{kk}}[\text{msg}'])\}$ 
return  $\mathcal{Q}_i[\text{msg}']$ 

```

Figure 21: Proof of Lemma 7: Game_2 .

Notice that Game_5 and $\mathcal{B}_2$'s simulation are identical in the view of $\mathcal{A}_5$, except for the win event trigger in $\mathcal{B}_2$, which is where the Fork-IND-CPA challenges are planted. If $\mathcal{A}_5$ never calls any of the random oracles on $(\text{pk}_i, \text{msg}_i)$ as described in Game_5 and so win does not happen, then the challenge transcript $(\text{kk}', \text{ctxt}, \text{pk}', \text{up})$ is independent of a real challenge $(\text{kk}_b, \text{ctxt}, \text{pk}', \text{up})$ (for Game_5 's b) in the ROM, hence $\mathcal{A}_5$ has no advantage in winning Game_5 . Accordingly, $\mathcal{B}_2$ in this case returns a random bit b' to its challenger. Otherwise, if $\mathcal{A}_5$ queries its H_{kk} on any $\rho_b^{(i)}$ for some $b \in \{0, 1\}$, this means that it wins Game_5 . However, $\mathcal{B}_2$ then returns b to its Fork-IND-CPA challenger and halts, winning Fork-IND-CPA, unless $\mathcal{A}_5$ queried any $\rho_{1-b}^{(i)}$ (the reduction does not know the correct b), which can happen with probability at most $q_{\text{RO}} \cdot 2^{-\lambda}$. Thus, using the win event to distinguish, we have that

$$\text{Adv}_{\text{Game}_5}(\mathcal{A}_5) \leq \Pr[\text{win}] \leq \text{Adv}_{\text{Fork-IND-CPA}}^{\text{UPKE}}(\mathcal{B}_2) + q_{\text{RO}} \cdot 2^{-\lambda}. \quad \square$$

5.6 Proof of Lemma 7

Proof. Correctness is immediate. The proof proceeds via a sequence of hybrid games and is a typical FO-style proof. We will turn an honest IND-CCA⁻ challenger playing a game against $\mathcal{A}$ into an algorithm for IND-CPA⁺. All algorithms have access to UKEM = (Setup, KGen, Encaps, Decaps), the underlying UPKE = (KGen, Enc, Dec, UpdPk, UpdSk), and their parameters.

Game₀(λ): This game is the IND-CCA⁻ game in Fig. 4.

Game₁(λ): This game is the same as **Game₀**, except that whenever there is a correctness error in any of the oracles, the game aborts and grants $\mathcal{A}$ a free win. There can be 1 error due to KGen failing, $4q_{\text{Encaps}}$ errors due to the OEncaps calls, $6q_{\text{Decaps}}$ errors due to the ODecaps⁻ calls and 4 errors due to the OMChal call. Indeed, each OEncaps and OMChal call results in the challenger running HonestNode, and thus first Encaps and then Decaps. Encaps can fail when calling UPKE.Enc or UPKE.UpdPk. Conditioned on Encaps succeeding, the re-encryption and re-update steps (both **assert** instructions) always succeed, thus Decaps can fail when calling UPKE.Dec and UPKE.UpdSk. Things are different for ODecaps⁻ calls, which run Decaps but not necessarily on honestly generated Encaps output (the challenger does not call HonestNode here). Decaps can fail by calling UPKE.Dec, UPKE.Enc, the re-encryption check, UPKE.UpdPk, the re-update check, or UPKE.UpdSk. This accounts for honest correctness errors, but additionally $\mathcal{A}$ may intentionally query its H_{ctxt} oracle on many (pk, msg) until it finds a pair such that UPKE.Dec would fail on $\text{ctxt} := \text{UPKE.Enc}(\text{pk}, \text{msg}; \text{H}_{\text{ctxt}}(\text{pk}, \text{msg}))$. The probability of a correctness error occurring in the experiment is by δ -correctness of UPKE and the union bound then

$$|\text{Adv}_{\text{Game}_0}(\mathcal{A}) - \text{Adv}_{\text{Game}_1}(\mathcal{A})| \leq \Pr[\text{cor. error}] \leq \frac{4q_{\text{Encaps}} + 6q_{\text{Decaps}} + 5 + q_{\text{RO}}}{2^\delta}.$$

By the fundamental lemma of game playing, we assume there are no correctness errors from now on. Note that the numerator is bounded from above by $6t + q_{\text{RO}}$. For simpler notation, we omit this in pseudocode.

Game₂(λ): This game in Fig. 21 is the same as **Game₁** except that the random oracles H_{kk} , H_{ctxt} , and H_{up} that $\mathcal{A}$ can access record each query in a table $\mathcal{Q}^*$. This is denoted by the prefix $\mathcal{A}.\text{H}_i$. More precisely, whenever $\mathcal{A}$ queries any of its random oracles with (pk, msg) , **Game₂** simulates $\text{kk} \leftarrow \text{H}_{\text{kk}}(\text{pk}, \text{msg})$,

$\text{ODecaps}^-(i, \text{ctxt}, \text{pk}')$
$\text{assert } \text{sk}_i \neq \perp; \text{ if } i^* \neq \perp: \text{ assert } \text{ctxt}^* \neq \text{ctxt} \text{ or } \text{pk}_{i^*} \neq \text{pk}_i$ $(\text{kk}, \text{sk}') \leftarrow \text{Decaps}(\text{pp}, \text{sk}_i, \text{ctxt}, \text{pk}_i, \text{pk}'); \text{ if } \text{kk} = \perp: \text{ return } \perp$ $\text{if } \nexists (\text{up}', \text{msg}', \text{kk}') \text{ s.t. } (\text{pk}_i, \text{pk}', \text{up}', \text{msg}', \text{ctxt}, \text{kk}') \in \mathcal{Q}^*: \text{ bad} := 1$ $j := j + 1; (\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}', \text{sk}', i, 0, \text{dec}); \text{ return } \text{kk}$

Figure 22: Proof of Lemma 7: Game_3 .

$\text{ODecaps}^-(i, \text{ctxt}, \text{pk}')$
$\text{assert } \text{sk}_i \neq \perp; \text{ if } i^* \neq \perp: \text{ assert } \text{ctxt}^* \neq \text{ctxt} \text{ or } \text{pk}_{i^*} \neq \text{pk}_i$ $(\text{kk}, \text{sk}') \leftarrow \text{Decaps}(\text{pp}, \text{sk}_i, \text{ctxt}, \text{pk}_i, \text{pk}'); \text{ if } \text{kk} = \perp: \text{ return } \perp$ $\text{if } \nexists (\text{up}', \text{msg}', \text{kk}') \text{ s.t. } (\text{pk}_i, \text{pk}', \text{up}', \text{msg}', \text{ctxt}, \text{kk}') \in \mathcal{Q}^*: \text{ bad} := 1 \text{ and abort}$ $j := j + 1; (\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}', \text{sk}', i, 0, \text{dec}); \text{ return } \text{kk}$

Figure 23: Proof of Lemma 7: Game_4 .

$\rho_{\text{ctxt}} \leftarrow \text{H}_{\text{ctxt}}(\text{pk}, \text{msg})$, and $\rho_{\text{up}} \leftarrow \text{H}_{\text{up}}(\text{pk}, \text{msg})$. It returns the value queried to $\mathcal{A}$ and records the tuple $(\text{pk}, \text{pk}', \text{dup}, \text{msg}, \text{ctxt}, \text{kk})$ to $\mathcal{Q}^*$, where $\text{ctxt} \leftarrow \text{UPKE.Enc}(\text{pk}, \text{msg}; \rho_{\text{ctxt}})$, $\text{pk}' \leftarrow \text{UPKE.UpdPk}_{\text{CR}}(\text{pk}, \rho_{\text{up}})$ and dup is generated from ρ_{up} using UPKE.H . Since there is no change in behaviour, Game_1 and Game_2 are clearly the same from $\mathcal{A}$'s perspective, and

$$\text{Adv}_{\text{Game}_2}(\mathcal{A}) = \text{Adv}_{\text{Game}_1}(\mathcal{A}).$$

Game₃(λ): This game in Fig. 22 is the same as Game_2 , except that if $\mathcal{A}$ ever queries $(i, \text{ctxt}, \text{pk}')$ to ODecaps^- for which there is no $(\text{dup}, \text{msg}, \text{kk})$ such that $(\text{pk}_i, \text{pk}', \text{dup}, \text{msg}, \text{ctxt}, \text{kk}) \in \mathcal{Q}^*$, but the ciphertext still decapsulates successfully, this game sets a flag bad . Since there is no change in behaviour, Game_2 and Game_3 are clearly the same from $\mathcal{A}$'s perspective, and

$$\text{Adv}_{\text{Game}_3}(\mathcal{A}) = \text{Adv}_{\text{Game}_2}(\mathcal{A}).$$

Game₄(λ): This game in Fig. 23 is the same as Game_3 , except that this game's partial decapsulation oracle also aborts (that is, returns $\perp$) when setting the bad flag. The games Game_3 and Game_4 are identical until the bad flag is set, therefore by [BR06, Lem. 2] the advantage of $\mathcal{A}$ in distinguishing the two games is at most $\Pr[\text{bad} = 1]$. The bad flag is set exactly when $\mathcal{A}$ queries ODecaps^- with $(i, \text{ctxt}, \text{pk}')$ that passes the re-encryption and the re-update checks in Decaps , but has not queried any of the random oracles with $(\text{pk}_i, \text{msg})$ for some $\text{msg} \in \{0, 1\}^\lambda$. In the ROM, random bit-strings $(\rho_{\text{ctxt}}, \rho_{\text{up}})$ are then independent of pk_i and this msg , thus

$$\begin{aligned}
& |\text{Adv}_{\text{Game}_4}(\mathcal{A}) - \text{Adv}_{\text{Game}_3}(\mathcal{A})| \leq \Pr[\text{bad} = 1] \\
& \leq \Pr_{\rho_{\text{ctxt}}, \rho_{\text{up}} \leftarrow \mathbb{S}\{0, 1\}^\lambda} \left[\begin{array}{l} \text{ctxt} = \text{UPKE.Enc}(\text{pk}_i, \text{msg}; \rho_{\text{ctxt}}) \text{ and} \\ \text{pk}' = \text{UPKE.UpdPk}(\text{pk}_i; \rho_{\text{up}}) \end{array} \right] \\
& \leq \Pr_{\rho_{\text{ctxt}} \leftarrow \mathbb{S}\{0, 1\}^\lambda} [\text{ctxt} = \text{UPKE.Enc}(\text{pk}_i, \text{msg}; \rho_{\text{ctxt}})] \leq q_{\text{Decaps}} \cdot 2^{-\gamma},
\end{aligned}$$

where the last inequality is due to γ -spreadness of UPKE and the union bound. From now on we may assume bad is never set.

Game₅(λ): This game in Fig. 24 is the same as Game_4 , except that the game now returns kk' from its records $\mathcal{Q}^*$ instead of kk from decapsulation. This change is purely syntactic, since $\text{kk} = \text{kk}'$ by correctness, and

$$\text{Adv}_{\text{Game}_5}(\mathcal{A}) = \text{Adv}_{\text{Game}_4}(\mathcal{A}).$$

```

ODecaps-( $i, \text{ctxt}, \text{pk}'$ )


---


assert  $\text{sk}_i \neq \perp$ ; if  $i^* \neq \perp$ : assert  $\text{ctxt}^* \neq \text{ctxt}$  or  $\text{pk}_{i^*} \neq \text{pk}_i$ 
 $(\text{kk}, \text{sk}') \leftarrow \text{Decaps}(\text{pp}, \text{sk}_i, \text{ctxt}, \text{pk}_i, \text{pk}')$ ; if  $\text{kk} = \perp$ : return  $\perp$ 
if  $\nexists (\text{dup}', \text{msg}', \text{kk}') \text{ s.t. } (\text{pk}_i, \text{pk}', \text{dup}', \text{msg}', \text{ctxt}, \text{kk}') \in \mathcal{Q}^*$ : bad := 1 and abort
 $j := j + 1$ ;  $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}', \text{sk}', i, 0, \text{dec})$ ; return  $\text{kk}'$ 

```

Figure 24: Proof of Lemma 7: Game_5 .

```

ODecaps-( $i, \text{ctxt}, \text{pk}'$ )


---


assert  $\text{sk}_i \neq \perp$ ; if  $i^* \neq \perp$ : assert  $\text{ctxt}^* \neq \text{ctxt}$  or  $\text{pk}_{i^*} \neq \text{pk}_i$ 
if  $\nexists (\text{dup}', \text{msg}', \text{kk}') \text{ s.t. } (\text{pk}_i, \text{pk}', \text{dup}', \text{msg}', \text{ctxt}, \text{kk}') \in \mathcal{Q}^*$ : bad := 1 and abort
 $\text{sk}' \leftarrow \text{UPKE.UpdSk}_{\text{CU}}(\text{sk}_i; \text{dup}')$ 
 $j := j + 1$ ;  $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}', \text{sk}', i, 0, \text{dec})$ ; return  $\text{kk}'$ 

```

Figure 25: Proof of Lemma 7: Game_6 .

$\text{Game}_6(\lambda)$: This game in Fig. 25 is the same as Game_5 , except that the game never decapsulates. Instead, if the **bad** flag is not set, it manually updates the secret key and returns kk' from its records $\mathcal{Q}^*$. The two games are the same from $\mathcal{A}$'s perspective, since in Game_5 we only ever return the result of decapsulation if it was already stored in $\mathcal{Q}^*$. Observe that Game_6 does not use its secret keys anywhere, except for tracking how they are updated, and

$$\text{Adv}_{\text{Game}_6}(\mathcal{A}) = \text{Adv}_{\text{Game}_5}(\mathcal{A}).$$

We give a reduction $\mathcal{B}$ in Fig. 26 from IND-CPA^+ to Game_6 . Let $\mathcal{A}_6$ be a PPT adversary against Game_6 . The reduction $\mathcal{B}$ is almost identical to Game_6 except that it takes (pp, pk_0) as input from the IND-CPA^+ challenger instead of generating its own key material. It uses the provided oracles in the IND-CPA^+ game to simulate Game_6 for $\mathcal{A}_6$; namely, it simulates the partial decapsulation oracle ODecaps^- using its own update oracle OUpd , so that $\mathcal{A}_6$ winning Game_6 translates to $\mathcal{B}$ winning the IND-CPA^+ game. $\mathcal{B}$ simply passes through the other oracle queries that $\mathcal{A}_6$ may make to its IND-CPA^+ challenger and returns the result. In contrast to Game_6 , $\mathcal{B}$ queries its challenger's oracle H_{kk} instead of simulating one for $\mathcal{A}_6$. In Fig. 26, we denote by $\mathcal{A}_6.\text{O}$ the oracle O available to $\mathcal{A}_6$ and simulated by $\mathcal{B}$ for $\mathcal{A}_6$, and by O the oracle O available to $\mathcal{B}$ as provided by its challenger. $\mathcal{B}$ returns $\mathcal{A}_6$'s bit.

$\mathcal{B}$ and Game_6 are clearly the same from $\mathcal{A}_6$'s perspective. The advantage of $\mathcal{A}_6$ in winning Game_6 directly translates to the advantage of $\mathcal{B}$ in winning the IND-CPA^+ game against its own challenger and we have

$$\text{Adv}_{\text{Game}_6}(\mathcal{A}_6) \leq \text{Adv}_{\text{IND-CPA}^+}^{\text{UKEM}_{\text{FO}}}(\mathcal{B}).$$

We conclude by translating this advantage to an advantage for the ‘randomised’ UKEM (instead of the ‘de-randomised’ UKEM_{FO}). The only difference between the two games is that the adversary $\mathcal{B}$ is able to obtain messages from nodes that are not in $\text{ChalSet}(i^*)$ and check whether the game was instantiated with UKEM or UKEM_{FO} (say, by recomputing a deterministic encryption). The advantage in both games is the same if $\mathcal{B}$ never queries $(\text{pk}_{i^*}, \text{msg}_{i^*})$ for the challenge message msg_{i^*} to H_{kk} , hence it is enough to fix the challenge msg_{i^*} as uniformly random at the start and abort if at any point $\text{msg} = \text{msg}_{i^*}$ when creating a new node other than i^* . This happens with probability at most $(q_{\text{RO}} + q_{\text{Encaps}}) \cdot 2^{-\lambda}$. $\square$

5.7 Proof of Lemma 8

Proof. The proof proceeds via a sequence of hybrid games, and closely follows the proof of [AFMR25, Thm. 7]. We will mould an honest IND-CCA challenger playing a game against $\mathcal{A}$ into an algorithm for IND-CCA^- .

$\mathcal{B}(\text{pp}, \text{pk}_0)$	
$(\text{par}_0, \text{rev}_0, j, \text{typ}_0) := (\perp, 0, 0, \perp); (i^*, \text{ctxt}^*) := (\perp, \perp); \text{pp}' := (\mathcal{A}_6.H_{\text{kk}}, \mathcal{A}_6.H_{\text{ctxt}}, \mathcal{A}_6.H_{\text{up}})$ $b \leftarrow \mathcal{A}_6^{\text{OEncaps}, \text{ORev}, \text{OMChal}, \text{OUdp}, \text{ODecaps}^-}(\text{pp}', \text{pk}_0); \text{ return } b$	
$\mathcal{A}_6.\text{OEncaps}(i)$	for $i \in \{\text{kk}, \text{ctxt}, \text{up}\} : \mathcal{A}_6.H_i(\text{msg}')$
$(\text{kk}, \text{ctxt}, \text{pk}_j, \text{up}) \leftarrow \text{OEncaps}(i)$	parse $(\text{pk}, \text{msg}) \leftarrow \text{msg}'; \text{kk} \leftarrow H_{\text{kk}}(\text{pk}, \text{msg})$
return $(\text{kk}, \text{ctxt}, \text{pk}_j)$	for $j \in \{\text{ctxt}, \text{up}\}$ if $\text{msg}' \notin \mathcal{Q}_j : \mathcal{Q}_j[\text{msg}'] \leftarrow \mathcal{s} \{0, 1\}^\lambda$
$\mathcal{A}_6.\text{ORev}(i)$	$\text{ctxt} \leftarrow \text{UPKE}.\text{Enc}(\text{pk}, \text{msg}; \mathcal{Q}_{\text{ctxt}}[\text{msg}'])$
return $\text{ORev}(i)$	$(\text{pk}', \_) \leftarrow \text{UPKE}.\text{UpdPk}(\text{pk}; \mathcal{Q}_{\text{up}}[\text{msg}'])$
$\mathcal{A}_6.\text{OMChal}(i)$	$\text{dup} \leftarrow \text{UPKE}.\text{H}(\mathcal{Q}_{\text{up}}[\text{msg}'])$
$(\text{kk}, \text{ctxt}^*, \text{pk}_j, \text{up}) \leftarrow \text{OMChal}(i)$	$\mathcal{Q}^* := \mathcal{Q}^* \cup \{(\text{pk}, \text{pk}', \text{dup}, \text{msg}, \text{ctxt}, \text{kk})\}$
return $(\text{kk}, \text{ctxt}^*, \text{pk}_j)$	return $(\text{kk}, \mathcal{Q}_{\text{ctxt}}[\text{msg}'], \mathcal{Q}_{\text{up}}[\text{msg}'])[i]$
$\mathcal{A}_6.\text{OUdp}(i, \text{dup})$	$\mathcal{A}_6.\text{ODecaps}^-(i, \text{ctxt}, \text{pk}')$
return $\text{OUdp}(i, \text{dup})$	if $i^* \neq \perp : \text{assert } \text{ctxt}^* \neq \text{ctxt} \text{ or } \text{pk}_i^* \neq \text{pk}_i$
	if $\exists (\text{dup}, \text{msg}, \text{kk}) \text{ s.t. } (\text{pk}_i, \text{pk}', \text{dup}, \text{msg}, \text{ctxt}, \text{kk}) \in \mathcal{Q}^* :$
	$\_ \leftarrow \text{OUdp}(i, \text{dup}); \text{ return } \text{kk}$
	return $\perp$

Figure 26: Proof of Lemma 7: reduction from IND-CPA⁺ to Game₆.

$\text{UKEM}.\text{Encaps}(\text{pp}, \text{pk})$
$\dots; \text{mt} \leftarrow \text{NIZKPoK}.\text{Sim}(\text{pp}_{\text{mt}}, \text{pk}, \text{pk}'); \text{ return } (\text{kk}, \text{ctxt}, \text{pk}', \text{mt})$

Figure 27: Proof of Lemma 8: Game₂.

Game₀(λ): This game is the IND-CCA game in Fig. 4.

Game₁(λ): This game is the same as Game₀ except that we switch to the alternative setup in MT. Since the MT construction we have in mind uses Fischlin's transform which does not require a trapdoor, we omit this step in pseudocode. Nevertheless, if an alternative setup is required, there is a reduction $\mathcal{B}_1$ such that

$$|\text{Adv}_{\text{Game}_1}(\mathcal{A}) - \text{Adv}_{\text{Game}_0}(\mathcal{A})| \leq \text{Adv}_{\text{SI}}^{\text{MT}}(\mathcal{B}_1) .$$

Game₂(λ): This game in Fig. 27 is the same as Game₁ except that we replace all member tags mt output as the last component of OEncaps and OMChal oracle queries with simulated proofs. There is a reduction $\mathcal{B}_2$ such that

$$|\text{Adv}_{\text{Game}_2}(\mathcal{A}) - \text{Adv}_{\text{Game}_1}(\mathcal{A})| \leq \text{Adv}_{\text{ZK}}^{\text{MT}}(\mathcal{B}_2) .$$

Game₃(λ): This game in Fig. 28 is the same as Game₂ except that we will extract witnesses from member tags provided by $\mathcal{A}$ to the ODecaps oracle. We will also start keeping track of half-nodes, minding that they do not occur in the IND-CCA⁻ game.

For a node index i , let cop_i be the set of indices of half-nodes with all parameters (except sk_i) the same as node i , in particular, all of these nodes are children of par_i and have pk_i as their public key. When $\mathcal{A}$ calls ODecaps on $(i, \text{ctxt}, \text{pk}, \text{mt})$, we first check whether mt verifies. If not, the oracle outputs $\perp$ as usual, otherwise we separate two cases:

- (i) If there is an **enc**-type node with pk as its public key then the game has previously simulated a proof for $x = (\text{pk}_i, \text{pk})$. Let $\hat{i}$ be the first index with $\text{pk}_{\text{par}_{\hat{i}}} = \text{pk}_i$ and additionally $\text{par}_{\hat{i}} \neq i$. If i is a half-node, the new node j will also be a half-node, hence j is added to cop_i and the game outputs $\perp$. Otherwise, if i is a full node, the game attempts to decapsulate ctxt . If successful, j is created as a full node, and the decapsulated key kk is returned to $\mathcal{A}$. Otherwise j is a half-node and is added to cop_i .

```

ODecaps( $i, \text{ctxt}, \text{pk}, \text{mt}$ )
assert  $\text{pk}_i \neq \perp$ ; if  $i^* \neq \perp$ : assert  $\text{ctxt}^* \neq \text{ctxt}$  or  $\text{pk}_{i^*} \neq \text{pk}_i$ 
assert  $\forall f(\text{pp}, \text{pk}_i, \text{pk}, \text{mt})$ ;  $j := j + 1$ 
if  $\exists i$  s.t.  $\text{pk}_i = \text{pk}$ :  $\hat{i} := 1$ st s.t.  $\text{pk}_{\text{par}_{\hat{i}}} = \text{pk}_i$  and  $\text{par}_{\hat{i}} \neq i$  // Case (i)
  if  $\text{sk}_i = \perp$ :  $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \perp, i, 0, \text{dec})$ ;  $\text{cop}_i := \text{cop}_i \cup \{j\}$ ; return  $\perp$ 
  else:  $(\text{kk}, \text{sk}) \leftarrow \text{Decaps}(\text{pp}, \text{sk}_i, \text{ctxt}, \text{pk}_i, \text{pk})$ 
    if  $(\text{kk}, \text{sk}) \neq \perp$ :  $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \text{sk}, i, 0, \text{dec})$ ; return  $\text{kk}$ 
    else:  $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \perp, i, 0, \text{dec})$ ;  $\text{cop}_i := \text{cop}_i \cup \{j\}$ ; return  $\perp$ 
else: // Case (ii)
  if  $\text{sk}_i = \perp$ :  $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \perp, i, 0, \text{dec})$ ; return  $\perp$ 
   $\text{dup} \leftarrow \text{NIZKPoK.Ext}(\text{pp}, (\text{pk}_i, \text{pk}), \text{mt})$ ; if  $\text{dup}$  does not update  $\text{pk}_i \rightarrow \text{pk}$ : abort
   $(\text{kk}, \text{sk}) \leftarrow \text{Decaps}(\text{pp}, \text{sk}_i, \text{ctxt}, \text{pk}_i, \text{pk})$ 
  if  $(\text{kk}, \text{sk}) \neq \perp$ :  $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \text{sk}, i, 0, \text{dec})$ ; return  $\text{kk}$ 
  else:  $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \perp, i, 0, \text{dec})$ ; return  $\perp$ 

```

Figure 28: Proof of Lemma 8: Game_3 . Only main changes are highlighted.

- (ii) If there is no such **enc**-type node, the game has not previously simulated a proof for $x = (\text{pk}_i, \text{pk})$. Hence the game extracts $\mathcal{A}$'s witness dup from $\mathcal{A}$'s random oracle queries by running the extractor $\text{NIZKPoK.Ext}(\text{pp}_{\text{mt}}, (\text{pk}_i, \text{pk}), \text{mt})$ of the member tag scheme. It checks whether dup updates pk_i to pk , and aborts if not. This allows us to turn $\mathcal{A}$ into an adversary against $\text{Game}_{\text{Ext}}^{\text{MT}}$. Moreover, if i is a full node, the game attempts to decapsulate ctxt , creates a full node if successful and a half-node otherwise. If i is a half-node, a half-node is created.

The above gives a natural adversary $\mathcal{B}_3$ against $\text{Game}_{\text{Ext}}^{\text{MT}}$ from Fig. 5 if Game_3 ever aborts in case (ii), thus distinguishing Game_2 from Game_3 ; that is, $\mathcal{B}_3$ is a reduction from $\text{Game}_{\text{Ext}}^{\text{MT}}$ to distinguishing Game_2 from Game_3 (using some $\mathcal{A}_{2,3}$), which can only happen by triggering the abort. Indeed, the code of $\mathcal{B}_3$ is almost the same as that of Game_3 , thus playing the game as $\mathcal{A}_{2,3}$ expects, except that instead of calling NIZKPoK.Sim we use the provided OSim oracle, instead of calling NIZKPoK.Ext we use the provided OExt oracle, and all oracle queries concerning MT are simply forwarded to $\mathcal{B}_3$'s challenger in $\text{Game}_{\text{Ext}}^{\text{MT}}$. If case (ii) ever happens and Game_3 would abort, then $\mathcal{B}_3$ has queried its OExt oracle on $x = (\text{pk}_i, \text{pk})$ and $\pi = \text{mt}$ such that mt verifies, $\pi_{\text{Sim}}[\text{pk}_i, \text{pk}] = \perp$ (that is, the proof has not been simulated for this instance), but the extracted witness w is invalid. Thus if $\mathcal{A}_{2,3}$ distinguishes Game_2 from Game_3 by triggering the abort, $\mathcal{B}_3$ has won $\text{Game}_{\text{Ext}}^{\text{MT}}$. Since changes other than aborting in case (ii) are internal and thus the same in $\mathcal{A}_{2,3}$'s view, we have

$$|\text{Adv}_{\text{Game}_3}(\mathcal{A}_{2,3}) - \text{Adv}_{\text{Game}_2}(\mathcal{A}_{2,3})| \leq \text{Adv}_{\text{Ext}}^{\text{MT}}(\mathcal{B}_3).$$

By the fundamental lemma of game playing, we assume from now on that we do not abort in case (ii).

We conclude with a reduction $\mathcal{B}$ in Fig. 29 from IND-CCA^- to Game_3 . Indeed, let $\mathcal{A}_3$ be a PPT adversary against Game_3 . The reduction $\mathcal{B}$ is similar to Game_3 except that it takes (pp, pk_0) as input from the IND-CCA^- challenger instead of generating its own key material. It uses the provided oracles in the IND-CCA^- game to simulate Game_3 for $\mathcal{A}_3$. As $\mathcal{B}$ no longer knows the secret keys that have not been revealed through ORev oracle queries by $\mathcal{A}_3$, it will simulate ODecaps using OUpd and ODecaps^- provided by its challenger. The reduction must also keep its own experiment tree that corresponds to the one kept by the IND-CCA^- challenger, except that the reduction's tree also potentially contains half-nodes. Half-nodes will be marked by $\text{sk} = \perp$ and full nodes will be marked by $\text{sk} = \top$ (or a key if known). In the following, notice that there is a discrepancy between the indices of nodes in the IND-CCA -like Game_3 that $\mathcal{A}_3$ is playing, and those of the IND-CCA^- game that $\mathcal{B}$ is playing with its challenger. Let $\iota: \mathbb{N} \rightarrow \mathbb{N}$ be a translation function that $\mathcal{B}$ implicitly keeps, so that $\mathcal{A}_3$'s oracle calls that are passed through by $\mathcal{B}$ to its own challenger target the intended node. In particular, if $j \in \text{cop}_i$, then ι maps $j \mapsto i$.

Concretely, when $\mathcal{A}_3$ queries ODecaps with $(i, \text{ctxt}, \text{pk}, \text{mt})$ and case (i) happens, that is, there exists $\hat{i}$ with $(\text{pk}_{\text{par}_{\hat{i}}}, \text{pk}) = (\text{pk}_i, \text{pk})$, $\mathcal{B}$ proceeds as follows. If $\hat{i}$ is a half-node, then $\mathcal{B}$ creates a new half-node in its

$\mathcal{B}(\text{pp}, \text{pk}_0)$ $(\text{par}_0, \text{rev}_0, j, \text{typ}_0) := (\perp, 0, 0, \perp)$ $(i^*, \text{ctxt}^*) := (\perp, \perp)$ $b \leftarrow \mathcal{A}_3^{\text{OEncaps}, \text{ORev}, \text{OMChal}, \text{ODecaps}}(\text{pp}, \text{pk}_0)$ return b $\mathcal{A}_3.\text{OEncaps}(i)$ $(\text{kk}, \text{ctxt}, \text{pk}_j, \text{up}) \leftarrow \text{OEncaps}(\iota(i))$ create node $j := j + 1$ normally $\text{mt} \leftarrow \text{NIZKPoK.Sim}(\text{pp}, (\text{pk}_i, \text{pk}_j))$ return $(\text{kk}, \text{ctxt}, \text{pk}_j, \text{mt})$ $\mathcal{A}_3.\text{ORev}(i)$ $\text{sk}_i \leftarrow \text{ORev}(\iota(i));$ return sk_i $\mathcal{A}_3.\text{OMChal}(i)$ $(\text{kk}, \text{ctxt}^*, \text{pk}_j, \text{up}) \leftarrow \text{OMChal}(\iota(i))$ create node $j := j + 1$ normally $\text{mt} \leftarrow \text{NIZKPoK.Sim}(\text{pp}, (\text{pk}_i, \text{pk}_j))$ return $(\text{kk}, \text{ctxt}^*, \text{pk}_j, \text{mt})$ for $i \in \{\text{kk}, \text{ctxt}, \text{up}\} : \mathcal{A}_3.\text{H}_i(\text{msg}')$ if $\text{msg}' \notin \mathcal{Q}_i : \mathcal{Q}_i[\text{msg}'] \leftarrow \text{H}_i(\text{msg}')$ return $\mathcal{Q}_i[\text{msg}']$	$\mathcal{A}_3.\text{ODecaps}(i, \text{ctxt}, \text{pk}, \text{mt})$ assert $\text{pk}_i \neq \perp$; assert $\forall f(\text{pp}, \text{pk}_i, \text{pk}, \text{mt})$ if $i^* \neq \perp : \text{assert } \text{ctxt}^* \neq \text{ctxt} \text{ or } \text{pk}_{i^*} \neq \text{pk}_i$ if $\exists \hat{i} \text{ s.t. } \text{pk}_{\hat{i}} = \text{pk} : \quad // \text{ Case (i)}$ $j := j + 1; \hat{i} := \text{1st s.t. } \text{pk}_{\text{par}_{\hat{i}}} = \text{pk}_i \text{ and } \text{par}_{\hat{i}} \neq i$ if $\text{sk}_i = \perp :$ $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \perp, i, 0, \text{dec})$ $\text{cop}_{\hat{i}} := \text{cop}_{\hat{i}} \cup \{j\};$ return $\perp$ else : $\text{kk} \leftarrow \text{ODecaps}^-(\iota(i), \text{ctxt}, \text{pk})$ if $\text{kk} \neq \perp :$ $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \top, i, 0, \text{dec})$ return kk else : $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \perp, i, 0, \text{dec})$ $\text{cop}_{\hat{i}} := \text{cop}_{\hat{i}} \cup \{j\};$ return $\perp$ else : $// \text{ Case (ii)}$ if $\text{sk}_i = \perp :$ $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \perp, i, 0, \text{dec})$ $\text{kk} \leftarrow \text{ODecaps}^-(\iota(i), \text{ctxt}, \text{pk})$ if $\text{kk} \neq \perp :$ $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \top, i, 0, \text{dec})$ return kk else : $\text{dup} \leftarrow \text{NIZKPoK.Ext}(\text{pp}, (\text{pk}_i, \text{pk}), \text{mt})$ $\_ \leftarrow \text{OUpd}(\iota(i), \text{dup})$ $(\text{pk}_j, \text{sk}_j, \text{par}_j, \text{rev}_j, \text{typ}_j) := (\text{pk}, \top, i, 0, \text{dec})$ return $\perp$
---	---

Figure 29: Proof of Lemma 8: reduction from IND-CCA^- to Game_3 .

own experiment tree and adds it to $\text{cop}_{\hat{i}}$. Otherwise, $\mathcal{B}$ forwards $(\iota(i), \text{ctxt}, \text{pk})$ to its ODecaps^- oracle. If the query was successful and a shared key kk was decapsulated, $\mathcal{B}$ creates a new node marked as full like its IND-CCA^- challenger and returns kk back to $\mathcal{A}_3$. Otherwise $\mathcal{B}$ adds a new half-node to its own experiment tree, adds it to $\text{cop}_{\hat{i}}$, and returns $\perp$ to $\mathcal{A}_3$.

For all other $\text{ODecaps}(i, \text{ctxt}, \text{pk}, \text{mt})$ calls with valid member tags that correspond to case (ii) above, $\mathcal{B}$ proceeds as follows. First it queries ODecaps^- on $(\iota(i), \text{ctxt}, \text{pk})$, and if successful, simply creates a node marked as full like its IND-CCA^- challenger and returns the decapsulated key to $\mathcal{A}_3$. Otherwise, ODecaps^- did not create a new node in the experiment tree of the IND-CCA^- challenger, so $\mathcal{B}$ extracts the witness dup from mt , and calls $\text{OUpd}(i, \text{dup})$ to create the missing node. Even though this node is full in the challenger's experiment tree, $\mathcal{B}$ marks it as a half-node to conform to the behaviour of Game_3 since decapsulation failed.

When $\mathcal{A}_3$ outputs a bit b' , $\mathcal{B}$ forwards it to its IND-CCA^- challenger. If $\mathcal{A}_3$ makes no queries that violate the revealing condition on the challenge set of i^* in the IND-CCA -like Game_3 , neither does $\mathcal{B}$ violate the corresponding condition in the IND-CCA^- game, since the sets $\text{cop}_{\hat{i}}$ only contain half-nodes by definition. Thus

$$\text{Adv}_{\text{Game}_3}(\mathcal{A}_3) \leq \text{Adv}_{\text{IND-CCA}^-}^{\text{UKEM}_{\text{FO}}}(\mathcal{B}). \quad \square$$

6 No Dice! MLWE and (S)PCE

We study a version of our construction over module lattices, with the goal of replacing LWE with MLWE. The degree of the number field used in MLWE will be reflected in SPCE as the finite extension field degree.

Unfortunately, the Frobenius structure of finite extension fields makes SPCE there significantly easier than SPCE over prime fields. Intuitively, each SPCE sample over a finite extension field of degree d gives d samples via the action of the field automorphism group, using a process we call *Frobenius sample amplification*. Thus to make an SPCE instance hard enough, we are required to use a very small rate that is inverse-linearly proportional to d to thwart the linearisation attack from [BCD⁺24b]. Besides giving a restrictive bound on the degree wrt. the length of the code by the attack from [BMS26], this counteracts the efficiency gain of using MLWE in the first place.

Let $\mathcal{K}$ be the cyclotomic field of conductor $f = p^r$ for a rational prime p , and let $\mathcal{R}$ be its ring of integers.¹⁴ We first consider the inert case where the quotient $\mathcal{R}_q$ for a rational prime q is a finite field. We show that from each SPCE sample we obtain degree-many samples, making the instance vulnerable to a linearisation attack akin to [BCD⁺24b]. When $\mathcal{R}_q$ is not a field but rather a product of (isomorphic) fields, we show how the splitting behaviour of the ring $\mathcal{R}_q$ from Proposition 2 allows us to consider our sample amplification technique on each CRT slot separately, again leading to degree-many samples, cumulatively.

6.1 The Case of Inert Prime q

When $\mathcal{R}_q$ is a field, that is q is a primitive root modulo the conductor f , the (S)PCE problems on $\mathcal{R}_q$ are thus defined over $\mathcal{R}_q$ exactly as in Definition 1 by interpreting $\mathcal{R}_q$ as the prime-power field $\mathbb{F}_{q^d}$ of degree $d = \phi(f)$. Then the Galois group $\text{Gal}(\mathbb{F}_{q^d})$ is isomorphic to $\mathbb{Z}_d$, and generated by the q -Frobenius automorphism $x \mapsto x^q$. Let $\otimes$ be the Kronecker product of matrices.

Proposition 4. *Let $n, k, q \in \mathbb{Z}$ with $0 < k < n$ and q^d a prime power. Let $\mathbf{A}, \mathbf{B} \in \mathbb{F}_{q^d}^{k \times n}$ whose rows generate linear codes with $\mathbf{B} = \mathbf{U} \cdot \mathbf{A} \cdot \mathbf{O}$ for $\mathbf{U} \in \mathcal{GL}_k(\mathbb{F}_{q^d})$ and $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$, and let the rows of $\mathbf{B}^\perp \in \mathbb{F}_{q^d}^{(n-k) \times n}$ generate $\text{Span}(\mathbf{B})^\perp$. Then for all $\phi \in \text{Gal}(\mathbb{F}_{q^d})$ applied entry-wise we have*

$$(\phi(\mathbf{A}) \otimes \phi(\mathbf{B}^\perp)) \cdot \text{vec}(\mathbf{O}) = \mathbf{0}^{k \cdot (n-k)}.$$

Proof. Since ϕ is a field automorphism, apply it to the linearised system in [BCD⁺24b, Prop. 2], and use $\phi(\text{vec}(\mathbf{O})) = \text{vec}(\mathbf{O})$, since $\{0, \pm 1\} \subseteq \mathbb{F}_q$. $\square$

Given an instance $(\mathbf{A}, \mathbf{B})$ of (S)PCE over $\mathbb{F}_{q^d}$, we apply all d automorphisms to the instance and stack the linearised systems from Proposition 4 to get a combined system matrix with $d \cdot k \cdot (n - k)$ rows and $\text{vec}(\mathbf{O})$ in its right kernel.

Definition 24 (Frobenius sample amplification). *Let $n, k, q \in \mathbb{Z}$ with $0 < k < n$ and q^d a prime power, and let $\phi_i \in \text{Gal}(\mathbb{F}_{q^d})$ for $i \in \mathbb{Z}_d$. Given an (S)PCE row-instance $(\mathbf{A}, \mathbf{B} = \mathbf{U} \cdot \mathbf{A} \cdot \mathbf{O})$ of $[n, k]$ -linear codes over $\mathbb{F}_{q^d}$ with $\mathbf{U} \in \mathcal{GL}_k(\mathbb{F}_{q^d})$ and $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$, compute any row generator $\mathbf{B}^\perp$ of $\text{Span}(\mathbf{B})^\perp$ and construct the following matrix in $\mathbb{F}_{q^d}^{d \cdot k \cdot (n-k) \times n^2}$ with $\text{vec}(\mathbf{O})$ in the right kernel*

$$\left[\mathbf{A} \otimes \mathbf{B}^\perp \parallel \phi_1(\mathbf{A}) \otimes \phi_1(\mathbf{B}^\perp) \parallel \cdots \parallel \phi_{d-1}(\mathbf{A}) \otimes \phi_{d-1}(\mathbf{B}^\perp) \right].$$

We then hope the rank of this matrix is close enough to $n^2 - 1$ to extract the secret (signed) permutation. The aforementioned ‘hope’ is formalised in the following assumption and later tested experimentally in Section 6.4.

Assumption 2 *Let $n, k, q \in \mathbb{Z}$ with $0 < k < n$ and q^d a prime power. Then for a random $[n, k]$ -linear code $\mathfrak{C}$ over $\mathbb{F}_{q^d}$ with $k \leq \frac{n}{2}$ sampled either uniformly or with any hull dimension $h \leq k$, we have with non-negligible probability that the rank of the system from Definition 24 is $\min\{d \cdot k \cdot (n - k), n^2 - 1\}$.*

Assumption 2 with respect to every hull dimension is based on the discussion in [BCD⁺24b, Sec. 4.3], where the authors argue the hull dimension does not play a role in the linearisation attack, including for self-dual codes, which they verify experimentally. We thus have the following.

¹⁴ We leave non-prime-power conductors for future work.

Lemma 9. Let $n, k, q \in \mathbb{Z}$ with $0 < k < n$ and q^d a prime power. Under Assumption 2, if $d \geq \left\lfloor \frac{n^2}{k \cdot (n-k)} \right\rfloor + 1$, there exists an algorithm running in time $\mathcal{O}(n^{2\omega})$ and with non-negligible advantage against (search or decision) (S)PCE $_{n,k,q^d}(\mathbf{D})$ for $\mathbf{D} \in \left\{ \mathbf{U}_{n,k,q^d}, \mathbf{D}_{n,k,q^d}^{\chi,h}, \mathbf{D}_{n,k,q^d}^k \mid 0 \leq h < k, \chi \in \{\pm 1\} \right\}$, where $\mathbf{U}_{n,k,q^d}$ is the distribution of uniformly random full-rank matrices in $\mathbb{F}_{q^d}^{k \times n}$.

Proof. By Assumption 2 we simply construct the system in Definition 24 and compute its right kernel using linear algebra, costing $\mathcal{O}(n^{2\omega})$. The success probability is the non-negligible success probability in Assumption 2. $\square$

Our attack can likely make use of many of the same optimisations discussed in [BCD⁺24b]. We do not pursue this further, as we only use this attack to discourage us from using structured lattices.

6.2 The Case of Split Prime q

We now consider primes q that are not inert in $\mathcal{R}$, that is, when (q) splits properly into a product of prime ideals and thus $\mathcal{R}_q$ splits properly into a product of fields. We may be concerned that we have wandered into the unfamiliar territory of linear codes over rings; however, the following statement assures us that our equivalence problem is indeed just a collection of $\frac{d}{m}$ instances of (S)PCE over the subfield $\mathbb{F}_{q^m}$ of $\mathbb{F}_{q^d}$, where m is the order of q modulo f , courtesy of the CRT.

Lemma 10. Assume the notation of Proposition 2. Consider the translation of Definition 1 to free modules over the ring $\mathcal{R}_q$. Each (S)PCE instance $(\mathbf{A}, \mathbf{B} = \mathbf{S}\mathbf{F}(\mathbf{A} \cdot \mathbf{O}))$ of free modules over $\mathcal{R}_q$ yields $\frac{d}{m}$ independent (S)PCE instances $(\mathbf{A}_i, \mathbf{B}_i = \mathbf{S}\mathbf{F}(\mathbf{A}_i \cdot \mathbf{O}))$ over the field $\mathbb{F}_{q^m}$ sharing the same isometry $\mathbf{O}$.

Proof. Denote by Π_i for $i = 1, \dots, \frac{d}{m}$ the CRT projection map $\Pi_i: \mathcal{R}_q \rightarrow \mathbb{F}_{q^m}$ to the i -th component field. Let $(\mathbf{A}, \mathbf{B})$ be any (S)PCE instance over $\mathcal{R}_q$ for a secret $\mathbf{O}$ and define $(\mathbf{A}_i = \Pi_i(\mathbf{A}), \mathbf{B}_i = \mathbf{S}\mathbf{F}(\Pi_i(\mathbf{B})))$ over the finite field $\mathbb{F}_{q^m}$. Then $\Pi_i(\mathbf{O}) = \mathbf{O}$ for all $i = 1, \dots, \frac{d}{m}$, since Π_i is a ring homomorphism. $\square$

By the inert case, we get that each of the samples over $\mathbb{F}_{q^m}$ yields m samples sharing the same isometry. Therefore, regardless of the splitting behaviour of q in $\mathcal{R}$, one (S)PCE sample over $\mathcal{R}_q$ is expected to yield degree-many samples of (S)PCE over some finite extension field of $\mathbb{F}_q$.

Theorem 4. Let $n, k \in \mathbb{Z}$ with $0 < k < n$. Assume the notation of Proposition 2 and Lemma 10, except that q is a rational prime of order m modulo f , the conductor of $\mathcal{R}$, that splits into $\frac{d}{m}$ prime ideals in $\mathcal{R}$, which has degree $d = \phi(f)$. Under Assumption 2, if $d \geq \left\lfloor \frac{n^2}{k \cdot (n-k)} \right\rfloor + 1$, there exists an algorithm running in time $\mathcal{O}(n^{2\omega})$ and with non-negligible advantage against (search or decision) (S)PCE defined for $[n, k]$ -linear codes over $\mathcal{R}_q$ with distribution $\mathbf{D} \in \left\{ (\mathbf{U}_{n,k,q^m})^{\frac{d}{m}}, (\mathbf{D}_{n,k,q^m}^{\chi,h})^{\frac{d}{m}}, (\mathbf{D}_{n,k,q^m}^k)^{\frac{d}{m}} \mid 0 \leq h < k, \chi \in \{\pm 1\} \right\}$, that is exactly like in Lemma 9 but applied to each of the $\frac{d}{m}$ slots of $\mathcal{R}_q$ individually.

Proof. The algorithm simply computes the CRT projections $(\Pi_j(\mathbf{A}), \Pi_j(\mathbf{B}))$, and then uses Frobenius sample amplification from Definition 24 on each projection to get $(\mathbf{A}_{(i,j)}, \mathbf{B}_{(i,j)}) = (\phi_i(\Pi_j(\mathbf{A})), \phi_i(\Pi_j(\mathbf{B})))$ for each $i \in \mathbb{Z}/m$ and $j \in \mathbb{Z}/\frac{d}{m}$, constructing the linearised system $[\dots \|\phi_i(\Pi_j(\mathbf{A})) \otimes \phi_i(\Pi_j(\mathbf{B}))^\perp\| \dots]_{i,j}$, where $\Pi_j(\mathbf{B})^\perp$ is any generator of $\mathfrak{C}(\Pi_j(\mathbf{B}))^\perp$. Since the distribution $\mathbf{D}$ samples for each slot of $\mathcal{R}_q$ independently, by Assumption 2 the rank of this matrix is $\min\{d \cdot k \cdot (n-k), n^2 - 1\}$, and we recover $\mathbf{O}$ from its kernel using linear algebra, costing $\mathcal{O}(n^{2\omega})$. The success probability is that in Assumption 2. $\square$

6.3 A Bound on the Extension Degree

The authors of [BCD⁺24b] give an algorithm that solves (S)PCE with at least $\left\lfloor \frac{1}{r \cdot (1-r)} \right\rfloor + 1$ samples sharing the same isometry, where r is the code rate, subject to mild linear independence assumptions they experimentally verify. Combined with Theorem 4, this yields the following upper bound on the extension field degree.

Corollary 4. *Assume the notation of Theorem 4. Under Assumption 2, a necessary condition for the hardness of (S)PCE of rate r over $\mathbb{F}_{q^d}$ is $d < \left\lfloor \frac{1}{r \cdot (1-r)} \right\rfloor + 1$.*

Moreover, in [BCD⁺24b, Sec. 4.3] the authors use a guessing technique to reduce the number of samples needed to solve PCE of rate $\frac{1}{2}$ to 2, suggesting not even degree $d = 2$ is secure despite the generic bound for $r = \frac{1}{2}$ requiring $d \geq 5$. Their guessing technique can be counteracted by using longer instances with smaller rates by a factor linear in d . On top of the concrete attack in Theorem 4, the mere presence of other ‘inner products’ induced by non-trivial field automorphisms noted in [BOST19, Prop. 8] makes us wary of using extension fields. Investigating hollow codes wrt. other geometries marks another target for future work, since experimentally our attack also works when the instance pair has all degree-many hulls with large dimension (contrary to [BOST19]). We conclude our UPKE scheme over $\mathcal{R}_q$ results in no improvement, since the factor of d size decrease due to modules is negated by roughly a factor of d size increase due to longer codes.

6.4 Experiments

We implemented the attack given in Theorem 4, and tested Assumption 2 experimentally. Our code is in Appendix C.1, and also **attached** to this document. The code for testing Assumption 2 depends on [ABL25b]. Some experimental data for Assumption 2 are in Appendix B.1. We observe that the rank condition in Assumption 2 is true with high probability for all the tested parameters (and growing with q), otherwise we observe co-rank 1 relative to Assumption 2. Even in the cases when Assumption 2 fails, we observe that the rank deficiency in the linearisation attack from [BCD⁺24b] if $d \cdot k \cdot (n - k) \geq n^2$ is so small that the (signed) permutation can simply be guessed. We ran the same experiment on linear codes whose hull dimension is non-trivial (large) for every *Hermitian inner product* [BOST19]. This confirms our attack also works in this setting, where the attack in [BOST19] fails.

7 A Scheme with Multiple Keys

In this section, we study the multi-key version of the UPKE scheme from Fig. 3. Concretely, instead of only one secret vector $\mathbf{r} \in \{\pm 1\}^n$ we now have ℓ of them assembled into a matrix $\mathbf{R} \in \{\pm 1\}^{n \times \ell}$, and the public key becomes a matrix $\mathbf{A}$ generating a $[n, k]$ -linear h -hollow code along with a matrix of syndromes $\mathbf{S}^T = \mathbf{R}^T \cdot \mathbf{A}$. This slight increase in the public key size is motivated by a large decrease in ciphertext size when encrypting many bits, due to the reuse of the first and largest component of the ciphertext.

To reason about the security of this new UPKE scheme and the resulting UKEM, we introduce a new PCE-like assumption. After introducing our assumption and the multi-key version of the UPKE scheme, we show that based on this assumption our multi-key scheme admits update freshness, and thus this UPKE scheme is Fork-IND-CPA-secure, allowing us to compile it to an IND-CCA-secure UKEM as in Fig. 16 with a generalised member tag construction based on Section 4. We conclude by discussing cryptanalysis of the assumption.

7.1 Hinted Signed Permutation Equivalence

Consider the following problem, where we are given an SPCE instance (search or distinguish), a random signed matrix, and an image of this matrix with the same secret isometry (masked with another signed permutation).

Definition 25 (Hinted SPCE). *Let $n, k, \ell, q \in \mathbb{Z}$ parametrised by λ with $k < n$ and q prime, and let $\mathbf{D}$ be a distribution on the subset of full-rank matrices in $\mathbb{F}_q^{n \times k}$. The advantage of any PPT adversary $\mathcal{A}$ against*

$\text{KGen}(1^\lambda)$ $\mathbf{A} \leftarrow \$ D_{n,k,q}^{\chi,h}$ // fix from [Ben26] $\mathbf{R} \leftarrow \$ \{\pm 1\}^{n \times \ell}$; $\mathbf{S}^T := \mathbf{R}^T \cdot \mathbf{A} \bmod q$ $\text{pk} := (\mathbf{A}, \mathbf{S})$; $\text{sk} := \mathbf{R}$; return (pk, sk)	$\text{Enc}(\text{pk}, \text{msg} \in \mathbb{Z}_p^\ell)$ $\mathbf{x} \leftarrow \$ \mathbb{F}_q^k$; $\mathbf{e} \leftarrow \$ \eta^n$; $\mathbf{f} \leftarrow \$ \eta^\ell$ $\mathbf{c}_0 := \mathbf{A} \cdot \mathbf{x} + \mathbf{e} \bmod q$ $\mathbf{c}_1 := \mathbf{S}^T \cdot \mathbf{x} + \mathbf{f} + \left\lfloor \frac{q}{p} \right\rfloor \cdot \text{msg} \bmod q$ return ctxt := $(\mathbf{c}_0, \mathbf{c}_1)$
$\text{Dec}(\text{sk}, \text{ctxt})$ return $\left\lfloor \frac{p}{q} \cdot (\mathbf{c}_1 - \mathbf{R}^T \cdot \mathbf{c}_0 \bmod q) \right\rfloor \bmod p$	$\text{UpdPk}(\text{pk})$ $\rho \leftarrow \$ \{0, 1\}^\lambda$; $(\mathbf{O}, \mathbf{P}) \leftarrow \mathbf{H}(\rho)$ $[\mathbf{A}' \parallel \mathbf{S}']$, $\mathbf{U} := \text{SF}(\text{diag}(\mathbf{O}, \mathbf{P}) \cdot [\mathbf{A} \parallel \mathbf{S}])$ $\text{pk}' := (\mathbf{A}', \mathbf{S}')$; $\text{up} \leftarrow \text{Enc}(\text{pk}, \rho)$ return (pk', up)
$\text{UpdSk}(\text{sk}, \text{up})$ $\rho \leftarrow \text{Dec}(\text{sk}, \text{up})$; $(\mathbf{O}, \mathbf{P}) \leftarrow \mathbf{H}(\rho)$ $\mathbf{R}' := \mathbf{O} \cdot \mathbf{R} \cdot \mathbf{P}^T$; return sk' := $\mathbf{R}'$	

Figure 30: Heuristic (multi-key) updatable public key encryption (cf. Fig. 3).

Hinted $\Delta\text{SPCE}_{n,k,\ell,q}(\mathbf{D})$ is defined as

$$\text{Adv}_{\text{Hint}\Delta\text{SPCE}}(\mathcal{A}) = \left| \Pr \left[b = b' \mid \begin{array}{l} \mathbf{G} \leftarrow \$ \mathbf{D}; \mathbf{R} \leftarrow \$ \{\pm 1\}^{n \times \ell}; \mathbf{S}^T := \mathbf{R}^T \cdot \mathbf{G} \\ (\mathbf{O}, \mathbf{P}) \leftarrow \$ \mathcal{O}_n(\mathbb{Z}) \times \mathcal{O}_\ell(\mathbb{Z}); \mathbf{U} \leftarrow \$ \mathcal{GL}_k(\mathbb{F}_q) \\ \mathbf{R}' := \mathbf{O} \cdot \mathbf{R} \cdot \mathbf{P}^T; \mathbf{H}_0 \leftarrow \$ \mathbf{D}; \mathbf{S}_0^T := \mathbf{R}'^T \cdot \mathbf{H}_0 \\ \mathbf{H}_1 := \mathbf{O} \cdot \mathbf{G} \cdot \mathbf{U}; \mathbf{S}_1^T := \mathbf{R}'^T \cdot \mathbf{H}_1 \\ b \leftarrow \$ \{0, 1\}; b' \leftarrow \mathcal{A}(\mathbf{G}, \mathbf{S}, \mathbf{H}_b, \mathbf{S}_b, \mathbf{R}') \end{array} \right] - \frac{1}{2} \right|.$$

Assumption 3 (Hinted SPCE) *Let $D_{n,k,q}^{\chi,h}$ be the distribution of uniformly random $[n, k]$ -linear h -hollow codes over $\mathbb{F}_q$ with square class χ like in Definition 1, and $\ell = \mathcal{O}(\log(n))$. Then*

$$\forall \text{PPT } \mathcal{A}: \text{Adv}_{\text{Hint}\Delta\text{SPCE}_{n,k,\ell,q}^{\chi,h}}(\mathcal{A}) \leq \text{Adv}_{\Delta\text{PCE}_{n,k,q}^{\chi,h-\ell}}(\mathcal{A}).$$

This assumption may seem contrived at first, but consider [ABL25a, Lem. 11]. We show in Proposition 5 that [ABL25a, Lem. 11] proves that Hinted ΔSPCE is hard in the single-key case of $\ell = 1$, assuming ΔPCE . Having $\ell \geq 1$ then allows for ℓ secret keys, as shown in Fig. 30. We reduce the hull dimension by ℓ in Assumption 3 to capture that the adversary may consider the stacked instance $([\mathbf{G} \parallel \mathbf{S}^T], [\mathbf{H}_b \parallel \mathbf{S}_b^T])$, where the hull dimension may drop by ℓ , though the instance is not random.

7.2 The Multi-Key Scheme

We give our multi-key version of the UPKE scheme from Fig. 3 relying on Assumption 3, and, analogously to Section 5, compile it to a UKEM.

Definition 26 (Heuristic Multi-Key UPKE). *Let $n, k, \ell, h, q, p, \chi \in \mathbb{Z}$ with $n = 2k$, $2h \leq k$, $\ell \leq \mathcal{O}(\log(n))$ and $\ell \leq h$, $\chi \in \{\pm 1\}$ and q prime with p like in Proposition 3. Let $\mathbf{H}: \{0, 1\}^\lambda \rightarrow \mathcal{O}_n(\mathbb{Z}) \times \mathcal{O}_\ell(\mathbb{Z})$ be a random oracle. The multi-key UPKE scheme is given in Fig. 30.*

We give multi-key versions of Lemmas 4 and 5 in Section 7.3, showing our multi-key variant also satisfies Fork-IND-CPA security and update freshness (under Assumption 3). The proofs are identical, except that we use Hinted ΔSPCE instead of ΔPCE . Applying Theorem 3 we get the following.

MTag(pp, pk, pk', dup)	Setup(1^λ)
parse $((\mathbf{A}, \mathbf{S}), (\mathbf{B}, \mathbf{T})) \leftarrow (\text{pk}, \text{pk}')$	return pp $\leftarrow \Pi.\text{Setup}(1^\lambda)$
parse $(\mathbf{O}, \mathbf{P}) \leftarrow \text{dup}; (\mathbf{B}', \mathbf{U}) := \text{SF}(\mathbf{O} \cdot \mathbf{A})$	
assert $\mathbf{B} = \mathbf{B}'$ and $\mathbf{T}^\top = \mathbf{P} \cdot \mathbf{S}^\top \cdot \mathbf{U}$	Vf (pp, pk, pk', mt)
$\pi \leftarrow \Pi.\text{Prove}(\text{pp}, (\mathbf{A}, \mathbf{S}^\top, \mathbf{B}, \mathbf{T}^\top), (\mathbf{O}, \mathbf{P}))$	parse $((\mathbf{A}, \mathbf{S}), (\mathbf{B}, \mathbf{T}), \pi) \leftarrow (\text{pk}, \text{pk}', \text{mt})$
return mt $:= \pi$	return $\Pi.\text{Vf}(\text{pp}, (\mathbf{A}, \mathbf{S}^\top, \mathbf{B}, \mathbf{T}^\top), \pi)$

Figure 31: Our explicit member tags MT for the scheme in Fig. 30 using the proof of isometry NIZKPoK scheme $\Pi = (\text{Setup}, \text{Prove}, \text{Vf}, \text{Sim}, \text{Ext})$ that is the Σ -protocol in Fig. 8 compiled with Fischlin's transform in Fig. 7 (or Fig. 10).

7.3 Multi-key Scheme Security

We give the equivalent statements for our multi-key UPKE scheme in Fig. 30.

Lemma 11 (Multi-key weak update freshness). *For any PPT algorithm $\mathcal{A}$ against Game_{wUF} for the UPKE scheme in Definition 26 there is a PPT algorithm $\mathcal{B}$ such that*

$$\text{Adv}_{\text{wUF}}(\mathcal{A}) \leq \text{Adv}_{\text{Hint}\Delta\text{SPCE}}(\mathcal{B}),$$

where wUF stands for weak update freshness and is identical to Definition 23 except that $\mathcal{A}$ is not given sk_0 .

Proof (Sketch). The proof is the same as in [ABL25a, Lem. 11], except that we use $\text{pk} = (\mathbf{G}, \mathbf{S})$, $\text{pk}_b = (\mathbf{H}_b, \mathbf{S}_b)$, and $\text{sk}_b = \mathbf{R}'$ from the Hinted SPCE instance instead of the re-randomised all-1's secret keys.

Lemma 12 (Multi-key Fork-IND-CPA security). *For the parameters under which Hinted ΔSPCE , LWE^h and DSIS^h are assumed hard, the UPKE scheme from Fig. 30 is Fork-IND-CPA secure. Concretely, for any PPT algorithm $\mathcal{A}$ against UPKE that makes q_{Chal} calls to OChal , there are PPT algorithms $\mathcal{B}_1, \mathcal{B}_2$ such that*

$$\text{Adv}_{\text{Fork-IND-CPA}}^{\text{UPKE}}(\mathcal{A}) \leq \text{Adv}_{\text{wUF}}^{\text{UPKE}}(\mathcal{B}_1) + \text{Adv}_{\text{IND-CPA}}^{\text{UPKE}}(\mathcal{B}_2) + \frac{q_{\text{RO}}}{2^\lambda},$$

where $\mathcal{B}_1$ makes q_{Chal} queries to OChal and $\mathcal{B}_2$ makes q_{Chal} queries to OEnc .

Proof (Sketch). Exactly the same as the proof of Lemma 5, except that we use Hinted ΔSPCE instead of ΔPCE in the game hop from Game_3 to Game_4 with Lemma 11.

We also compile our multi-key scheme under the assumption that the update freshness game hop in Lemma 12 can be proven from weak update freshness or from Hinted ΔSPCE directly.

Corollary 5 (of Theorem 3 (heuristic)). *For the parameters under which Hint- ΔSPCE , LWE^h and DSIS^h are assumed hard, the UKEM in Fig. 16, constructed from the UPKE scheme in Fig. 30 and MT in Fig. 31, is correct and IND-CCA-secure in the ROM.*

7.4 Initial Cryptanalysis of Hinted SPCE

Let us make some cryptanalytic remarks. First, the hardness of Hinted SPCE with $\ell = 1$ is implied by the hardness of PCE. This follows directly from the proof technique of [ABL25a, Lem. 11], and we recall it here.

Proposition 5 ($\Delta\text{PCE}_{n,k,q}^{\chi,h} \rightarrow \text{Hint}\Delta\text{SPCE}_{n,k,1,q}^{\chi,h}$). *Let $n, k, h, q, \chi \in \mathbb{Z}$ with $0 \leq h \leq k \leq n$, q prime and $\chi \in \{\pm 1\}$. For any PPT algorithm $\mathcal{A}$ against $\text{Hint}\Delta\text{SPCE}_{n,k,1,q}^{\chi,h}$ there is a PPT algorithm $\mathcal{B}$ against $\Delta\text{PCE}_{n,k,q}^{\chi,h}$ such that*

$$\text{Adv}_{\text{Hint}\Delta\text{SPCE}_{n,k,1,q}^{\chi,h}}(\mathcal{A}) \leq \text{Adv}_{\Delta\text{PCE}_{n,k,q}^{\chi,h}}(\mathcal{B}).$$

Proof. On input a Δ PCE instance $(\mathbf{A}, \mathbf{B})$, the reduction $\mathcal{B}$ samples $\mathbf{O}_\mathbf{A}, \mathbf{O}_\mathbf{B} \leftarrow \mathcal{O}_n(\mathbb{Z})$, computes $\tilde{\mathbf{A}} = \text{SF}(\mathbf{O}_\mathbf{A} \cdot \mathbf{A})$, $\tilde{\mathbf{B}} = \text{SF}(\mathbf{O}_\mathbf{B} \cdot \mathbf{B})$, $\mathbf{r}_\mathbf{A} = \mathbf{O}_\mathbf{A} \cdot [1]^n$, $\mathbf{r}_\mathbf{B} = \mathbf{O}_\mathbf{B} \cdot [1]^n$, and sends $(\tilde{\mathbf{A}}, \mathbf{r}_\mathbf{A}^\top \cdot \tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \mathbf{r}_\mathbf{B}^\top \cdot \tilde{\mathbf{B}}, \mathbf{r}_\mathbf{B})$ to $\mathcal{A}$. It then outputs $\mathcal{A}$'s bit. If $\mathbf{B} = \text{SF}(\mathbf{P} \cdot \mathbf{A})$ is a real sample, then $\tilde{\mathbf{B}} = \text{SF}(\mathbf{O}_\mathbf{B} \cdot \mathbf{P} \cdot \mathbf{O}_\mathbf{A}^{-1} \cdot \tilde{\mathbf{A}})$ is a properly distributed Δ SPCE sample, and $\mathbf{r}_\mathbf{B} = \mathbf{O}_\mathbf{B} \cdot \mathbf{P} \cdot \mathbf{O}_\mathbf{A}^{-1} \cdot \mathbf{r}_\mathbf{A}$, since $\mathbf{P} \cdot [1]^n = [1]^n$ for any permutation $\mathbf{P}$. Otherwise the samples are unrelated. $\square$

We were not able to provide a further reduction for $\ell > 1$, and leave it for future work. Further cryptanalytic effort could go in many directions, such as:

Structured assumption. Let $\mathbf{r} = \text{vec}(\mathbf{R})$ and $\mathbf{r}' = \text{vec}(\mathbf{R}')$ be vectors with $n \cdot \ell$ entries obtained by stacking the columns of the matrices. If $\mathbf{R} \in \{\pm 1\}^{n \times \ell}$ is uniformly random, then also $\mathbf{r} \in \{\pm 1\}^{n \cdot \ell}$ is uniformly random. The vectors $\mathbf{r}$ and $\mathbf{r}'$ are connected by a *structured signed permutation* $\mathbf{O} \otimes \mathbf{P}$. Thus the Hinted-SPCE assumption for $\ell > 1$ can be understood as a structured version of the Hinted-SPCE assumption for $\ell = 1$, to which PCE reduces.

Syndromes as codes. Recall the syndrome matrices $\mathbf{S} = \mathbf{G}^\top \cdot \mathbf{R}$ and $\mathbf{S}_b = \mathbf{H}_b^\top \cdot \mathbf{R}'$ given as part of the Hinted Δ SPCE instance. If this is a real instance ($b = 1$) with $\mathbf{H}_1, \mathbf{U} = \text{SF}(\mathbf{O} \cdot \mathbf{G})$, then $\mathbf{S}_1 = \mathbf{U}^\top \cdot \mathbf{S} \cdot \mathbf{P}^\top$. In the random case $\mathbf{S}_0$ is unrelated to $\mathbf{S}$. If $\ell > k$ (which is outside the scope of the assumption), then this is a Δ SPCE instance for random codes. Indeed, since $\mathbf{S}_b$ and $\mathbf{S}$ behave like random matrices by Theorem 1, we may consider this as a random Δ SPCE instance $(\mathbf{S}^\top, \mathbf{S}_b^\top)$ connected by the secret signed permutation $\mathbf{P}$ and masked by the basis change $\mathbf{U}$. Analogously for the search version.

8 Parameters

We modified the parameter selection Python script [ABL25b] from [ABL25a] (depending on [APS15]) to handle our parameter regime, IND-CCA security, and the multi-key setting. Our script for estimating parameters is in Appendix C.2, and also [attached](#) to this document. Like [ABL25a], we make sure that PCE and LWE are hard simultaneously, and also increase the hull dimension to resist the new attack in [BMS26]. Concretely, we require $\binom{n}{h - \sqrt{2n}} \geq 2^\lambda$ per the countermeasure in [BMS26, p. 4691]. We set $n = 2k$ for DSIS parameters per Theorem 1, and add ℓ to h per Assumption 3. The optimal p is always 2 because of public key growth, and we always assume a 128-bit message for a fair comparison in the UKEM setting. To estimate member tag size, our script finds the optimal challenge length and weight that minimise proof size with fixed-weight challenges from Section 4.3. We picked the number of rounds $r = 32$ as the smallest $r \mid \lambda$ such that the two terms in the estimated number of operations are below roughly 2^{30} (recall that even for the FS transform, the prover needs to do $\lambda \cdot \mathcal{O}(n^\omega)$ operations which is about 30 bits for a 10-bit n). Our script also picks the optimal number of secret keys for the scheme in Section 7, which is $\ell = 16$ for $\lambda = 128$. We present our parameters and corresponding public key, ciphertext, and member tag sizes in Table 1. We also include some more aggressively set parameters by setting $\sigma = 0.5$ instead of the standard 3.2 (we include the same estimates for the scheme in [ABL25a]). Table 1 concludes with the numbers reported in [AFMR26].

Remark 5. While we believe our transformation in Theorem 3 for unbounded UPKE can be used on e.g. the UPKE scheme in [BFK⁺26] to produce other UKEMs admitting member security, we are reluctant to provide concrete parameters in Table 1 until parameters are updated by the authors due to the recent attacks in [Wes26, Udo26], and until an obvious ‘best’ candidate for member tags emerges. The original reported sizes in [BFK⁺26] are 408 bytes for $|\mathbf{pk}|$, and 2183 bytes for $|\mathbf{ctxt}|$ and $|\mathbf{up}|$.

Acknowledgements

We thank Paolo Santini for pointing us to [BOST19, Prop. 8] and an anonymous reviewer at AC’26 for suggesting Theorem 1.

References

- ABL25a. Martin R. Albrecht, Benjamin Benčina, 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, Cham, May 2025. https://doi.org/10.1007/978-3-031-91101-9_13. 2, 3, 4, 5, 6, 7, 13, 14, 15, 20, 23, 37, 38, 39, 43, 44, 45
- ABL25b. Martin R. Albrecht, Benjamin Benčina, and Russell W. F. Lai. Hollow LWE: A new spin - unbounded updatable encryption from LWE and PCE, May 2025. https://doi.org/10.1007/978-3-031-91101-9_13. Artifact available at <https://artifacts.iacr.org/eurocrypt/2025/a9>. 36, 39, 48
- 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. https://doi.org/10.1007/978-3-030-56784-2_9. 1
- 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. https://doi.org/10.1007/978-3-031-58754-2_13. 2, 3, 7, 15
- AFMR25. Joël Alwen, Georg Fuchsbauer, Marta Mularczyk, and Doreen Riepel. Lattice-based updatable public-key encryption for group messaging. Cryptology ePrint Archive, Report 2025/365, 2025. <https://eprint.iacr.org/2025/365>. 21, 22, 30
- AFMR26. Joël Alwen, Georg Fuchsbauer, Marta Mularczyk, and Doreen Riepel. Lattice-Based Updatable KEM for Group Messaging. In Nadia Heninger and Mike Rosulek, editors, *CRYPTO 2026*, volume 16809 of *LNCS*, pages 168–199. Springer, August 2026. https://doi.org/10.1007/978-3-032-35428-0_6. 2, 3, 4, 7, 8, 19, 20, 39
- AL21. Martin R. Albrecht and Russell W. F. Lai. Subtractive sets over cyclotomic rings - limits of Schnorr-like arguments over lattices. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 519–548, Virtual Event, August 2021. Springer, Cham. https://doi.org/10.1007/978-3-030-84245-1_18. 16
- 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, October 2015. <https://doi.org/10.1515/jmc-2015-0016>. Commit 352ddaf of <https://github.com/malb/lattice-estimator/>. 39
- BBPS21. Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. LESS-FM: Fine-tuning signatures from the code equivalence problem. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *PQCrypto 2021*, pages 23–43. Springer, Cham, 2021. https://doi.org/10.1007/978-3-030-81293-5_2. 17
- BCD⁺24a. Markus Bläser, Zhili Chen, Dung Hoang Duong, Antoine Joux, Tuong Ngoc Nguyen, Thomas Plantard, Youming Qiao, Willy Susilo, and Gang Tang. On digital signatures based on group actions: QROM security and ring signatures. In Markku-Juhani Saariinen and Daniel Smith-Tone, editors, *PQCrypto 2024, Part I*, pages 227–261. Springer, Cham, June 2024. https://doi.org/10.1007/978-3-031-62743-9_8. 18
- BCD⁺24b. Alessandro Budroni, Jesús-Javier Chi-Domínguez, Giuseppe D’Alconzo, Antonio J. Di Scala, and Mukul Kulkarni. Don’t use it twice! Solving relaxed linear equivalence problems. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VIII*, volume 15491 of *LNCS*, pages 35–65. Springer, Singapore, December 2024. https://doi.org/10.1007/978-981-96-0944-4_2. 3, 34, 35, 36
- Ben26. Benjamin Benčina. On Local Invariants for Permutation Equivalence. Cryptology ePrint Archive, Paper 2026/995, 2026. <https://eprint.iacr.org/2026/995>. 4, 5, 6, 7, 16, 20, 37, 43
- BFK⁺26. Andrea Basso, Tako Boris Fouotsa, Fatna Kouider, Péter Kutas, Luciano Maino, and Laurane Marco. Updatable public-key encryption from FESTA. Cryptology ePrint Archive, Paper 2026/1014, 2026. <https://eprint.iacr.org/2026/1014>. to appear at ASIACRYPT 2027. 3, 19, 39
- BGPS23. Huck Bennett, Atul Ganju, Pura Peetathawatchai, and Noah Stephens-Davidowitz. Just how hard are rotations of $\mathbb{Z}^n$? algorithms and cryptography with the simplest lattice. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 252–281. Springer, Cham, April 2023. https://doi.org/10.1007/978-3-031-30589-4_9. 3
- BKV19. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 227–247. Springer, Cham, December 2019. https://doi.org/10.1007/978-3-030-34578-5_9. 17

- BMS26. Michele Battagliola, Rocco Mora, and Paolo Santini. Using the Schur Product to Solve the Code Equivalence Problem. *IEEE Transactions on Information Theory*, 72(7):4675–4694, 2026. <https://doi.org/10.1109/TIT.2026.3694630>. 2, 34, 39
- Bor23. Giacomo Borin. *Cryptographic Group Actions and Digital Signatures, with a focus on Code Equivalence Problems*. PhD thesis, Università di Trento, 2023. 13
- BOST19. Magali Bardet, Ayoub Otmani, and Mohamed Saeed-Taha. Permutation Code Equivalence is Not Harder Than Graph Isomorphism When Hulls Are Trivial. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2464–2468. IEEE, 2019. <https://doi.org/10.1109/ISIT.2019.8849855>. 36, 39, 47
- BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Berlin, Heidelberg, May / June 2006. https://doi.org/10.1007/11761679_25. 29
- 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. https://doi.org/10.1007/3-540-39200-9_16. 1
- CL24. Yi-Hsiu Chen and Yehuda Lindell. Optimizing and implementing fischlin’s transform for UC-secure zero knowledge. *CiC*, 1(2):11, 2024. <https://doi.org/10.62056/a66chey6b>. 10
- Dam10. Ivan Damgård. On Σ -protocols, 2010. <https://www.cs.au.dk/~ivan/Sigma.pdf>. Lecture Notes for CPT 2010 v2, at Aarhus University. 10
- 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. https://doi.org/10.1007/978-3-030-90456-2_9. 1, 6
- DKW22. Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Updatable public key encryption in the standard model. Cryptology ePrint Archive, Report 2022/068, 2022. <https://eprint.iacr.org/2022/068>. 15, 19
- DvW22. Léo Lucas and Wessel P. J. van Woerden. On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 643–673. Springer, Cham, May / June 2022. https://doi.org/10.1007/978-3-031-07082-2_23. 2, 3, 10
- Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005. https://doi.org/10.1007/11535218_10. 3, 10, 12, 16
- FPST26. Andrea Flamini, Federico Pintore, Edoardo Signorini, and Giovanni Tognolini. New Straight-Line Extractable NIZKs for Cryptographic Group Actions. In Nadia Heninger and Mike Rosulek, editors, *CRYPTO 2026*, volume 16803 of *LNCS*, pages 632–664. Springer, 2026. https://doi.org/10.1007/978-3-032-35398-6_21. 13
- FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO’86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. https://doi.org/10.1007/3-540-47721-7_12. 3, 10
- 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. <https://doi.org/10.1145/1374376.1374407>. 5
- 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. https://doi.org/10.1007/978-3-319-70500-2_12. 7
- HPS23. 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. https://doi.org/10.1007/978-981-99-8733-7_11. 2, 3, 4, 19, 23
- 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. https://doi.org/10.1007/978-3-030-17653-2_6. 1
- Kat21. Shuichi Katsumata. A new simple technique to bootstrap various lattice zero-knowledge proofs to QROM secure NIZKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 580–610, Virtual Event, August 2021. Springer, Cham. https://doi.org/10.1007/978-3-030-84245-1_20. 10
- Ks22. Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022*,

- Part II, volume 13792 of *LNCS*, pages 279–309. Springer, Cham, December 2022. https://doi.org/10.1007/978-3-031-22966-4_10. 13
- LP11. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Berlin, Heidelberg, February 2011. https://doi.org/10.1007/978-3-642-19074-2_21. 2
- LWZ26. Xiaoru Li, Qi Wang, and Yue Zhou. The automorphism groups of random linear codes, 2026. <https://arxiv.org/abs/2607.23936v2>. 19
- MP13a. Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 21–39. Springer, Berlin, Heidelberg, August 2013. https://doi.org/10.1007/978-3-642-40041-4_2. 14
- MP13b. Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. Cryptology ePrint Archive, Report 2013/069, 2013. <https://eprint.iacr.org/2013/069>. 14
- MR04. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004. <https://doi.org/10.1109/FOCS.2004.72>. 14
- NAB⁺20. Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>. 3
- Pas03. Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337. Springer, Berlin, Heidelberg, August 2003. https://doi.org/10.1007/978-3-540-45146-4_19. 10
- PS23. Edoardo Persichetti and Paolo Santini. A new formulation of the linear equivalence problem and shorter LESS signatures. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VII*, volume 14444 of *LNCS*, pages 351–378. Springer, Singapore, December 2023. https://doi.org/10.1007/978-981-99-8739-9_12. 3, 10, 16, 17
- Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. <https://doi.org/10.1145/1060590.1060603>. 2, 5
- Sen97. Nicolas Sendrier. On the Dimension of the Hull. *SIAM Journal on Discrete Mathematics*, 10(2):282–293, 1997. <https://doi.org/10.1137/S0895480195294027>. 4
- Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994. <https://doi.org/10.1109/SFCS.1994.365700>. 1
- SS13. Nicolas Sendrier and Dimitris E. Simos. The hardness of code equivalence over and its application to code-based cryptography. In Philippe Gaborit, editor, *PQCrypto 2013*, pages 203–216. Springer, Berlin, Heidelberg, June 2013. https://doi.org/10.1007/978-3-642-38616-9_14. 5
- Udo26. Aleksei Udovenko. Solving the supersingular isogeny problem in time $p^{2/5+o(1)}$ using bivariate multipoint evaluation. Cryptology ePrint Archive, Paper 2026/1575, 2026. <https://eprint.iacr.org/2026/1575>. 39
- Unr15. Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 755–784. Springer, Berlin, Heidelberg, April 2015. https://doi.org/10.1007/978-3-662-46803-6_25. 10
- Vin54. Ivan Matveevich Vinogradov. *Elements of Number Theory*. Dover Publications, 1954. Translated from the Fifth Revised Edition by Saul Kravetz. 13
- Was82. Lawrence C. Washington. *Introduction to Cyclotomic Fields*, volume 83 of *Graduate Texts in Mathematics*. Springer, 1982. <https://doi.org/10.1007/978-1-4684-0133-2>. 13
- Wes26. Benjamin Wesolowski. The supersingular isogeny problem in time and memory $p^{1/3+o(1)}$. Cryptology ePrint Archive, Paper 2026/1486, 2026. <https://eprint.iacr.org/2026/1486>. 39

A Sampling Self-Orthogonal Codes

This section contains our original reduction from LWE to Hollow DSIS, which has since been replaced by Theorem 1 due to better parameters, as suggested by an anonymous reviewer at AC’26. We attach this

$\mathcal{B}(\mathbf{A}, \mathbf{b})$	$\text{Punch}(\mathbf{B}, \mathbf{u})$
$\mathbf{B}, \mathbf{u} \leftarrow \text{Randomise}(\mathbf{A}, \mathbf{b})$	Let $\bar{\mathbf{L}} \in \mathbb{F}_q^{(2k) \times (k-h)}$ first $k-h$ columns of $\mathbf{B}$
$\mathbf{B}_{(\pm)}, \mathbf{u}_{(\pm)} := \text{Shift}(\mathbf{B}, \mathbf{u})$	Let $\mathbf{R} \in \mathbb{F}_q^{(2k) \times h}$ remaining h columns of $\mathbf{B}$
$\mathbf{C}, \mathbf{v} \leftarrow \text{Punch}(\mathbf{B}_{(\pm)}, \mathbf{u}_{(\pm)})$	$\underline{\mathbf{L}} \leftarrow \mathbb{F}_q^{(k+h) \times (k-h)}; \mathbf{C}_{(0)} := [\bar{\mathbf{L}} \parallel \underline{\mathbf{L}}]$
$b' \leftarrow \mathcal{A}(\mathbf{C}, \mathbf{v})$	if $\dim \text{hull}(\mathbf{C}_{(0)}) \neq 0$: return $\perp$
return b'	for $i = 1, \dots, h$ do // sample hull that starts with $\mathbf{R}$
	$\mathbf{G}_{,-} := \text{SF}(\mathbf{C}_{(i-1)}^\perp)$
	if $\mathbf{G}_{,-}[2k, : 2k] \neq \mathbf{I}_{2k}$: return $\perp$ // pivot check
	$\mathbf{y}_{(i)} \leftarrow \text{SSO}_{2k}^i(\mathbf{G}, \mathbf{R}_i); \mathbf{C}_{(i)} := [\mathbf{C}_{(i-1)}, \mathbf{y}_{(i)}]$
	$\tilde{\mathbf{r}} \leftarrow \mathbf{D}_\times^{k+h};$ Let $\tilde{\mathbf{C}} \in \mathbb{F}_q^{(k+h) \times k}$ last $k+h$ rows of $\mathbf{C}_{(h)}$
	$\tilde{\mathbf{u}}^\top := \mathbf{u}^\top + \tilde{\mathbf{r}}^\top \cdot \tilde{\mathbf{C}}; \mathbf{C}, \mathbf{U} := \text{SF}(\mathbf{C}_{(h)})$
	$\mathbf{v}^\top = \tilde{\mathbf{u}}^\top \cdot \mathbf{U};$ return $\mathbf{C}, \mathbf{v}$
	$\text{SSO}_{2k}^i(\mathbf{G}, \mathbf{v})$ // $\mathbf{v} \in \mathbb{F}_q^{2k}, \mathbf{G} \in \mathbb{F}_q^{(3k+h) \times (2k+2h-i+1)}$
$\text{Randomise}(\mathbf{A}, \mathbf{b})$	$D := \langle \mathbf{g}_{2k+1}, \mathbf{g}_{2k+2} \rangle^2 - \langle \mathbf{g}_{2k+1}, \mathbf{g}_{2k+1} \rangle \cdot \langle \mathbf{g}_{2k+2}, \mathbf{g}_{2k+2} \rangle$
// $\text{LWE} \rightarrow \text{DSIS}$	if $D = 0$: reorder cols. of $\mathbf{G}[2k+1 :]$ to get ineq.
$\mathbf{U} \leftarrow \mathcal{GL}_k(\mathbb{F}_q); \mathbf{P} \leftarrow \mathcal{P}_{2k}$	$M := \text{MaxSols}(D);$ Let X, Y be vars. over $\mathbb{F}_q$
$\mathbf{B} := \mathbf{P} \cdot [\mathbf{A}^\top \parallel \mathbf{I}] \cdot \mathbf{U}$	for $\_ = 1, \dots, \text{halt} := k \cdot (q+1)$ do
$\mathbf{u}^\top := \mathbf{b}^\top \cdot \mathbf{U}$	$\mathbf{u} \leftarrow \mathbb{F}_q^{2h-i-1}; \bar{\mathbf{x}} := [\mathbf{v} \parallel X \parallel Y \parallel \mathbf{u}]$ // start with $\mathbf{v}$
return $\mathbf{B}, \mathbf{u}$	$\mathbf{rs} := \text{roots of } f(X, Y) := \bar{\mathbf{x}}^\top \cdot \mathbf{G}^\top \cdot \mathbf{G} \cdot \bar{\mathbf{x}}$
	$u \leftarrow [0, 1];$ if $u > \frac{ \mathbf{rs} }{M}$: continue
	$(x, y) \leftarrow \mathbf{rs}; \mathbf{x} := \bar{\mathbf{x}}(X := x, Y := y)$
	return $\mathbf{y} := \mathbf{G} \cdot \mathbf{x}$
	return $\perp$
$\text{Shift}(\mathbf{B}, \mathbf{u})$	
// $\text{binary} \rightarrow \text{signed}$	
$\mathbf{v} := 2\mathbf{u} - \mathbf{B}^\top \cdot [1]^{2k}$	
return $\mathbf{B}, \mathbf{v}$	
$\text{MaxSols}(D \neq 0)$	
// from [ABL25a, Fig. 3]	
if $D \in \text{QR}(\mathbb{F}_q)$:	
return $2q - 1$	
if $D \notin \text{QR}(\mathbb{F}_q)$:	
return $q + 1$	

Figure 32: Adversary $\mathcal{B}$ against binary-secret binary-error LWE from Theorem 5. In SSO_{2k}^i the two chief differences from SSO in [ABL25a, Def. 10] are highlighted.

reduction because it implies the first known sampler for self-orthogonal codes by Remark 7; thus we consider it independently interesting.

In what follows, we denote $\mathbf{D}_+ := \mathcal{U}(\{0, 1\})$ and $\mathbf{D}_\times := \mathcal{U}(\{\pm 1\})$. Let us show that hollow decision SIS is hard under binary-secret binary-error LWE.

Theorem 5 ($\text{LWE}_{k,k,q,\mathbf{D}_+}(\mathbf{D}_+^k) \rightarrow \text{DSIS}_{k,3k+h,q}^h(\mathbf{D}_\times^{3k+h})$). *Let $h, k, q \in \mathbb{Z}$ with $k \geq 1, 0 \leq h \leq k, q$ odd prime, and denote $n = 3k + h$. If there exists a (t, ε) -algorithm $\mathcal{A}$ against $\text{DSIS}_{k,n,q}^h(\mathbf{D}_\times^n)$ then there exists a $(t + \text{poly}(k, h), \varepsilon')$ -algorithm $\mathcal{B}$ in Fig. 32 against $\text{LWE}_{k,k,q,\mathbf{D}_+}(\mathbf{D}_+^k)$, and*

$$\varepsilon' \geq \varepsilon \cdot \left(1 - \frac{1}{q} - \frac{1}{q^2}\right) \cdot \left(1 - \frac{h}{e^k}\right) \cdot \left(1 - \frac{h}{q^h}\right).$$

Notice that we do not account for the square class χ of the instance matrix $\mathbf{A}$ in Theorem 5. If a fixed χ is desired, this can be done by checking χ on $\mathbf{C}_{(0)}$ along with the hull dimension in Punch in Fig. 32 and incurring a factor of $\frac{1}{2}$ in the success probability by [Ben26, Thm. 1]. However, much like for Hollow LWE, the square class does not play a role here as long as it is accounted for in the security proof when using ΔPCE .

Proof. The high-level proof idea is the following. We will first transform the input LWE sample into a ‘decision SIS’ sample and shift it from binary to signed. This signed decision SIS sample is then extended such that

the instance matrix is h -hollow. This is similar to Hollow LWE in [ABL25a, Thm. 1], except that we will add new *rows* to the (tall) instance matrix instead of columns, forcing us to ensure that the hull vectors we sample start with a column of the instance matrix, which is the crux of the proof. Let us first describe $\mathcal{B}$ and consider where it can fail, giving us a lower bound on its success probability. We will then show that, conditioned on not aborting, the output distribution of this transformation is the instance distribution of Hollow DSIS, which we give to $\mathcal{A}$ and return its bit.

Input and output. We give $\mathcal{B}$ in Fig. 32, and provide a graphic of the reduction in Fig. 33. It takes a pair $(\mathbf{A}, \mathbf{b})$ as input where $\mathbf{A} \leftarrow \mathbb{F}_q^{k \times k}$, and either $\mathbf{b} \leftarrow \mathbb{F}_q^k$ in case $b = 0$, or $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ for $\mathbf{s}, \mathbf{e} \leftarrow \{0, 1\}^k$ in case $b = 1$, and aims to distinguish which case it is in. It returns a bit b' indicating its guess. We adopt the convention that $\mathcal{B}$ outputs a uniformly random bit whenever any of its subroutines fail, indicated by this routine returning $\perp$.

From LWE to decision SIS. Using **Randomise**, the algorithm $\mathcal{B}$ first transforms a binary-secret binary-error LWE instance $(\mathbf{A}, \mathbf{b})$ into a binary decision SIS instance $(\mathbf{B} = \mathbf{P} \cdot [\mathbf{A}^T \parallel \mathbf{I}_k] \cdot \mathbf{U}, \mathbf{u} = \mathbf{U}^T \cdot \mathbf{b})$, where $\mathbf{U} \leftarrow \mathcal{GL}_k(\mathbb{F}_q)$ and $\mathbf{P} \leftarrow \mathcal{P}_{2k}$. When $b = 1$, then the output instance corresponds to $\mathbf{r}^T \cdot \mathbf{B} = \mathbf{u}^T$ with $\mathbf{r} = \mathbf{P} \cdot [\mathbf{s} \parallel \mathbf{e}]$, and to a uniform pair $(\mathbf{B}, \mathbf{u})$ when $b = 0$. Note $[\mathbf{A}^T \parallel \mathbf{I}] \cdot \mathbf{U}$ always contains a full-rank $k \times k$ submatrix at the bottom, which is not the case for random full-rank $2k \times k$ matrices. The random permutation $\mathbf{P}$ is therefore applied to randomise the pivots. This step runs in polynomial time, and always succeeds. Since random matrices from $\mathbb{F}_q^{2k \times k}$ are full-rank with probability at least $1 - \frac{k}{q^k}$, this distribution of $\mathbf{B}$ is statistically close (in k) to uniformly random $\mathbf{B} \in \mathbb{F}_q^{2k \times k}$.

From binary to signed. Further, using **Shift**, the algorithm $\mathcal{B}$ shifts the secret vector's domain (if $b = 1$) from $\{0, 1\}^{2k}$ to $\{\pm 1\}^{2k}$. The transform preserves the uniform distribution of $\mathbf{B}$ and also maps a uniform $\mathbf{u}$ to a uniform $\mathbf{v}$ if $b = 0$.

Extending to hollow. Next, in **Punch**, $\mathcal{B}$ ‘punches’ holes in $\mathbf{B}$ to construct an h -hollow matrix $\mathbf{C}$ while preserving the uniform distribution (now conditioned on hull dimension h). We refer to Fig. 33 for a visualisation. Observe that **Punch** outputs a code with hull dimension h if it succeeds. Indeed, each iteration of the loop for $i = 1, \dots, h$ calls the helper function SSO_{2k}^i that outputs a random self-orthogonal vector starting with the column vector $\mathbf{R}_i$ that is orthogonal to all columns from $\mathbf{C}_{(i-1)}$, since we are sampling from the dual $\mathbf{C}_{(i-1)}^\perp$ (we deal with the exact distribution below). The algorithm **Punch** aborts if $\mathbf{C}_{(0)}$ does not have hull dimension 0. By Lemma 1 this happens with probability at most $\frac{1}{q} + \frac{1}{q^2}$. When $h = k$ we treat $\mathbf{C}_{(0)}$ as the trivial code $\{\mathbf{0}\} \leq \mathbb{F}_q^n$ with the dual $\mathbb{F}_q^n$ spanned by $\mathbf{I}_n$, which is always 0-hollow. Moreover, **Punch** aborts whenever it does not encounter $\mathbf{C}_{(i-1)}^\perp$ for $i = 1, \dots, h$ of the correct shape, that is, with pivots in the first $2k$ columns (see ‘pivot check’). Since $2k + 2h - i + 1 \geq 2k + h$, the first $2k$ columns contain pivots with probability at least

$$\prod_{i=h+1}^k \left(1 - \frac{1}{q^i}\right) \geq 1 - \frac{1}{q^h} \cdot \sum_{i=1}^{k-h} \frac{1}{q^i} \geq 1 - \frac{1}{q^h},$$

where the first inequality is Weierstrass’ product inequality, and the second inequality is due to the sum being smaller than 1 for $q \geq 2$. Repeating this for each $i = 1, \dots, h$, we succeed with probability at least $1 - \frac{h}{q^h}$, again using Weierstrass’ product inequality. In each of the h repetitions, the algorithm can also abort in $\text{SSO}_{2k}^i(\mathbf{G}, \mathbf{v})$ by failing to sample a self-orthogonal vector with rejection, analogous to [ABL25a, Lem. 5]. Indeed, denoting $\gamma = \frac{q+1}{q}$, the probability of not failing to sample with rejection in all h repetitions is bounded from below by $1 - \frac{h}{\gamma^{k \cdot (q+1)}} \geq 1 - \frac{h}{e^k}$.

Uniform distribution. It remains to be shown that $(\mathbf{C}, \mathbf{v})$ are correctly distributed to apply our adversary $\mathcal{A}$, which expects $\mathbf{C}$ to be a uniformly random h -hollow code and $\mathbf{v}$ either $\mathbf{v}^T = \mathbf{r}^T \cdot \mathbf{C}$ for some $\mathbf{r} \leftarrow \mathcal{D}_\times^{3k+h}$ (if $b = 1$) or $\mathbf{v} \leftarrow \mathbb{F}_q^k$ (if $b = 0$). Since a code has hull dimension h if and only if it has a basis where

the first $k - h$ vectors form a 0-hollow code, and the last h vectors generate the hull, $\mathcal{B}$ clearly constructs each h -hollow code with a generator matrix that starts with $\mathbf{B}$ with non-zero probability. To see that each h -hollow code with a generator matrix that starts with $\mathbf{B}$ is equally likely to be sampled, it is enough to verify that the conic rejection sampler captured in the helper function $\text{SSO}_{2k}^i(\mathbf{G}, \mathbf{v})$, analogous to the one in [ABL25a, Def. 10], in each repetition $i = 1, \dots, h$ samples a uniformly random self-orthogonal vector $\mathbf{y}$ from the set of all self-orthogonal vectors in $\mathbf{C}_{(i-1)}^\perp$ with $\mathbf{y}[2k] = \mathbf{R}_i$. Using that $\mathbf{R}$ is statistically close to a uniformly random matrix together with [ABL25a, Cor. 1] will then conclude the proof.

Output of SSO_{2k}^i wrt. coefficients. Our argument is identical to the proof of [ABL25a, Lem. 5] and we repeat it here, except that the probabilities are conditioned on the sampled vector starting with $\mathbf{R}_i$ in the i -th repetition. First observe that the biased sampler $\text{SSO}_{2k}^i(\mathbf{G}, \mathbf{v})$ is exactly the same as the algorithm $\text{SSO}(\mathbf{G})$ from [ABL25a, Def. 10], except for two important changes which are highlighted in Fig. 32. The first is that when looking for two columns with non-zero discriminant, the algorithm $\text{SSO}_{2k}^i(\mathbf{G}, \mathbf{v})$ ignores the first $2k$ columns. Since the matrix $\mathbf{C}_{(i-1)}$ in iteration i is $(i-1)$ -hollow, hence so is $\mathbf{C}_{(i-1)}^\perp$ and thus $\mathbf{G}$, and there are $2h - i + 1$ columns of $\mathbf{G}$ we do not ignore, the matrix $\mathbf{G}[2k+1:]$ is at least $2h - 2i + 2 \geq 2$ hollow, hence we can always find two columns with non-zero discriminant by [ABL25a, Lem. 3]. The second change is that we set the first $2k$ coefficients of the sampled vector to $\mathbf{v}$. This clearly just conditions the uniform output distribution of the sampling algorithm on self-orthogonal vectors whose first $2k$ coefficients equal $\mathbf{v}$, but otherwise keeps the output distribution uniform.

Output of SSO_{2k}^i wrt. coordinates. Next observe that by the pivot check in **Punch**, we have in each iteration i that $\mathbf{G}$ is in column echelon form and starts with $\mathbf{I}_{2k}$ (otherwise we abort, which we have dealt with above). It follows that $\mathbf{y} \leftarrow \text{SSO}_{2k}^i(\mathbf{G}, \mathbf{R}_i)$ not only has the first $2k$ coefficients (with respect to $\mathbf{G}$) equal to $\mathbf{R}_i$, but also its first $2k$ coordinates. Thus $\mathbf{y}$ sampled in iteration i is uniformly random self-orthogonal from $\mathbf{C}_{(i-1)}^\perp$ conditioned on having the first $2k$ entries equal to $\mathbf{R}_i$. Indeed, by the rejection sampling step in $\text{SSO}_{2k}^i(\mathbf{G}, \mathbf{v})$ we have

$$\Pr[\mathbf{y} \text{ sampled in iter. } i] = \frac{1}{q^{2h-i-1}} \cdot \frac{1}{|\mathbf{rs}|} \cdot \frac{|\mathbf{rs}|}{M} = \frac{1}{q^{2h-i-1}} \cdot \frac{1}{M}.$$

*Output of **Punch**.* Since $\mathbf{B}$ is statistically close to uniformly random, so is $\mathbf{R}$, hence in each iteration i the sampled vector $\mathbf{y} \leftarrow \text{SSO}_{2k}^i(\mathbf{G}, \mathbf{R}_i)$ is indeed distributed statistically close to a uniformly random self-orthogonal vector in $\mathbf{C}_{(i-1)}^\perp$. By [ABL25a, Cor. 1], the output matrix $\mathbf{C}$ of **Punch** is statistically close to a uniformly random h -hollow matrix. The output syndrome $\mathbf{v}$ of **Punch** is clearly correctly distributed if $b = 1$, and uniform otherwise.

The final argument. Thus $\mathcal{B}$ may call $\mathcal{A}$ on the pair $(\mathbf{C}, \mathbf{v})$ it constructed, which is a correctly distributed Hollow DSIS instance, and output $\mathcal{A}$'s bit. The success probability ε' of $\mathcal{B}$ is the success probability ε of $\mathcal{A}$ multiplied by the probability $\mathcal{B}$ does not abort before calling $\mathcal{A}$, which by our analysis above is bounded as

$$\varepsilon' \geq \varepsilon \cdot \left(1 - \frac{1}{q} - \frac{1}{q^2}\right) \cdot \left(1 - \frac{h}{e^k}\right) \cdot \left(1 - \frac{h}{q^h}\right),$$

where the first factor is a lower bound on the probability that $\mathbf{C}_{(0)}$ is 0-hollow, the second factor is a lower bound on the probability of successfully sampling with rejection, and the third factor is a lower bound on the probability of successfully passing all pivot checks. $\square$

Remark 6. Notice that the term h in $n = 3k + h$ is tunable. Indeed, it only controls the probability that the first $2k$ pivots are not permuted away from the first $2k$ coordinates. More generally, we could reduce to $\text{DSIS}_{k,n,q}^h(\mathbf{D}_\times^n)$ where $n = 3k + \text{gap}$, and the corresponding factor in the drop in success probability would instead read $1 - \frac{h}{q^{\text{gap}}}$. The reason we pick $\text{gap} = h$ is that a concrete instantiation of the UPKE scheme from [ABL25a] with a security parameter λ requires $q^h \geq 2^\lambda$, implying that $1 - \frac{h}{q^h} = 1 - \text{negl}(\lambda)$.

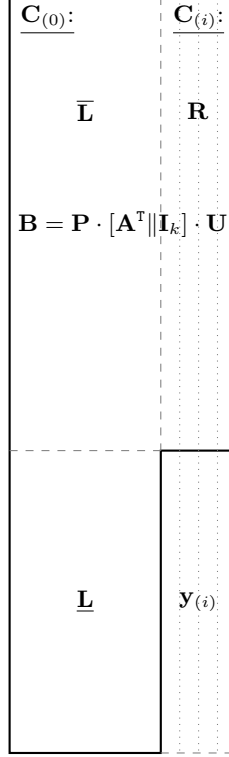


Figure 33: The transformation in **Punch** from Theorem 5.

Remark 7. Since Theorem 5 permits $h = k$, it also yields, to the best of our knowledge, the first algorithm for sampling uniformly random self-orthogonal codes of rate at most $\frac{k}{3k+\text{gap}}$. Moreover, if we do not require DSIS to be hard, we can simply omit the steps before calling **Punch** and give a uniformly random $\mathbf{B} \in \mathbb{F}_q^{k \times k}$ (and any $\mathbf{u}$) to **Punch** instead of $\mathbf{B} \in \mathbb{F}_q^{2k \times k}$ (as defined in Fig. 32). This obtains a sampler for uniformly random self-orthogonal codes of rate at most $\frac{k}{2k+\text{gap}}$, where at the very least $\text{gap} \geq 2$. For the proof to hold as is, we require full-rank $\mathbf{B}$, though the algorithm can be adapted to support non-full-rank $\mathbf{B}$ (by checking a rank condition inside the loop of **Punch**). We do not pursue this further because in our setting $\mathbf{B}$ is full rank by construction.

B Miscellaneous

B.1 Experimental Data for Assumption 2

Table 2 is produced by running `test_frob()` from `frobenius.py`. The experiment in every repetition samples an h -hollow matrix $\mathbf{A} \leftarrow \mathcal{D}_{n,k,q}^h$ (in various regimes discussed later) and acts on it with $\mathbf{O} \leftarrow \mathcal{O}_n(\mathbb{Z})$ to obtain $\mathbf{B}$. We then construct the linearised system $\mathbf{M}$ in Definition 24 and report its rank deficiency relative to Assumption 2, that is, we report $\delta = \min\{d \cdot k \cdot (n - k), n^2 - 1\} - \text{rank}(\mathbf{M})$. Note that when parameters are such that the expected rank of $\mathbf{M}$ is $n^2 - 1$, the kernel of $\mathbf{M}$ has $q^{\delta+1}$ elements. In all of our experiments, reported in Table 2, we most often have $\delta = 0$ but always $\delta \leq 1$. Thus the kernel of $\mathbf{M}$ is so small that $\mathbf{O}$ can be naïvely found with an exhaustive search. The primes we chose are $q \in \{5, 7, 17, 19\}$, two primes that are 1 modulo 4 and two primes that are 3 modulo 4, and we considered three parameter regimes. Each parameter set was tested on 10^3 samples. In the first parameter regime, we fix $(n, k, h) = (32, 8, 0)$ (Euclidean hull only) and then consider increasing $d \in \{2, 3, 4, 5\}$. In the second parameter regime, we fix $(n, k, d) = (16, 8, 2)$ and

Table 2: We report the difference δ between Assumption 2 and the observed rank of the linearised system. First two table sections: the behaviour when d increases. Next two table sections: the behaviour when h increases. Last table section: h is the Euclidean and the Hermitian hull dimension.

n	k	h	p^d	$\delta = 0$	$\delta = 1$	n	k	h	p^d	$\delta = 0$	$\delta = 1$
32	8	0	5^2	1000	0	32	8	0	7^2	1000	0
32	8	0	5^3	1000	0	32	8	0	7^3	1000	0
32	8	0	5^4	762	238	32	8	0	7^4	645	355
32	8	0	5^5	1000	0	32	8	0	7^5	1000	0
32	8	0	17^2	1000	0	32	8	0	19^2	1000	0
32	8	0	17^3	1000	0	32	8	0	19^3	1000	0
32	8	0	17^4	952	48	32	8	0	19^4	977	23
32	8	0	17^5	1000	0	32	8	0	19^5	1000	0
16	8	0	5^2	788	212	16	8	0	7^2	882	118
16	8	1	5^2	714	286	16	8	1	7^2	737	263
16	8	2	5^2	693	307	16	8	2	7^2	834	166
16	8	3	5^2	833	167	16	8	3	7^2	882	118
16	8	4	5^2	785	215	16	8	4	7^2	834	166
16	8	0	17^2	881	119	16	8	0	19^2	952	48
16	8	1	17^2	955	45	16	8	1	19^2	1000	0
16	8	2	17^2	929	71	16	8	2	19^2	881	119
16	8	3	17^2	808	192	16	8	3	19^2	978	22
16	8	4	17^2	952	48	16	8	4	19^2	856	144
16	4	2	5^2	1000	0	16	4	2	7^2	1000	0
16	4	2	17^2	1000	0	16	4	2	19^2	1000	0

then consider increasing $h \in \{0, 1, 2, 3, 4\}$ (again Euclidean hull only). In the third parameter regime, we fix $(n, k, h, d) = (16, 4, 2, 2)$, but h is taken to be the hull dimension with respect to both the Euclidean and the Hermitian inner product. Note that Assumption 2 does not cover the uniform distribution on the set of hollow codes with respect to multiple geometries; we include the third parameter regime because it represents a setting in which the attack from [BOST19] fails (since every Hermitian hull is large) while our linearisation attack seems to work.

B.2 Raw Data for Table 1

Produced by running `table(cores=8)` from `parameters.py`:

```

=== {'secpair': 128, 'p': 2, 'mode': 'self-dual'}:
[-MK +MT] UPKE(k: 610, n: 1220, q: 2^12, d: 1, p: 2): 906.5 KiB, h = 69, nkeys = 1, |pk| =
  ↳ 546.0, |ct| = 1.8, |mt| = 130.1 (mt: MT(dist: FixBer(33, 2)^32, trans: Fischlin with (4, 32,
  ↳ 0, 9), size: 130.1))
[+MK +MT] UPKE(k: 610, n: 1220, q: 2^12, d: 1, p: 2): 705.3 KiB, h = 84, nkeys = 16, |pk| =
  ↳ 559.4, |ct| = 1.8, |mt| = 131.5 (mt: MT(dist: FixBer(33, 2)^32, trans: Fischlin with (4, 32,
  ↳ 0, 9), size: 131.5))

=== {'secpair': 128, 'p': 2, 'sigma': 0.5, 'mode': 'self-dual'}:
[-MK +MT] UPKE(k: 490, n: 980, q: 2^9, d: 1, p: 2): 505.0 KiB, h = 65, nkeys = 1, |pk| =
  ↳ 264.3, |ct| = 1.1, |mt| = 99.9 (mt: MT(dist: FixBer(33, 2)^32, trans: Fischlin with (4, 32,
  ↳ 0, 9), size: 99.9))
[+MK +MT] UPKE(k: 490, n: 980, q: 2^9, d: 1, p: 2): 382.4 KiB, h = 80, nkeys = 16, |pk| =
  ↳ 272.4, |ct| = 1.1, |mt| = 101.2 (mt: MT(dist: FixBer(33, 2)^32, trans: Fischlin with (4, 32,
  ↳ 0, 9), size: 101.2))

=== {'secpair': 128, 'p': 2, 'mode': 'lhl'}:
[-MK +MT] UPKE(k: 420, n: 6825, q: 2^13, d: 1, p: 2): 6414.2 KiB, h = 131, nkeys = 1, |pk| =
  ↳ 4269.6, |ct| = 10.8, |mt| = 762.2 (mt: MT(dist: FixBer(33, 2)^32, trans: Fischlin with (4, 32,
  ↳ 0, 9), size: 762.2))

```

```

=== {'secpair': 128, 'p': 16, 'mode': 'lhl'}:
[-MK +MT] UPKE(k: 530, n: 10600, q: 216, d: 1, p: 16): 12345.4 KiB, h = 159, nkeys = 1, |pk| =
  ↪ 10425.1, |ct| = 20.7, |mt| = 1257.9 (mt: MT(dist: FixBer(33, 2)32, trans: Fischlin with (4, 32,
  ↪ 0, 9), size: 1257.9))

=== {'secpair': 128, 'p': 2, 'sigma': 0.5, 'mode': 'lhl'}:
[-MK +MT] UPKE(k: 530, n: 6625, q: 210, d: 1, p: 2): 5721.1 KiB, h = 130, nkeys = 1, |pk| =
  ↪ 3943.9, |ct| = 8.1, |mt| = 740.4 (mt: MT(dist: FixBer(33, 2)32, trans: Fischlin with (4, 32,
  ↪ 0, 9), size: 740.4))

```

C Code

C.1 Code for Section 6 (frobenius.py)

The script is also [attached](#). It depends on the script `selforthogonal.py` from [ABL25b].

```

from sage.all import GF, copy, matrix, random_matrix, span, vector

import selforthogonal as so # [EC:AlbBenLai25] Artifact at https://artifacts.iacr.org/eurocrypt/2025/a9/

import sys, time
from multiprocessing import Pool, Value

def _field_map_aut(a, i):
    """
    Map an element a from the extension field with its i-th automorphism over the base field.
    """
    d = a.parent().degree()
    i = i % d
    sigma_i = a.parent().frobenius_endomorphism(i)
    return sigma_i(a)

def map_aut(x, i):
    """
    Map an element a from the extension field E with its i-th automorphism over the base field
    F where i in ZZ_[E:F].
    """
    return vector([_field_map_aut(a, i) for a in x])

def dont_use_it_d_times(A, B, return_mat=False):
    """
    A lazy implementation of the Don't Use It Twice attack, using automorphisms to generate more
    samples. Outputs the possible solution space, if too big, otherwise attempts to recover the
    permutation (NotImplemented).
    """
    assert A.base_ring() == B.base_ring()
    E = A.base_ring()
    d = E.degree()
    A = A.echelon_form()
    B = B.echelon_form()
    L = A.tensor_product(so.dual(B))
    for i in range(1, d):
        Ai = matrix(E, [map_aut(x, i) for x in A])
        Bi = matrix(E, [map_aut(x, i) for x in B])
        L = L.stack(Ai.tensor_product(so.dual(Bi)))
    K = L.right_kernel_matrix()
    if K.rank() > 1:
        if return_mat:
            return K, L
        return K
    else:
        n = A.ncols()
        P = matrix(n, n, K[0])
        if return_mat:
            return P, L
        return P

counter, F, autos = None, None, None

```

```

def pool_init(c, field, gal_group):
    global counter, F, autos
    counter = c
    F = field
    autos = gal_group

def test_assumption1_worker(n, k, h, q, d, total):
    A = None
    while True:
        try:
            if total:
                A = SampleTotal(n, k, h, q, d)
            else:
                A = so.SampleCode(n, k, h, q ** d)
            if A is not None:
                break
        except:
            continue
    O = so.random_signed_permutation_matrix(n)
    A = A.echelon_form()
    B = (A * O).echelon_form()
    B_ = so.dual(B)
    constraints = A.tensor_product(B_).rows() # i=0
    for i in range(1, d):
        Ai = matrix(F, [[autos[i](x) for x in row] for row in A])
        Bi_ = matrix(F, [[autos[i](x) for x in row] for row in B_])
        constraints += Ai.tensor_product(Bi_).rows()
    M = matrix(F, constraints)
    deficiency = min(d * k * (n-k), n * n - 1) - M.rank()
    with counter.get_lock():
        counter.value += 1
    return deficiency

def test_assumption1(reps=100, n=16, k=8, h=2, q=17, d=2, total=False,
                    verbose=True, workers=8):
    """
    Test Assumption 1 for one set of parameters. Returns a map corank -> instances.
    """
    res = {}
    num_done = Value("i", 0)
    # prepare some sage objects in advance:
    field = GF(q ** d)
    autos = [field.frobenius_endomorphism(i) for i in range(d)]
    with Pool(processes=workers,
              initializer=pool_init,
              initargs=(num_done, field, autos)) as pool:
        labour = pool.starmap_async(test_assumption1_worker,
                                   [(n, k, h, q, d, total)] * reps)

    while not labour.ready():
        sys.stdout.write(f"\r{num_done.value}/{reps}")
        sys.stdout.flush()
        time.sleep(1)
    results = labour.get()
    for cr in results:
        if cr not in res:
            res[cr] = 1
        else:
            res[cr] += 1

    if verbose:
        print(f"\nExperiment on ({n}, {k}, {h}, {q}~{d}, total: {total}) with {reps}",
              f"repetitions, expected rank - actual rank: {res}")
    return res

def test_frob(reps=1000, workers=8, log="test.log"):
    """
    Test Assumption 1 on some parameters. Takes a long time.
    """
    params = [
        # small prime 1mod4 increasing degree
        {"n": 32, "k": 8, "h": 0, "q": 5, "d": 2},
        {"n": 32, "k": 8, "h": 0, "q": 5, "d": 3},
        {"n": 32, "k": 8, "h": 0, "q": 5, "d": 4},
        {"n": 32, "k": 8, "h": 0, "q": 5, "d": 5},
    ]

```

```

# small prime 3mod4 increasing degree
{"n": 32, "k": 8, "h": 0, "q": 7, "d": 2},
{"n": 32, "k": 8, "h": 0, "q": 7, "d": 3},
{"n": 32, "k": 8, "h": 0, "q": 7, "d": 4},
{"n": 32, "k": 8, "h": 0, "q": 7, "d": 5},
# prime 1mod4 increasing degree
{"n": 32, "k": 8, "h": 0, "q": 17, "d": 2},
{"n": 32, "k": 8, "h": 0, "q": 17, "d": 3},
{"n": 32, "k": 8, "h": 0, "q": 17, "d": 4},
{"n": 32, "k": 8, "h": 0, "q": 17, "d": 5},
# prime 3mod4 increasing degree
{"n": 32, "k": 8, "h": 0, "q": 19, "d": 2},
{"n": 32, "k": 8, "h": 0, "q": 19, "d": 3},
{"n": 32, "k": 8, "h": 0, "q": 19, "d": 4},
{"n": 32, "k": 8, "h": 0, "q": 19, "d": 5},
# small prime 1mod4 increasing hull
{"n": 16, "k": 8, "h": 0, "q": 5, "d": 2},
{"n": 16, "k": 8, "h": 1, "q": 5, "d": 2},
{"n": 16, "k": 8, "h": 2, "q": 5, "d": 2},
{"n": 16, "k": 8, "h": 3, "q": 5, "d": 2},
{"n": 16, "k": 8, "h": 4, "q": 5, "d": 2},
# small prime 3mod4 increasing hull
{"n": 16, "k": 8, "h": 0, "q": 7, "d": 2},
{"n": 16, "k": 8, "h": 1, "q": 7, "d": 2},
{"n": 16, "k": 8, "h": 2, "q": 7, "d": 2},
{"n": 16, "k": 8, "h": 3, "q": 7, "d": 2},
{"n": 16, "k": 8, "h": 4, "q": 7, "d": 2},
# prime 1mod4 increasing hull
{"n": 16, "k": 8, "h": 0, "q": 17, "d": 2},
{"n": 16, "k": 8, "h": 1, "q": 17, "d": 2},
{"n": 16, "k": 8, "h": 2, "q": 17, "d": 2},
{"n": 16, "k": 8, "h": 3, "q": 17, "d": 2},
{"n": 16, "k": 8, "h": 4, "q": 17, "d": 2},
# prime 3mod4 increasing hull
{"n": 16, "k": 8, "h": 0, "q": 19, "d": 2},
{"n": 16, "k": 8, "h": 1, "q": 19, "d": 2},
{"n": 16, "k": 8, "h": 2, "q": 19, "d": 2},
{"n": 16, "k": 8, "h": 3, "q": 19, "d": 2},
{"n": 16, "k": 8, "h": 4, "q": 19, "d": 2},
# all primes total hull
{"n": 16, "k": 4, "h": 2, "q": 3, "d": 2, "total": True},
{"n": 16, "k": 4, "h": 2, "q": 5, "d": 2, "total": True},
{"n": 16, "k": 4, "h": 2, "q": 17, "d": 2, "total": True},
{"n": 16, "k": 4, "h": 2, "q": 19, "d": 2, "total": True},
]
fd = None
if log is not None:
    fd = open(log, "w")
for p in params:
    n, k, h, q, d = p["n"], p["k"], p["h"], p["q"], p["d"]
    total = p.get("total", False)
    res = test_assumption1(reps=reps, n=n, k=k, h=h, q=q, d=d, verbose=True,
                          workers=workers)
    if log is not None:
        fd.write(f"{p} -> {res}\n")
        fd.flush()
if log is not None:
    fd.close()

# generate codes with large Euclidean and Hermitian hull, somehow:

def _field_coef_proj(a):
    """
    Return a list of coefficient projections of the field element a.
    """
    d = a.parent().degree()
    # list(p) gives a list of coefficients INCLUDING the 0s
    cs = list(a.polynomial())
    assert len(cs) <= d
    # pad with leading term coefficients with 0s
    if len(cs) < d:
        cs += [0] * (d - len(cs))
    return cs

def _field_coef_lift(x):

```

```

"""
Lift a vector over the base field F into an element of the extension field E. Degree
is determined by the length.
"""
d = len(x)
F = x.base_ring()
E = GF(len(F) ** d)
z = E.gen()
basis = vector(E, [z ** i for i in range(d)])
return E(x * basis)

def coefficient_code(x):
"""
Return a generator matrix for the coefficient code of the vector x from E^n.
"""
F = x.base_ring().base_ring()
pis = matrix(F, [_field_coef_proj(ent) for ent in x])
return pis.T

def coefficient_lift(A):
"""
Lift a matrix of elements in a prime field to a vector in the extension field.
Degree is determined by the number of rows.
"""
d = A.nrows()
F = A.base_ring()
E = GF(len(F) ** d)
A = A.T
v = vector(E, [_field_coef_lift(row) for row in A])
return v

def SampleTotal(n, k, h, p, d):
"""
[Figure 23]

Sample an [n,k]-linear code over GF(p ** d) such that it has total hull dimension h.
Requires d * k <= n/2. The code rate is upper-bounded by 1/(2*d).
"""
assert k >= 1 and h >= 0 and h <= k and 2 * k * d <= n
F, E = GF(p), GF(p ** d)
A_0 = None
for i in range(k):
    A_0 = random_matrix(E, k-h, n)
    if A_0.rank() == k-h or (A_0 * A_0.T).rank() == k-h:
        break
if A_0 is None:
    return None
if h == 0:
    return A_0
T_ = []
for row in A_0:
    T_ += list(coefficient_code(row))
A_i, H_ = matrix(F, T_), []
for i in range(d * h):
    A_i_ = so.Dual(A_i)
    y = None
    for j in range(d * k * p):
        y = so.SampleSelfOrthogonal(A_i_)
        if y is None:
            continue
        if i > 0 and y in span(so.hull(A_i)):
            y = None
            continue
        break
    if y is None:
        return None
    A_i = A_i_.stack(matrix(F, 1, n, y))
    H_.append(y)
hvs = [coefficient_lift(matrix(H_[d*i:d*(i+1)])) for i in range(h)]
A = matrix(E, hvs).stack(A_0)
return A

```

C.2 Code for Parameter Selection (parameters.py)

The script is also [attached](#). Table 1 can be reproduced by running `table(cores=8)` (and the output is also provided below the code).

```
from dataclasses import dataclass
from estimator.estimator import LWE, ND
from estimator.estimator.cost import Cost
from sage.all import (
    RR,
    binomial,
    ceil,
    divisors,
    euler_phi,
    is_even,
    is_power_of_two,
    is_prime_power,
    log,
    sqrt,
)
import sys
import os
import multiprocessing as mp

class HiddenPrints:
    def __enter__(self):
        self._original_stdout = sys.stdout
        sys.stdout = open(os.devnull, "w")

    def __exit__(self, *_):
        sys.stdout.close()
        sys.stdout = self._original_stdout

def _kb(v):
    """
    Convert bits to kilobytes.
    """
    return round(float(v / 8.0 / 1024.0), 1)

def hulldim(n, k, log_q, secpar=128):
    """
    Output a minimal hull dimension of a [n,k]-linear code over FF_q such that PCE is hard wrt.
    secpar.
    """

    def entropy(x):
        return RR(-x * log(x, 2) - (1 - x) * log(1 - x, 2))

    def CF_attack(n, k):
        return 0.5 * entropy(k / n) * n

    # hull collision attacks min h
    h_collision = 0
    for h in range(1, k + 1):
        cf = CF_attack(n, h)
        if cf >= secpar:
            h_collision = h
            break
    if h_collision == 0:
        raise ValueError("h_collision == 0")
    # hull attacks min h
    h_hull = max(ceil(secpar / log_q), ceil(secpar / log(n, 2)))
    if h_hull > k:
        raise ValueError(f"h_hull ({h_hull}) > k ({k})")
    # schur attack [cryptoeprint:2025/1017] min h
    h_schur = 1
    while log(binomial(n, h_schur - ceil(sqrt(2 * n))), 2) < secpar:
        h_schur += 1
    h_schur = max(h_schur, ceil(sqrt(2 * n)))
    if h_schur > k:
        raise ValueError(f"h_schur ({h_schur}) > k ({k})")
    return max(h_collision, h_hull, h_schur)
```

```

def mt_bernoullify_fs(n, secp=128, max_scale=10, step=1):
    """
    Search for smaller proof size in the FS-transformed NIZKPoK by switching the uniform challenge
    distribution to longer challenges of length l and fixed weight t such that
    l choose t >= 2 ** secp.
    """
    avg = secp // 2
    sp = n + n * ceil(log(n, 2)) # bit-size of signed permutation
    bin_unif_cost = secp + secp + (avg * secp + avg * sp)
    smallest_cost = {"dist": "unif", "chal_len": secp, "weight": None, "size": bin_unif_cost}
    for chals in range(secp, max_scale * secp + 1, step):
        ones = 0
        for ones_ in range(1, chals):
            if binomial(chals, ones_) >= 2**secp:
                ones = ones_
                break
        if ones == 0:
            continue
        size = secp + chals + (ones * sp + (chals - ones) * secp)
        if size < smallest_cost["size"]:
            smallest_cost = {"dist": "bern", "chal_len": chals, "weight": ones, "size": size}
    return smallest_cost

def mt_bernoullify_fischlin(n, secp=128, r=32, max_scale=10, step=1):
    """
    Search for smaller proof size in the Fischlin-transformed NIZKPoK with parameters (b, r, S=0, t) by
    ↪ switching the
    uniform challenge distribution (of 2 ** t challenges per r rounds with r*b=secp) to longer
    ↪ challenges of length
    l and fixed weight w such that l choose w >= 2 ** t.
    """
    b = ceil(secp / r)
    t = b + 5 if r <= 64 else b + 6 # rule of thumb [CiC:CheLin24b]

    avg = r * t // 2
    sp = n + n * ceil(log(n, 2)) # bit-size of signed permutation
    bin_unif_cost = secp + r * t + (avg * secp + avg * sp)
    smallest_cost = {"dist": "unif", "chal_len": t, "weight": None,
                    "size": bin_unif_cost, "params": (b, r, 0, t),}
    for chals in range(t, max_scale * t + 1, step):
        ones = 0
        for ones_ in range(1, chals):
            if binomial(chals, ones_) >= 2**t:
                ones = ones_
                break
        if ones == 0:
            continue
        size = secp + r * chals + (r * ones * sp + r * (chals - ones) * secp)
        if size < smallest_cost["size"]:
            smallest_cost = {"dist": "bern", "chal_len": chals, "weight": ones,
                            "size": size, "params": (b, r, 0, t),}
    return smallest_cost

class MTPParams:
    """
    Represents a member tag.
    """

    def __init__(self, secp, n, transform="Fischlin", r=32, nkeys=1):
        """
        Set up the member tag construction with the following parameters:
        - secp      -- the security parameter
        - n         -- dimension of the code
        - transform  -- either "FS" or "Fischlin", if "Fischlin" then r must be set
        - r         -- number of rounds, implicit when transform="FS"
        - nkeys     -- number of keys
        """
        self.distribution = ""
        self.size = 0
        self.transform = transform
        self.params = None
        self.n = n

        if transform == "FS":

```

```

        self._init_FS(secpar, n, nkeys)
    elif transform == "Fischlin":
        self._init_Fischlin(secpar, n, r, nkeys)
    else:
        raise NotImplementedError(f"transform {transform} not supported")

def _init_FS(self, secpar, n, nkeys):
    AA_proof = mt_bernoullify_fs(n + nkeys, secpar=secpar, m_min=secpar, m_max=10 * secpar, step=1)
    distribution = ""
    if AA_proof["dist"] == "unif":
        distribution += f"Unif([2 ** {str(AA_proof['chal_len'])}]))"
    else:
        distribution += f"FixBer({str(AA_proof['chal_len'])}, {str(AA_proof['weight'])})"
    self.distribution = distribution
    self.size = AA_proof["size"]

def _init_Fischlin(self, secpar, n, r, nkeys):
    AA_proof = mt_bernoullify_fischlin(n + nkeys, secpar=secpar, r=r)
    distribution = ""
    if AA_proof["dist"] == "unif":
        distribution += f"Unif([2 ** {AA_proof['chal_len']}])^{r}"
    else:
        distribution += f"FixBer({AA_proof['chal_len']}, {AA_proof['weight']})^{r}"
    self.distribution = distribution
    self.size = AA_proof["size"]
    self.params = AA_proof["params"]

def kb_size(self):
    return _kb(self.size)

def __repr__(self):
    return f"MT(dist: {self.distribution}, trans: {self.transform} with {self.params}, size: {self.size} \
    ↪ kb_size())"

@dataclass
class UPKEParams:
    """
    All solvers return an instance of this class.
    """
    secpar: int
    k: int
    n: int
    log_q: int
    p: int
    d: int
    lwe_costs: list

    def __repr__(self):
        return f"UPKE(k: {self.k:4d}, n: {self.n:4d}, q: 2^{self.log_q:2d}, d: {self.d:2d}, p: {self.p:2d} \
        ↪ {})"

    def pk(self, nkeys=1):
        return _kb(
            (self.n - self.k) * self.k * self.d * self.log_q + nkeys * self.k * self.d * self.log_q
        )

    def ct(self, nkeys=1):
        return _kb((self.n + nkeys) * self.log_q * self.d)

    def mt(self, nkeys=1):
        return MTParams(self.secpar, self.n, transform="Fischlin", r=32, nkeys=nkeys)

    def size(self, nkeys=1, kklen=None, include_mt=False):
        kklen = self.secpar if kklen is None else kklen
        if include_mt:
            s = (
                self.pk(nkeys=nkeys)
                + ceil(kklen / log(self.p, 2) / self.d / nkeys) * self.ct(nkeys=nkeys)
                + self.mt(nkeys=nkeys).kb_size()
            )
        else:
            s = self.pk(nkeys=nkeys) + ceil(kklen / log(self.p, 2) / self.d / nkeys) * self.ct(
                nkeys=nkeys
            )
        return s

```

```

def hulldim(self):
    return hulldim(self.n, self.k, self.log_q, self.secpair)

def display_str(self, nkeys=1, include_mt=True):
    m = self.mt(nkeys=nkeys)
    s = f"{self}:"
    s += f"{self.size(nkeys=nkeys, include_mt=include_mt):8.1f} KiB, "
    s += f"h = {self.hulldim() + nkeys:3d}, "
    s += f"nkeys = {nkeys}, "
    s += f"|pk| = {self.pk(nkeys=nkeys):7.1f}, "
    s += f"|ct| = {self.ct(nkeys=nkeys):4.1f}, "
    s += f"|mt| = {m.kb_size():7.1f} "
    s += f"(mt: {m})"
    return s

def display(self):
    print(self.display_str())

def _primitive_root_exists(f):
    """
    Return whether there exists a primitive root modulo f, that is iff:
    f {1, 2, 4, p^k, 2*p^k ; p odd prime, k NN}
    """
    if f < 1:
        return False
    if f in [1, 2, 4]:
        return True
    if is_power_of_two(f):
        return False
    if is_prime_power(f):
        return True
    if is_even(f) and is_prime_power(f // 2):
        return True
    return False

def upke_params(secpair=128, c=0.25, sigma=3.2, p=2, f=1, max_rel_h=1, mode="dsis", lwe_kwds=None):
    """
    Estimate UPKE parameters and sizes.

    :param secpair: target security parameter
    :param c: LHL constant (ignored if mode is dsis)
    :param sigma: standard deviation of the LWE error distribution
    :param p: message space size per slot (should be a power of 2)
    :param f: conductor of the number field
    :param max_rel_h: maximum hull dimension allowed as a fraction of the code dimension
    :param mode: either the lhl, the dsis, or the self-dual regime in picking the length of the code
    :param lwe_kwds: extra arguments to the lattice estimator

    :returns: 'UPKEParams' secure wrt. 'secpair'
    """
    if not _primitive_root_exists(f):
        raise ValueError(f"No primitive root exists for f: {f}")

    d = euler_phi(f)

    if max_rel_h > 1:
        raise ValueError(f"max_rel_h {max_rel_h} > 1")

    log_q = 0 # make the linter happier

    for k in range(60, 1500, 10):
        # LWE correctness condition

        if mode == "lhl":
            for log_q in range(3, 31):
                if 2**log_q / sqrt(log_q) >= p**2 / 2 + p * sigma * sqrt(
                    (secpair + 1) * (1 + c) * k
                ):
                    break
            elif mode == "dsis":
                for log_q in range(3, 31):
                    if 2**log_q >= p**2 / 2 + p * sigma * sqrt((secpair + 1) * (3 * k + secpair / log_q)):
                        break

```

```

elif mode == "self-dual":
    for log_q in range(3, 31):
        if 2**log_q >= p**2 / 2 + p * sigma * sqrt((secpair + 1) * (2 * k)):
            break
    else:
        raise NotImplementedError(f"mode: {mode}")

# either LHL or DSIS condition
n = None
halt = 5
if mode == "lhl":
    # NOTE ignoring - secpair/log(n)
    n = ceil((1 + c) * k * log_q)
elif mode == "dsis":
    n = 3 * k
    try:
        h = hulldim(n, k, log_q, secpair)
    except ValueError:
        continue
    if h > k * max_rel_h:
        continue
    count = 0
    while count < halt:
        count += 1
        n = 3 * k + min(ceil(secpair / log_q), ceil(h / log_q))
        try:
            h_ = hulldim(n, k, log_q, secpair)
        except ValueError:
            continue
        if h == h_:
            break
        h = h_
elif mode == "self-dual":
    n = 2 * k
    try:
        h = hulldim(n, k, log_q, secpair)
    except ValueError:
        continue
    if h > k * max_rel_h:
        continue
    else:
        raise NotImplementedError(f"mode: {mode}")
    try:
        hulldim(n, k, log_q, secpair)
    except ValueError:
        continue

lwe_pk = LWE.Parameters(
    n=k * d,
    m=k * d,
    q=2**log_q,
    Xs=ND.UniformMod(2),
    Xe=ND.UniformMod(2),
)

lwe_enc = LWE.Parameters(
    n=k * d,
    m=n * d,
    q=2**log_q,
    Xs=ND.UniformMod(2**log_q),
    Xe=ND.DiscreteGaussian(sigma),
)

# NOTE these won't matter
deny_list = ("arora-gb", "bkw", "bdd_hybrid", "bdd_mitm_hybrid")
with HiddenPrints():
    if lwe_kwds is None:
        lwe_kwds = {}

    costs_enc = LWE.estimate(lwe_enc, deny_list=deny_list, **lwe_kwds)

    if mode != "lhl":
        costs_pk = LWE.estimate(lwe_pk, deny_list=deny_list, **lwe_kwds)
    else:
        costs_pk = {"foo": Cost(rop=2**secpair)} # fake

if (min(cost["rop"] for cost in costs_pk.values()) >= 2**secpair) and (

```

```

        min(cost["rop"] for cost in costs_enc.values()) >= 2**secpair
    ):
        upke = UPKEParams(secpair, k, n, log_q, p, d, [costs_pk, costs_enc])
        return upke
    raise ValueError("No working parameters found.")

def find_optimal_nkeys(upke, kklen=None):
    """
    Finds number of keys to encrypt secpair bits which minimises
    |pk| + |ctxt| + |mt|
    """
    kklen = upke.secpair if kklen is None else kklen
    s = None
    n = 0
    for nkeys in divisors(ceil(kklen / log(upke.p, 2))):
        s_ = upke.size(nkeys=nkeys, kklen=kklen, include_mt=True)
        if n == 0:
            n, s = nkeys, s_
            continue
        if s_ <= s:
            n, s = nkeys, s_
            continue
    return n, s

def process_row(row):
    return upke_params(
        secpair=row["secpair"],
        p=row.get("p", 2),
        sigma=row.get("sigma", 3.2),
        mode=row["mode"],
        max_rel_h=1,
    )

def table(cores=None):
    """
    Table 1

    :param cores: Number of cores to use.

    """
    rows = [
        {"secpair": 128, "p": 2, "mode": "self-dual"},
        {"secpair": 128, "p": 2, "sigma": 0.5, "mode": "self-dual"},
        {"secpair": 128, "p": 2, "mode": "lhl"},
        {"secpair": 128, "p": 16, "mode": "lhl"},
        {"secpair": 128, "p": 2, "sigma": 0.5, "mode": "lhl"},
    ]

    with mp.Pool(cores) as pool:
        results = pool.map(process_row, rows)

    for i in range(len(rows)):
        upke = results[i]
        row = rows[i]
        print()
        print(f"=== {row}:")
        print("[-MK +MT]", upke.display_str(nkeys=1, include_mt=True))
        n, _ = find_optimal_nkeys(upke)
        print("[+MK +MT]", upke.display_str(nkeys=n, include_mt=True))

if __name__ == "__main__":
    table(cores=8)

```