Hollow LWE: A New Spin

Unbounded Updatable Encryption from LWE and PCE

Martin R. Albrecht[1⋆], Benjamin Benčina[2⋆⋆], 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) allows anyone to update a public key while simultaneously producing an update token, given which the secret key holder could consistently update the secret key. Furthermore, ciphertexts encrypted under the old public key remain secure even if the updated secret key is leaked – a property much desired in secure messaging. All existing lattice-based constructions of UPKE update keys by a noisy linear shift. As the noise accumulates, these schemes either require super-polynomial-size moduli or an a priori bounded number of updates to maintain decryption correctness.

Inspired by recent works on cryptography based on the lattice isomorphism problem, we propose an alternative way to update keys in lattice-based UPKE. Instead of shifting, we rotate them. As rotations do not induce norm growth, our construction supports an unbounded number of updates with a polynomial-size modulus. The security of our scheme is based on the LWE assumption over hollow matrices – matrices which generate linear codes with non-trivial hull – and the hardness of permutation code equivalence. Along the way, we also show that LWE over hollow matrices is as hard as LWE over uniform matrices, and that a leftover hash lemma holds for hollow matrices.

1 Introduction

In secure (group) messaging protocols, a strongly desired notion is *forward security* – messages sent before a compromise remain unknown to the attacker. Perhaps the most known example is that of the ephemeral Diffie-Hellman key exchange, where the two parties use fresh secret values a and b every interaction and exchange (authenticated with a long-term secret) public values g^a and g^b to form a shared secret $g^{a \cdot b}$. Forward secrecy is guaranteed by the decisional Diffie-Hellman assumption, even if the authentication mechanism is also compromised. However, this requires interaction at every key exchange, so a more desirable solution is that of a forward-secure (FS) public-key encryption (PKE) [CHK03].

In a FS-PKE, the key generation algorithm outputs an initial key-pair $(\mathsf{pk}_0, \mathsf{sk}_0)$, after which every subsequent public key pk_i can be computed from the previous public key pk_{i-1} (by anyone) and similarly every subsequent secret key sk_i can be computed from the previous secret key sk_{i-1} (only by the initial holder). The central property of a FS-PKE is that even if sk_i is compromised, any message encrypted under pk_j for $j < i$ remains hidden to the attacker, in other words it is only possible to move *forwards* in the secret key chain $\mathsf{sk}_0 \to \mathsf{sk}_1 \to \cdots$, but not backwards.[4] Unfortunately, forward secure schemes typically suffer in efficiency. It has been argued [DJK22] that the update mechanism, which yields the chains of public and secret keys, is unnecessarily strict and requires the parties to stay synchronised, which is not always suitable in the context of secure messaging.

⋆ The research of Albrecht was supported by UKRI Grant EP/Y02432X/1.
⋆⋆ The research of Benčina was supported by UKRI Grant EP/S021817/1.
⋆⋆⋆ The research of Lai was supported by Research Council of Finland grant 358951.

[4] Another desired property is *post-compromise security*, which informally refers to the time required for the protocol to "heal", that is how far can the adversary traverse the secret key chain after the compromise has ended.

1.1 Updatable Public-Key Encryption

To remedy these concerns, the notion of *updatable public-key encryption* (UPKE) was introduced by [JMM19] and [ACDT20] as a slight relaxation of forward security. A UPKE is a PKE scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ that comes with an update mechanism in the form of two additional algorithms UpdPk and UpdSk. Instead of having strict chains of public and private keys, any sender, including a malicious one, can decide to update another user's public key pk by running UpdPk and generate a new public key pk' along with an *update token* up, which the receiver can use to update their corresponding secret key sk to sk' by running UpdSk.

A UPKE scheme is deemed secure if it admits a relaxed notion of forward security. Namely, a UPKE scheme is secure if an exposed secret key sk' does not reveal the messages encrypted under any prior public key, provided at least one honest update was applied in the update chain leading to sk', that is an update whose randomness is not controlled by the adversary.

Both of the approaches of [JMM19] and [ACDT20], while having major efficiency improvements over Forward-Secure PKE schemes, rely on the hardness of discrete logarithms, which is not post-quantum secure. The authors of [DKW21] then constructed the first post-quantum secure UPKE scheme; like ours, it is based on the dual-Regev PKE [GPV08], and proved it secure in the standard model. Their security proof, however, relies on the noise-drowning technique and thus requires a super-polynomial modulus. The number of updates is also bounded, since each update adds a small random vector to the (short) secret key, making it longer with every update. Recently, a construction proposed in [HPS23] requires only a polynomial modulus while also relying on the learning with errors (LWE) assumption for security; however, the growing noise issue persists, as they again support only a bounded number of updates.

1.2 Our Contributions

We construct a UPKE scheme whose IND-CPA security as a PKE is based on the LWE assumption: it is a mild variant of the dual-Regev PKE scheme. Briefly, LWE states that $(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$ and $(\mathbf{A}, \mathbf{u})$ are indistinguishable, where $\mathbf{A}$ is a (possibly tall) random matrix mod q, $\mathbf{s}$ is a random vector mod q and $\mathbf{e}$ a vector with entries following a (discrete) Gaussian distribution. Our scheme's update mechanism is based on the (decision) permutation code equivalence (PCE) assumption, in the Random Oracle Model (ROM). Briefly, PCE states that under some conditions (see below), $(\mathbf{A}, \mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U})$ and $(\mathbf{A}, \mathbf{B})$ are indistinguishable for $\mathbf{A}, \mathbf{B}$ tall random matrices mod q, where $\mathbf{O}$ is a permutation matrix and $\mathbf{U}$ is an invertible matrix. In other words, the IND-CR-CPA security (Definition 7) of our UPKE follows from the LWE and PCE assumptions in the ROM. Both LWE and PCE are conjectured to be post-quantum secure (under certain caveats we discuss below). Since the update mechanism consists of applying a uniformly random signed permutation to the key-pair, updating is "free" in the sense that there is no accumulation of small noise in the key material that eventually renders it useless (such as in all prior LWE-based schemes). As a consequence, there is no practical upper bound on the number of updates in our construction.

The interplay of LWE and PCE is complicated by "hull attacks". Concretely, a uniformly random tall matrix $\mathbf{A}$ over $\mathbb{Z}_q$ forming a dual-Regev public key can be seen as a column generator matrix of a linear code over $\mathbb{Z}_q$. It is well-known that random codes have small hull dimension [Sen97], in fact the hull is most likely trivial as soon as $q > 2$. There exist algorithms for PCE that are exponential in the dimension of the hull but polynomial in every other parameter, providing de facto polynomial solvers for PCE on random codes [Sen00,BOST19]: The complexity of the support splitting algorithm [Sen00] is proportional to q^h where h is the hull dimension, and the complexity of the BOS algorithm [BOST19] is proportional to n^h where n is the length of the code. Hence, we need to restrict ourselves to matrices whose columns generate linear codes with a hull dimension h that push these complexities above 2^λ for the desired security parameter λ.[5]

To this end, we first give an algorithm for sampling a uniformly random code with a desired hull dimension. We were not able to find such an algorithm anywhere in literature, although sampling random codes with a sufficiently high hull dimension has been reported to be efficient (e.g. [BBPS21]).We then show that there exists an efficient reduction from LWE to LWE with the promise that the instance matrix generates a linear

[5] We review this literature in Appendix A.

code with hull dimension h. This allows us to replace uniformly random dual-Regev public key matrices with matrices that generate codes with a desired hull dimension. We also show that if the codewords are long enough, a version of the Leftover Hash Lemma (LHL) is true, which allows us to sample dual-Regev key-pairs where the public key matrix generates a linear code with a specified hull dimension.

We present example parameters in Table 1. As can be seen from that table, our parameters are not practical. The central reasons for this are (a) the reliance on a variant of the LHL instead of LWE for the security of the public key and (b) instantiating our scheme over $\mathbb{Z}$ instead of some more general ring of integers of a cyclotomic number field. As a consequence, our parameters as is are worse when compared to [HPS23] for all number of updates considered in their work. We note that the parameters in [HPS23] are given for Module LWE rather than plain LWE. We refer to Section 1.4 for more discussions.

Table 1: Parameters for the given λ and p with $c = 0.25$ and $s = 8$.

λ	p	n	$k \log_2(q)$	h	$\lvert\mathsf{ctxt}\rvert$	$\lvert\mathsf{up}\rvert$
128	2	7313	450	13	27	11.6 KiB	1485.7 KiB
128	16	11000	550	16	26	21.5 KiB	687.6 KiB
192	32	20250	900	18	37	44.5 KiB	1708.7 KiB
256	32	29688	1250	19	48	68.9 KiB	3525.6 KiB
[HPS23] with 2^{20} updates						
128	–	–	–	36	–	9.1 KiB	27 KiB

1.3 Technical Overview

Consider the inhomogeneous dual-Regev PKE scheme, where the secret key is a short vector $\mathbf{r} \in \mathbb{Z}_q^n$, and the public key is a uniformly random tall matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times k}$ and a vector $\mathbf{u} \in \mathbb{Z}_q^k$ such that $\mathbf{r}^\intercal \cdot \mathbf{A} = \mathbf{u}^\intercal$. The current paradigm of lattice-based UPKE schemes [DKW21,HPS23] has been to somehow *add* a small random noise ρ to the public key. This noise is then encrypted as an update token in UpdPk, and in UpdSk ρ is decrypted and added to the secret key in a way that makes the new key-pair is valid.

Our main idea is to instead take a random lattice isometry $\mathbf{O} \in \mathcal{O}_n(\mathbb{R})$ and *rotate* the public key into $\mathbf{A}' = \mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U}$ and $\mathbf{u}' = \mathbf{U}^\intercal \cdot \mathbf{u}$, where $\mathbf{U}$ is a wlog. the systematic form transformation, but can in principle be any random basis change. The update token is then just the encryption of the seed ρ for $\mathbf{O}$ – i.e. we have $\mathbf{O} = \mathsf{H}(\rho)$ where H is modelled as a Random Oracle – under $\mathbf{A}$, and the new secret key is computed again by rotation $\mathbf{r}' = \mathbf{O} \cdot \mathbf{r}$. Since $\mathbf{O}^\intercal \cdot \mathbf{O} = \mathbf{I}$ and $\mathbf{U}$ is full-rank, the new key-pair satisfies $\mathbf{r}'^\intercal \cdot \mathbf{A}' = \mathbf{u}'^\intercal$, and crucially $\|\mathbf{r}'\| = \|\mathbf{r}\|$, i.e. its quality does not deteriorate. As we are dealing with q-ary lattices (by completing the basis $\mathbf{A}$ with $q \cdot \mathbf{e}_i$), we require that the updated keys are integral as well, so we limit ourselves to $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$ which is a finite group of signed permutations, that is the automorphisms of $\mathbb{Z}^n$. As far as we are aware, no prior work combines "established" lattice techniques such as LWE with those developed around the Lattice Isomorphism Problem (LIP) [DvW22]. Note that LIP over q-ary lattices (with q prime) and restricted to integral isometries is precisely the signed permutation equivalence problem (SPCE) for linear codes over $\mathbb{Z}_q$. As SPCE is easy for random codes due to hull attacks, we limit ourselves to public keys $\mathbf{A}$ that generate a code with hull dimension h big enough so that hull attacks are not efficient, and we show how to do this securely in Sections 3 and 4.

The security proof proceeds in four stages. First, we simplify the IND-CR-CPA game by showing that the update oracle can be simulated. We also plant a "bad" query into the RO, i.e. the seed ρ for $\mathbf{O}$, giving the adversary a free win if they ever query it. The adversary's view, omitting $\mathbf{u}$ for brevity, consists of

$$\Big(\mathsf{pk}_0 = \mathbf{A},\ \mathsf{pk} = \mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U},\ \mathsf{sk} = \mathbf{O} \cdot \mathbf{r},\ \mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathbf{A}, \mathsf{msg}_b),\ \mathsf{up} \leftarrow \mathsf{Enc}(\mathbf{A}, \rho)\Big).$$

Notice that giving sk to the adversary leaks the multiset of entries of $\mathbf{r}$, up to signs. To eliminate this, we choose $\mathbf{r} \in \{\pm 1\}^n$. It will become clear why in the next stage.

Second, we replace the updated pk and sk with a fresh key-pair $(\mathbf{B}, \mathbf{r}')$. This is indistinguishable if PCE is hard.[6] Indeed, take a decision-PCE instance $(\mathbf{G}, \mathbf{H})$ and denote by $\mathbf{u}, \mathbf{v}$ the sum of their respective rows. Thus, the all 1's vector $[1]^n$ is a valid secret key for both $(\mathbf{G}, \mathbf{u})$ and $(\mathbf{H}, \mathbf{v})$, which crucially is fixed by any permutation. We now randomise the instance to make it a proper SPCE instance by multiplying each code with a random isometry, that is we sample $\mathbf{O_G}$ and $\mathbf{O_H}$ and construct $\tilde{\mathbf{G}} = \mathbf{O_G} \cdot \mathbf{G}$, $\mathbf{r} = \mathbf{O_G} \cdot [1]^n$, $\tilde{\mathbf{H}} = \mathbf{O_H} \cdot \mathbf{H}$, $\mathbf{r}' = \mathbf{O_H} \cdot [1]^n$. Distinguishing equivalent $(\tilde{\mathbf{G}}, \tilde{\mathbf{H}})$ from a random pair is thus as hard as solving decision-PCE for $(\mathbf{G}, \mathbf{H})$, even if we leak $\mathbf{r}'$ since it is uniformly random in $\{\pm 1\}^n$. The adversary's view is the following:

$$\Big(\mathsf{pk}_0 = \mathbf{A},\ \mathsf{pk} = \mathbf{B},\ \mathsf{sk} = \mathbf{r}',\ \mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathbf{A}, \mathsf{msg}_b),\ \mathsf{up} \leftarrow \mathsf{Enc}(\mathbf{A}, \rho)\Big).$$

Note that the challenger still knows a corresponding secret key $\mathbf{r}$ such that $\mathbf{r}^\intercal \cdot \mathbf{A} = \mathbf{u}$. We abandon this structure in the third stage of the proof by changing $(\mathbf{A}, \mathbf{u})$ everywhere it is used for $(\mathbf{A}', \mathbf{u}')$ where $\mathbf{u}'$ is uniformly random. To show this is indistinguishable we use a version of LHL for matrices with a hull. In the fourth and final stage, we replace all LWE ciphertexts with random values, so the adversary's view is

$$\Big(\mathsf{pk}_0 = \mathbf{A}',\ \mathsf{pk} = \mathbf{B},\ \mathsf{sk} = \mathbf{r}',\ \mathsf{ctxt} \leftarrow\!\!\$\ \mathbb{Z}_q^n,\ \mathsf{up} \leftarrow\!\!\$\ \mathbb{Z}_q^{n \times \lambda}\Big).$$

Since the only way to win now is to query $\mathsf{RO}(\rho)$, the advantage of the adversary in the last game is $2^{-\lambda}$, thus the scheme is IND-CR-CPA secure in the ROM assuming LWE and PCE for codes with hull dimension h is hard.

1.4 Open Problems

While the composition of LWE with PCE enables updatable public-key encryption with parameters that are independent of the number of updates, the concrete performance of our construction for realistic choices of settings remains low. There are several avenues to improving performance.

Avoiding the Leftover Hash Lemma. Our reduction relies on the LHL to argue that we may "forget" the secret key in the security experiment. This necessitates parameters $n = (1 + c) \cdot k \cdot \log q$ which in turn necessitate larger $q > \sqrt{n} \cdot \sigma$ to "accommodate" the noise. Replacing the invocation of the LHL with an application of some LWE-like assumption, cf. [LP11], would reduce parameters by more than an order of magnitude. While in this work we show that $(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$ remains indistinguishable from random, this change would require us showing that $(\mathbf{A}, \mathbf{r}^\intercal \cdot \mathbf{A})$ remains computationally indistinguishable for dimensions shorter than those required for an LHL-type argument.

Larger distributions of secret keys. Our reduction makes critical use of $\chi = \mathcal{U}(\{\pm 1\})$ when sampling the secret key $\mathbf{r} \leftarrow\!\!\$\ \chi^n$. This is because that distribution is guaranteed to leak no information about $\mathbf{O}$. Yet, while sampling the secret from a discrete Gaussian as $\mathbf{r} \leftarrow\!\!\$\ \mathcal{D}_{\mathbb{Z}^m, \sigma}$ and outputting $\mathbf{O} \cdot \mathbf{r}$ would leak information about $\mathbf{O}$ it seems plausible the underlying PCE problem remains hard. In other words, it seems plausible that PCE offers some form of leakage resilience. Establishing this would allow for smaller parameters.

Multi-key variant. Standard LWE-style reductions proving multi-key instances secure, e.g. $\mathbf{r}_i^\intercal \cdot \mathbf{A} = \mathbf{u}_i^\intercal \bmod q$ do not translate to our setting. This is because our reduction hardcodes $\mathbf{r} = [1]^n$ at one step and later randomises it using a diagonal matrix. Overcoming this limitation would allow to amortise the cost of encrypting λ bits.

[6] Note that we plant a PCE instance, not an SPCE instance.

Module variant. A natural extension is to consider a module variant of our construction. We note that while e.g. [HPS23] simply assumes their security proof translates from $\mathbb{Z}$ to $\mathcal{R} \coloneqq \mathbb{Z}[x]/f(x)$, where $f(x)$ is a cyclotomic polynomial, such an assumption would be more questionable in our setting. This is because we are not aware of any work studying PCE over modules, i.e. over $\mathcal{R}_q^{n\times k}$. Studying this problem would allow using the Module-LWE machinery [LS15]. We note that PCE is defined also over finite extension fields $\mathbb{F}_{q^n}$.

2 Preliminaries

We write $[\mathbf{A}\|\mathbf{B}]$ for $[\mathbf{A}^\mathsf{T} \mid \mathbf{B}^\mathsf{T}]^\mathsf{T}$, i.e. for stacking $\mathbf{A}$ on top of $\mathbf{B}$. We call an affine conic $F(x,y) = a\cdot x^2 + b\cdot xy + c\cdot y^2 + d\cdot x + e\cdot y + f$ over $\mathbb{Z}_q$ *smooth* if $D = b^2 - 4\cdot a\cdot c \neq 0 \pmod q$. A smooth affine conic over $\mathbb{Z}_q$ admits either $q-1$ or $2\cdot q-1$ solutions if $D \in \mathrm{QR}(\mathbb{Z}_q)$, and either 1 or $q+1$ solutions if $D \notin \mathrm{QR}(\mathbb{Z}_q)$ (see Appendix B).

Let q be a prime, and $n \geq k > 0$ be positive integers. An $[n,k]$-linear code $\mathfrak{C}$ over $\mathbb{Z}_q$ of length n and dimension k is a k-dimensional vector subspace of $\mathbb{Z}_q^n$. Its vectors are called codewords, and it is typically represented by a full-rank generator matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times k}$ whose columns generate $\mathfrak{C}$. As we can always choose a different basis, any $\mathbf{G}' = \mathbf{G}\cdot\mathbf{U}$ for $\mathbf{U} \in \mathcal{GL}_k(\mathbb{Z}_q)$ is also a generator matrix for $\mathfrak{C}$. The *systematic form* of $\mathfrak{C}$ is the unique generator matrix $\mathbf{S}$ for $\mathfrak{C}$ such that there exists a permutation matrix $\mathbf{P}$ so that $\mathbf{P}\cdot\mathbf{S}$ is of the form $\left[\mathbf{I_k}\|\mathbf{M}\right]$, where $\mathbf{M} \in \mathbb{Z}_q^{(n-k)\times k}$, and is thus the canonical choice of basis. For a generator matrix $\mathbf{G}$ of $\mathfrak{C}$, we denote by $\mathrm{SF}(\mathbf{G})$ the systematic form of $\mathfrak{C}$ computed by column-reducing $\mathbf{G}$.

A code may equivalently be represented as the kernel of its parity check matrix $\mathbf{H} \in \mathbb{Z}_q^{n\times(n-k)}$, that is we have $\mathbf{v} \in \mathfrak{C} \iff \mathbf{v}^\mathsf{T}\cdot\mathbf{H} = 0$. If $\left[\mathbf{I_k}\|\mathbf{M}\right]$ is the systematic form of $\mathfrak{C}$, then $\left[-\mathbf{M}^\mathsf{T}\|\mathbf{I_{n-k}}\right]$ is a parity check matrix for $\mathfrak{C}$. The code generated by $\mathbf{H}$ is called the dual code to $\mathfrak{C}$, i.e. $\mathfrak{C}^\perp = \{\mathbf{y} \in \mathbb{Z}_q^n \,;\, \forall \mathbf{x} \in \mathfrak{C}\colon \mathbf{x}^\mathsf{T}\cdot\mathbf{y} = 0\}$. The generator matrices of $\mathfrak{C}^\perp$ are precisely the parity check matrices of $\mathfrak{C}$, vice versa.

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

Lemma 1 (Special case of [Sen97, Thm. 4.18]). *Let n,k be positive integers with $2\cdot k \leq n$, and let q be prime. The probability that a randomly sampled $[n,k]$-linear code over $\mathbb{Z}_q$ has a trivial hull is*

$$\prod_{i=1}^{\lceil \frac{k}{2}\rceil}\left(1-\frac{1}{q^{2\cdot i-1}}\right)\cdot\left(1+\mathcal{O}\left(\frac{k}{q^{\frac{n}{2}}}\right)\right) \geq 1-\frac{1}{q}-\frac{1}{q^2}.$$

In particular, this is always greater than $\frac{1}{2}$ for odd primes q.

Proof. We apply the formula given in [Sen97, Thm. 4.18] for hull dimension $h = 0$. For the particular case, observe that it follows from [Sen97, Cor. 4.17] that the constant in $\mathcal{O}\left(\frac{k}{q^{\frac{n}{2}}}\right)$ is positive, hence the factor of $1+\mathcal{O}\left(\frac{k}{q^{\frac{n}{2}}}\right)$ is at least 1 and negligibly close to it. Then we bound

$$\begin{aligned}\prod_{i=1}^{\lceil \frac{k}{2}\rceil}\left(1-\frac{1}{q^{2\cdot i-1}}\right)\cdot\left(1+\mathcal{O}\left(\frac{k}{q^{\frac{n}{2}}}\right)\right) &\geq \prod_{i=1}^{\lceil \frac{k}{2}\rceil}\left(1-\frac{1}{q^{2\cdot i-1}}\right) \geq 1-\sum_{i=1}^{\lceil \frac{k}{2}\rceil}\frac{1}{q^{2\cdot i-1}}\\ &\geq 1-\sum_{i=1}^{\infty}\frac{1}{q^{2\cdot i-1}} = 1-\frac{q}{q^2-1} = \frac{q^2-q-1}{q^2-1} \geq \frac{q^2-q-1}{q^2} = 1-\frac{1}{q}-\frac{1}{q^2},\end{aligned}$$

where we use the Weierstrass product inequality. □

2.1 Indistinguishable Distributions

Two $[n,k]$-linear codes $\mathfrak{C}, \mathfrak{C}'$ over $\mathbb{Z}_q$ are *permutation equivalent*, if there exists a permutation $\pi \in S_n$ such that $\mathfrak{C}' = \{(\mathbf{v}_{\pi^{-1}(1)}, \ldots, \mathbf{v}_{\pi^{-1}(n)}) \; ; \; \mathbf{v} \in \mathfrak{C}\}$. Further, $\mathfrak{C}$ and $\mathfrak{C}'$ are *linearly equivalent*, if there exists a permutation $\pi \in S_n$ and a vector $\mathbf{u} \in (\mathbb{Z}_q^*)^n$ such that $\mathfrak{C}' = \{(\mathbf{u}_1 \cdot \mathbf{v}_{\pi^{-1}(1)}, \ldots, \mathbf{u}_n \cdot \mathbf{v}_{\pi^{-1}(n)}) \; ; \; \mathbf{v} \in \mathfrak{C}\}$. If the entries of $\mathbf{u}$ are limited to ± 1, we say $\mathfrak{C}$ and $\mathfrak{C}'$ are *signed permutation equivalent*.

In terms of generator matrices, two codes generated by generator matrices $\mathbf{G}$ and $\mathbf{H}$, respectively, are permutation equivalent if there exist a permutation matrix $\mathbf{P}$ and a non-singular matrix $\mathbf{U}$ such that $\mathbf{H} = \mathbf{P} \cdot \mathbf{G} \cdot \mathbf{U}$. Similarly, the codes are signed permutation equivalent if there exists a signed permutation matrix $\mathbf{O}$ and a non-singular matrix $\mathbf{U}$ such that $\mathbf{H} = \mathbf{O} \cdot \mathbf{G} \cdot \mathbf{U}$. An $n \times n$ matrix $\mathbf{M}$ over $\mathbb{Z}_q$ is called monomial, if it can be written as $\mathbf{M} = \mathbf{D} \cdot \mathbf{P}$, where $\mathbf{P}$ is a $n \times n$ permutation matrix, and $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ is a diagonal matrix with $d_i \in \mathbb{Z}_q^*$ (and similarly $\mathbf{M} = \mathbf{P} \cdot \mathbf{D}'$ for the same $\mathbf{P}$ and invertible diagonal $\mathbf{D}'$). In other words, $\mathbf{M}$ has exactly one non-zero entry in each row and each column. Then the codes generated by $\mathbf{G}$ and $\mathbf{H}$, respectively, are linearly equivalent if there exists a monomial matrix $\mathbf{M}$ and a non-singular matrix $\mathbf{U}$ such that $\mathbf{H} = \mathbf{M} \cdot \mathbf{G} \cdot \mathbf{U}$. This interpretation gives rise to the following computational problems, adapted from [SS13, Prob. 1] and [DG23, Def. 10], respectively.

Definition 1 (Permutation Code Equivalence). *Let D be a distribution on the subset full-rank matrices in $\mathbb{Z}_q^{n \times k}$ and let $\mathbf{G} \leftarrow\!\!\$\; \mathsf{D}$. Let $\mathbf{H}_0 \leftarrow\!\!\$\; \mathsf{D}$, $\mathbf{H}_1 = \mathbf{P} \cdot \mathbf{G} \cdot \mathbf{U}$ for a $n \times n$ permutation matrix $\mathbf{P}$ and non-singular $\mathbf{U} \in \mathcal{GL}_k(\mathbb{Z}_q)$, both chosen uniformly at random, and let $b \leftarrow\!\!\$\; \{0,1\}$.*

- *The problem $\Delta PCE_{n,k,q}(\mathsf{D})$ is to find b given $(\mathbf{G}, \mathbf{H}_b)$.*
- *The problem $sPCE_{n,k,q}(\mathsf{D})$ is to find $\mathbf{P}$ (or equivalently $\mathbf{U}$) given $(\mathbf{G}, \mathbf{H}_1)$.*

Definition 2 (Signed Permutation Code Equivalence). *Let D be a distribution on the subset full-rank matrices in $\mathbb{Z}_q^{n \times k}$ and let $\mathbf{G} \leftarrow\!\!\$\; \mathsf{D}$. Let $\mathbf{H}_0 \leftarrow\!\!\$\; \mathsf{D}$, $\mathbf{H}_1 = \mathbf{O} \cdot \mathbf{G} \cdot \mathbf{U}$ for a $n \times n$ signed permutation matrix $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$ and non-singular $\mathbf{U} \in \mathcal{GL}_k(\mathbb{Z}_q)$, both chosen uniformly at random, and let $b \leftarrow\!\!\$\; \{0,1\}$.*

- *The problem $\Delta SPCE_{n,k,q}(\mathsf{D})$ is to find b given $(\mathbf{G}, \mathbf{H}_b)$.*
- *The problem $sSPCE_{n,k,q}(\mathsf{D})$ is to find $\mathbf{O}$ (or equivalently $\mathbf{U}$) given $(\mathbf{G}, \mathbf{H}_1)$.*

Remark 1. We formulate the PCE and SPCE problems using a random basis change, despite our construction in Section 5 using the systematic form instead. We note here that the two formulations are equivalent, as any PPT algorithm may compute the systematic form in the former case, or apply a random basis change in the latter case.

In Definitions 1 and 2, if D is the distribution that samples uniformly random $[n,k]$-linear codes over $\mathbb{Z}_q$ with some fixed hull dimension $h \leq k$, we omit D and write $\mathrm{PCE}^h_{n,k,q}$ and $\mathrm{SPCE}^h_{n,k,q}$. If h is big enough, both problems are conjectured to be hard [SS13], which we review in Appendix A. We will also rely on the Learning with Errors assumption.

Definition 3 (Learning With Errors [Reg05]). *For integers $k, q \geq 1$, an error distribution χ over $\mathbb{Z}$, and a distribution D over $\mathbb{Z}_q^k$, the (decision)* LWE *problem is to distinguish, given arbitrarily many samples, the uniform distribution over $\mathbb{Z}_q^k \times \mathbb{Z}_q$ from $\mathcal{A}_{q,k,\chi}(\mathbf{s})$ for $\mathbf{s}$ sampled from D, where $\mathcal{A}_{q,k,\chi}(\mathbf{s})$ samples $\mathbf{a} \leftarrow \mathbb{Z}_q^k$ uniformly and $e \leftarrow \chi$, then outputs $(\mathbf{a}, \mathbf{b} = \langle \mathbf{a}, \mathbf{s} \rangle + e \bmod q)$.*

If the number of samples is limited to $n \in \mathbb{Z}$, the problem is denoted by $\mathrm{LWE}_{k,n,q,\chi}(\mathsf{D})$. If D is uniform over $\mathbb{Z}_q^k$, we omit it.

The following lemma is a slight modification of the binary Leftover Hash Lemma [Reg09, Claim 5.3].

Lemma 2 (Leftover Hash Lemma). *Let D_{real} be the distribution of $(\mathbf{A} \leftarrow\!\!\$\; \mathbb{Z}_q^{n \times k}, \mathbf{u} = \mathbf{A}^\intercal \cdot \mathbf{r}$ for $\mathbf{r} \leftarrow\!\!\$\; \{\pm 1\}^n)$, and let D_{unif} be the distribution of $(\mathbf{A} \leftarrow\!\!\$\; \mathbb{Z}_q^{n \times k}, \mathbf{u} \leftarrow\!\!\$\; \mathbb{Z}_q^k)$. Then*

$$\mathbb{E}[\Delta(\mathsf{D}_{real}, \mathsf{D}_{unif})] \leq \sqrt{q^k / 2^n}.$$

In particular, if $n \geq (1+c) \cdot k \cdot \log_2(q)$ for a positive real constant $c > 0$, then $\Delta(\mathsf{D}_{real}, \mathsf{D}_{unif}) \leq \mathsf{negl}(k)$.

Proof. In the proof of [Reg09, Claim 5.3] simply replace every binary vector $\mathbf{b}$ with the vector $[(-1)^{b_i}]_{b_i \in \mathbf{b}}$. Regarding the particular case, observe

$$\mathbb{E}[\Delta(\mathsf{D}_{\text{real}}, \mathsf{D}_{\text{unif}})] \leq \sqrt{q^{k-(1+c)\cdot k}} = q^{-\frac{c\cdot k}{2}},$$

which is negligible in k. □

2.2 Dual-Regev PKE

We rely on a variant of the Dual-Regev Public-Key Encryption scheme.

Definition 4 (Dual-Regev PKE [GPV08]). *The signed-secret dual-Regev PKE scheme (Figure 1) is a variant of dual-Regev PKE in the sense that the long-term secret vector $\mathbf{r} \in \{\pm 1\}^n$ is a random signed vector.*

$\mathsf{KGen}(1^\lambda)$

$\mathbf{A} \leftarrow\!\$\ \mathbb{Z}_q^{n\times k}$
$\mathbf{r} \leftarrow\!\$\ \{\pm 1\}^n$
$\mathbf{u}^\intercal \coloneqq \mathbf{r}^\intercal \cdot \mathbf{A} \mod q$
return $(\mathsf{pk} = (\mathbf{A}, \mathbf{u}), \mathsf{sk} = \mathbf{r})$

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ctxt})$

return $\left\lfloor \frac{p}{q}(c_1 - \langle \mathbf{r}, \mathbf{c}_0 \rangle \bmod q) \right\rceil$

$\mathsf{Enc}(\mathsf{pk} = (\mathbf{A}, \mathbf{u}), \mathsf{msg} \in \mathbb{Z} \cap [-p/2, p/2))$

$\mathbf{x} \leftarrow\!\$\ \mathbb{Z}_q^k$
$\mathbf{e} \leftarrow\!\$\ \chi^n$
$e' \leftarrow\!\$\ \chi$
$\mathbf{c}_0 \coloneqq \mathbf{A} \cdot \mathbf{x} + \mathbf{e} \mod q$
$c_1 \coloneqq \langle \mathbf{u}, \mathbf{x} \rangle + e' + \left\lfloor \frac{q}{p} \right\rceil \cdot \mathsf{msg} \mod q$
return $\mathsf{ctxt} \coloneqq (\mathbf{c}_0, c_1)$

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

The proofs of correctness and security of the signed-secret dual-Regev PKE follow analogously from those of the original scheme [GPV08] and are omitted. We note that, for proving security, we rely on Lemma 2 instead of the binary Leftover Hash Lemma.

Proposition 1. *Let $k \geq \Omega(\lambda)$ and $\epsilon \leq \mathsf{negl}(\lambda)$. If $\chi = \mathcal{D}_{\mathbb{Z},s}$ is the discrete Gaussian distribution[7] over $\mathbb{Z}$ with parameter $s \geq \sqrt{2}\eta_\epsilon(\mathbb{Z})$ above $\sqrt{2}$ times the smoothing parameter[8] of $\mathbb{Z}$, so that the convolution theorem of [MP13, Theorem 3.3 (eprint)] 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 4 is correct with overwhelming probability in λ. If q is prime and $n \geq (1+c)\cdot k \cdot \log_2(q)$ for some real $c > 0$, and if there exists a PPT algorithm against the IND-CPA security of the signed-secret dual-Regev PKE, then there exists a PPT algorithm against the $\mathrm{LWE}_{k,n,q,\chi}(\mathcal{U}(\{\pm 1\}))$ problem.*

2.3 Updatable Public-Key Encryption

We recall the necessary definitions for updatable public-key encryption, the scheme we are constructing in this work.

Definition 5 (UPKE [DKW21, Def. 3]). *An* updatable (public-key) encryption *(UPKE) scheme for a message space $\mathcal{M}$ is a set of five* PPT *algorithms:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$*: takes as input 1^λ, where λ is the security parameter, and outputs an initial public-private key pair* $(\mathsf{pk}, \mathsf{sk})$.

[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] We refer to [MR04] for the formal definition.

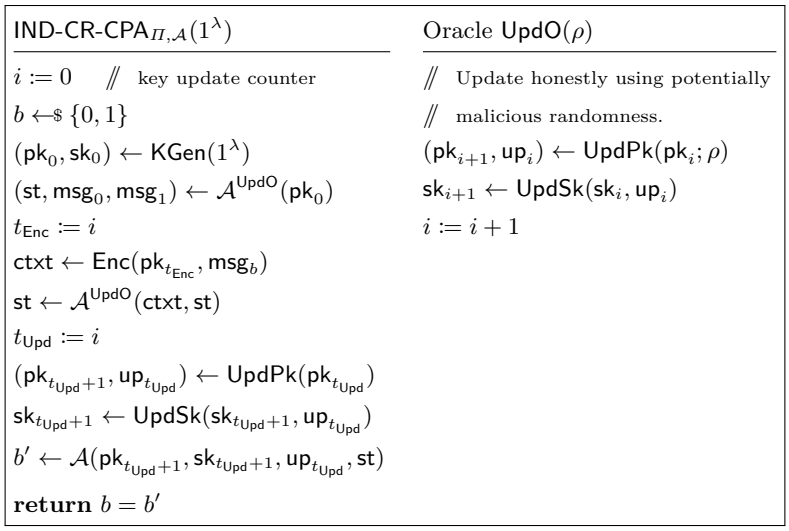

Fig. 2: The IND-CR-CPA game for a UPKE scheme.

- $\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$*: takes as input a public key* pk *and a message* $\mathsf{msg} \in \mathcal{M}$*, and outputs a ciphertext* ctxt*.*
- $\mathsf{msg} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ctxt})$*: takes as input a secret key* sk *and a ciphertext* ctxt*, and outputs a message* msg *or* $\perp$ *on failure.*
- $(\mathsf{pk}', \mathsf{up}) \leftarrow \mathsf{UpdPk}(\mathsf{pk})$*: takes as input a public key* pk*, and outputs a new public key* pk' *and an update token* up*.*
- $\mathsf{sk}' \leftarrow \mathsf{UpdSk}(\mathsf{sk}, \mathsf{up})$*: takes as input a secret key* sk *and an update token* up*, and outputs a new secret key* sk'*.*

Definition 6 (UPKE Correctness [DKW21, p. 264]). *A UPKE scheme* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdPk}, \mathsf{UpdSk})$ *for a message space* $\mathcal{M}$ *is said to be correct if*

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

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

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

Definition 7 (IND-CR-CPA Security of UPKE [DKW21, Def. 4]). *An UPKE scheme admits* indistinguishability under chosen randomness chosen plaintext attacks*, if any efficient (stateful) adversary* $\mathcal{A}$ *has negligible advantage in winning the security game in Figure 2.*

3 How to Sample a Hull

In this section, we provide explicit algorithms for sampling a random self-orthogonal vector of a code and sampling a random linear code of a given hull dimension. We first introduce some notation, then consider the behaviour of self-orthogonal codewords in a given code, and an algorithm for sampling uniformly random self-orthogonal codewords.

Definition 8 (h-hollow). *Let $n \geq k \geq h$ integers and q a prime power. We call a code $\mathfrak{C}$ h-*hollow*, if it has hull dimension h. A matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times k}$ is called h-*hollow *if it is full-rank and generates a $[n,k]$-linear h-hollow code over $\mathbb{Z}_q$.*[9]

Lemma 3. *Let $n \geq k$ and $h \leq k-2$ be integers and q prime. Any h-hollow matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times k}$ has 2 columns $\mathbf{v}, \mathbf{w}$ such that $\langle \mathbf{v}, \mathbf{w} \rangle^2 \neq \langle \mathbf{v}, \mathbf{v} \rangle \cdot \langle \mathbf{w}, \mathbf{w} \rangle$.*

Proof. Denote $\widetilde{\mathfrak{C}} = \mathrm{Span}(\mathbf{A})$. Since the inner product on $\mathbb{Z}_q^n$ is bilinear and any change of basis $\mathbf{U} \in \mathcal{GL}_k(\mathbb{Z}_q)$ is a linear transformation, it is enough to find a basis of $\widetilde{\mathfrak{C}}$ such that the property holds. Since $h \leq k-2$, there exists a subcode $\mathfrak{C}$ of $\widetilde{\mathfrak{C}}$ of dimension $h+2$ with $\mathrm{hull}(\mathfrak{C}) = \mathrm{hull}(\widetilde{\mathfrak{C}})$, and a subcode $\mathfrak{C}'$ of $\mathfrak{C}$ of dimension 2 that has trivial intersection with $\mathrm{hull}(\mathfrak{C})$. Let $\mathbf{v}, \mathbf{w}$ be a basis of $\mathfrak{C}'$. If $\langle \mathbf{v}, \mathbf{v} \rangle = \langle \mathbf{w}, \mathbf{w} \rangle = \langle \mathbf{v}, \mathbf{w} \rangle = 0$, then $\mathfrak{C}' \subseteq \mathrm{hull}(\mathfrak{C})$, a contradiction. If $\langle \mathbf{v}, \mathbf{w} \rangle = 0$, and only one of $\langle \mathbf{v}, \mathbf{v} \rangle$ and $\langle \mathbf{w}, \mathbf{w} \rangle$ is non-zero, again $\mathfrak{C}' \cap \mathrm{hull}(\mathfrak{C}) \neq \{0\}$. If $\langle \mathbf{v}, \mathbf{v} \rangle = \langle \mathbf{w}, \mathbf{w} \rangle = 0$, but $\langle \mathbf{v}, \mathbf{w} \rangle \neq 0$, the condition is satisfied, and likewise if only one of $\langle \mathbf{v}, \mathbf{v} \rangle$ and $\langle \mathbf{w}, \mathbf{w} \rangle$ is 0. The remaining case is that $\langle \mathbf{v}, \mathbf{v} \rangle$ and $\langle \mathbf{w}, \mathbf{w} \rangle$ are both non-zero. Let $\mathbf{w}' = \mathbf{w} - \frac{\langle \mathbf{v}, \mathbf{w} \rangle}{\langle \mathbf{v}, \mathbf{v} \rangle} \cdot \mathbf{v}$. Then $\mathbf{v}, \mathbf{w}'$ also generate $\mathfrak{C}'$, and satisfy the condition, since $\langle \mathbf{w}', \mathbf{w}' \rangle = 0$ would again yield a contradiction. □

The key observation of this section is the following.

Proposition 2. *Given a generator matrix $\mathbf{G}$ for an $[n,k]$-linear code $\mathfrak{C}$ over $\mathbb{Z}_q$, a codeword $\mathbf{x} = \sum_{i=1}^{k} a_i \cdot \mathbf{g}_i$ is self-orthogonal in $\mathfrak{C}$ if and only if $(a_1, \ldots, a_k)$ is a root of the polynomial $\mathbf{y}^{\mathrm{T}} \cdot \mathbf{G}^{\mathrm{T}} \cdot \mathbf{G} \cdot \mathbf{y} \in \mathbb{Z}_q[\mathbf{y}]$ where $\mathbf{y}$ is a vector of k unknowns.*

Definition 9. *Denote by $\mathcal{V}_{q,\mathbf{G}}$ the set of self-orthogonal codewords of a code $\mathfrak{C}$ given by a generator matrix $\mathbf{G}$, and by $\mathcal{U}(\mathcal{V}_{q,\mathbf{G}})$ the uniform distribution on it.*

Lemma 4. *Let $n > k \geq 2$ integers and q prime. Every $[n,k]$-linear code over $\mathbb{Z}_q$ has at least q^{k-2} self-orthogonal codewords.*

Proof. The proof is an application of Warning's Second Theorem [CW35, Satz 3], which states that a polynomial of degree d in m variables over $\mathbb{Z}_q$ that has at least one root must have at least q^{m-d} (distinct) roots. By Proposition 2, the self-orthogonal codewords of a code $\mathfrak{C}$ given by a generator matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times k}$ are precisely the solutions $\mathcal{V}_{q,\mathbf{G}}$ to a homogeneous quadratic equation in k variables over $\mathbb{Z}_q$. Since the equation is homogeneous, it has at least one solution, namely the zero vector is always self-orthogonal, hence it must have at least q^{k-2} solutions since it is degree-2. □

Remark 2. The Ax-Katz Theorem [Kat71], a well-known improvement of the (First) Chevalley-Warning Theorem [CW35] (which states the number of solutions of a system of r polynomials, each of total degree d_i, in m variables over $\mathbb{Z}_q$ is congruent to 0 modulo q if $\sum_{i=1}^{r} d_i < m$), states that the number of solutions to a polynomial system with m variables over $\mathbb{Z}_q$ is congruent to 0 modulo q^{μ} for $\mu = \left\lceil \frac{m - \sum_{i=1}^{r} d_i}{\max_{i=1,\ldots,r}\{d_i\}} \right\rceil$. This would only give us that we have at least $q^{\lceil k/2 \rceil - 1}$ self-orthogonal vectors, despite the stronger statement about the congruence modulo powers of q of the number of self-orthogonal vectors. While this number is still exponential in k, the often neglected Warning's Second Theorem from Chevalley's and Warning's original paper gives us a better lower bound on the number of self-orthogonal vectors.

We describe an algorithm SSO which, given $\mathbf{G}$, samples from $\mathcal{U}(\mathcal{V}_{q,\mathbf{G}})$.

Definition 10 (SSO). *Let n, k, h, q be positive integers with $k < n$, $h \leq \min\{k, n-k\} - 2$, q odd prime, and let $\mathbf{G}$ be a generator matrix of a $[n,k]$-linear h-hollow code $\mathfrak{C}$ over $\mathbb{Z}_q$. Consider the algorithm on the left in Figure 3 that returns a self-orthogonal codeword of $\mathfrak{C}$.*[10]

[9] Note that the term hollow matrix already has multiple meanings in mathematics, e.g. a sparse matrix or a matrix with all 0's on the diagonal.

[10] SSO stands for Sample Self-Orthogonal.

Lemma 5. *Assume the notation of Definition 10, and let* $\mathsf{halt} \in \mathbb{N}$. *The algorithm* SSO *is* PPT *and aborts with negligible probability in* n *if* $\mathsf{halt} \geq n \cdot (q+1)$.[11] *Furthermore, the distribution of its outputs conditioning on not aborting is precisely* $\mathcal{U}(\mathcal{V}_{q,\mathbf{G}})$.

Proof. We discuss efficiency and correctness, failure probability and uniformity of the output, respectively.

Efficiency and correctness. Assume the algorithm SSO returns with a vector $\mathbf{y}$. Then $\mathbf{y}$ is self-orthogonal by Proposition 2. The algorithm is clearly efficient, since it, in the worst case, computes $\mathcal{O}(k^2)$ inner products to reorder $\mathbf{G}$, then samples $\mathsf{halt} \cdot (k-2)$ elements from $\mathbb{Z}_q$, solves halt conic equations over $\mathbb{Z}_q$, samples from a set with at most $2 \cdot q - 1$ elements once, and computes one multiplication of $\mathbf{G}$ with the vector $[c_i]_{i=1}^{k}$. Computing the solutions to the conic equation naïvely requires the computations of the Jacobi symbol, a square root [Coh93, Sec. 1.5.1] and an inverse [Coh93, Sec. 1.6], each q times, together costing $q \cdot \mathcal{O}(\log_2^4(q))$. The worst case cost of running the algorithm is therefore $\mathcal{O}(\mathsf{halt} \cdot k \cdot q \cdot \log_2^4(q) + k^2)$.

Failure probability. We now analyse the failure probability of SSO. Denote by $\mathbf{g}_i$ the columns of $\mathbf{G}$. By the condition on h and Lemma 3, one can always reorder the columns of $\mathbf{G}$ such that $\langle \mathbf{g}_{k-1}, \mathbf{g}_{k-1} \rangle \cdot \langle \mathbf{g}_k, \mathbf{g}_k \rangle \neq \langle \mathbf{g}_{k-1}, \mathbf{g}_k \rangle^2$. We thus have that, for every choice of $c_1, \ldots, c_{k-2}$, the function $F(x,y)$ defines a smooth conic, since the coefficient a in front of x^2 is $\langle \mathbf{g}_{k-1}, \mathbf{g}_{k-1} \rangle$, the coefficient b in front of $x \cdot y$ is $2 \cdot \langle \mathbf{g}_{k-1}, \mathbf{g}_k \rangle$, the coefficient c in front of y^2 is $\langle \mathbf{g}_{k-1}, \mathbf{g}_{k-1} \rangle$, so the discriminant of $F(x,y)$ is

$$D_{\mathbf{G}} = b^2 - 4 \cdot a \cdot c = 4 \cdot \langle \mathbf{g}_{k-1}, \mathbf{g}_k \rangle^2 - 4 \cdot \langle \mathbf{g}_{k-1}, \mathbf{g}_{k-1} \rangle \cdot \langle \mathbf{g}_k, \mathbf{g}_k \rangle \neq 0 \pmod{q},$$

since $q \neq 2$. Recall from Section 2 and Appendix B that if $D_{\mathbf{G}} \in \mathrm{QR}(\mathbb{Z}_q)$, then the number of conic solutions is $S \in \{q-1, 2 \cdot q - 1\}$, and if $D_{\mathbf{G}} \notin \mathrm{QR}(\mathbb{Z}_q)$, then $S \in \{1, q+1\}$. Suppose first $D_{\mathbf{G}} \in \mathrm{QR}(\mathbb{Z}_q)$. The algorithm can only return $\perp$ if for each of the halt repetitions the conic $F(x,y)$ has $q-1$ solutions. Indeed the algorithm must have then sampled $u \leftarrow\!\!\$\, \mathcal{U}([0,1])$ such that $u > \frac{q-1}{2 \cdot q - 1}$ (which is not possible if $S = 2 \cdot q - 1$), which happens with probability $\frac{q}{2 \cdot q - 1}$, thus the failure probability is at most $\left(\frac{q}{2 \cdot q - 1}\right)^{\mathsf{halt}}$. Analogously for $D_{\mathbf{G}} \notin \mathrm{QR}(\mathbb{Z}_q)$, the conic in each repetition must have 1 solution to return $\perp$, and the algorithm must sample $u > \frac{1}{q+1}$, which happens with probability $\frac{q}{q+1}$, making the failure probability at most $\left(\frac{q}{q+1}\right)^{\mathsf{halt}}$. The overall failure probability is thus at most

$$\left(\frac{q}{q+1}\right)^{\mathsf{halt}} = \left(\left(1 - \frac{1}{q+1}\right)^{q+1}\right)^{\frac{\mathsf{halt}}{(q+1)}} \leq e^{\frac{-\mathsf{halt}}{(q+1)}} \leq e^{-n},$$

where the last inequality is due to $\mathsf{halt} \geq n \cdot (q+1)$.

Uniform sampling. To show this algorithm samples from $\mathcal{U}(\mathcal{V}_{q,\mathbf{G}})$, assume again it did not halt with $\perp$ but with a self-orthogonal vector $\mathbf{y} \in \mathfrak{C}$. It remains to be shown that any other self-orthogonal vector in $\mathbf{v} \in \mathfrak{C}$ is just as likely to have been chosen. Since a smooth affine conic always has at least 1 solution, we have that

$$\mathcal{V}_{q,\mathbf{G}} = \left\{\mathbf{v} = [c_1, \ldots, c_{k-2}, x, y]^{\mathsf{T}} \,;\, c_i \in \mathbb{Z}_q, F(x,y) = \mathbf{v}^{\mathsf{T}} \cdot \mathbf{G}^{\mathsf{T}} \cdot \mathbf{G} \cdot \mathbf{v} = 0\right\},$$

so the algorithm samples each vector with non-zero probability. Let $M = \mathsf{MaxSols}(D_{\mathbf{G}})$ be the maximum number of solutions to the obtained conic, which is $q+1$ if $D_{\mathbf{G}} \notin \mathrm{QR}(\mathbb{Z}_q)$ and $2 \cdot q - 1$ if $D_{\mathbf{G}} \in \mathrm{QR}(\mathbb{Z}_q)$. Regardless of the number of solutions S to the obtained conic, the probability of each vector being sampled is

$$\begin{aligned}\Pr[\mathbf{v} \leftarrow\!\!\$\, \mathsf{SSO}(\mathbf{G}, \mathsf{halt}) \mid \mathbf{v} \neq \perp] &= \frac{1}{q^{k-2}} \cdot \frac{1}{S} \cdot \Pr_{u \leftarrow\!\!\$ [0,1]}\left[u \leq \frac{S}{M}\right] \\ &= \frac{1}{q^{k-2}} \cdot \frac{1}{S} \cdot \frac{S}{M} = \frac{1}{q^{k-2}} \cdot \frac{1}{M}.\end{aligned}$$

[11] We remark in Appendix B that the algorithm in practice requires fewer than $n \cdot (q+1)$ repetitions for overwhelming success probability.

Since M only depends on $D_{\mathbf{G}}$, which depends only on $\mathbf{G}$ and is fixed at the beginning of execution, the probability of every self-orthogonal vector being sampled must be the same, that is SSO samples from $\mathcal{U}(\mathcal{V}_{q,\mathbf{G}})$. □

We will use the above algorithm to construct random linear codes that have a desired hull dimension.

Definition 11 ($\mathsf{SampleCode}$)**.** *Let n, k, h, q be positive integers with $2 \cdot k \leq n$, $2 \cdot h \leq k$,[12] and q odd prime. Consider the algorithm on the right in Figure 3 that outputs a generator matrix for a $[n, k]$-linear h-hollow code over $\mathbb{Z}_q$.*

$\mathsf{SSO}(\mathbf{G}, \mathsf{halt})$

$D \coloneqq \langle \mathbf{g}_{k-1}, \mathbf{g}_k \rangle^2 - \langle \mathbf{g}_{k-1}, \mathbf{g}_{k-1} \rangle \cdot \langle \mathbf{g}_k, \mathbf{g}_k \rangle$
if $D = 0$: reorder columns to get ineq.
$M \coloneqq \mathsf{MaxSols}(D)$
$f \coloneqq \mathbf{x}^{\mathrm{T}} \cdot \mathbf{G}^{\mathrm{T}} \cdot \mathbf{G} \cdot \mathbf{x}$
for $\_ \coloneqq 1, \ldots, \mathsf{halt}$ **do**
  for $i \coloneqq 1, \ldots, k-2$ **do** $c_i \leftarrow\!\!\$\ \mathbb{Z}_q$
  $F(x, y) = f(c_1, \ldots, c_{k-2}, x, y)$
  $\mathsf{rs} \coloneqq$ roots of F
  $u \leftarrow\!\!\$\ \mathcal{U}([0, 1])$; **if** $u > \frac{|\mathsf{rs}|}{M}$: **continue**
  $(c_{k-1}, c_k) \leftarrow\!\!\$\ \mathsf{rs}$
  return $\mathbf{y} \coloneqq \sum_{i=1}^{k} c_i \cdot \mathbf{g}_i$
return $\perp$

$\mathsf{SampleCode}(n, k, h, q)$

$\mathbf{A}_{(0)} \leftarrow\!\!\$\ \mathbb{Z}_q^{n \times (k-h)}$
if $\mathrm{rank}(\mathbf{A}_{(0)}) \neq k - h$: **return** $\perp$
if $\dim \mathrm{hull}(\mathbf{A}_{(0)}) \neq 0$: **return** $\perp$
for $i = 1, \ldots, h$ **do**
  $\mathbf{y}_{(i)} \leftarrow \mathsf{SSO}(\mathbf{A}_{(i-1)}^{\perp}, n \cdot (q+1))$
  if $\mathrm{rank}([\mathbf{A}_{(i-1)}, \mathbf{y}_{(i)}]) < k - h + i$:
    return $\perp$
  $\mathbf{A}_{(i)} \coloneqq [\mathbf{A}_{(i-1)}, \mathbf{y}_{(i)}]$
return $\mathrm{SF}(\mathbf{A}_{(h)})$

$\mathsf{MaxSols}(D)$

if $D \in \mathrm{QR}(\mathbb{Z}_q)$: **return** $2 \cdot q - 1$
if $D \notin \mathrm{QR}(\mathbb{Z}_q)$: **return** $q + 1$
return $\perp$

Fig. 3: The SSO and $\mathsf{SampleCode}$ algorithms.

Lemma 6. *Assume the notation of Definition 11. The* $\mathsf{SampleCode}$ *algorithm is* PPT*, it successfully terminates with probability at least $\left(1 - \frac{1}{q} - \frac{1}{q^2}\right) \cdot (1 - \mathsf{negl}(n))$, the output matrix generates a h-hollow code $\mathfrak{C}$, and $\mathfrak{C}$ is uniformly random in the set of all $[n, k]$-linear h-hollow codes over $\mathbb{Z}_q$.*

Proof. We discuss efficiency and correctness, failure probability and uniformity of the output, respectively.

Efficiency and correctness. First assume the algorithm $\mathsf{SampleCode}$ outputs a matrix $\mathbf{A}$. The algorithm outputs a full-rank $n \times k$ matrix over $\mathbb{Z}_q$, since we start with a full-rank $n \times (k-h)$ matrix and append h columns that increment the rank, otherwise we return $\perp$. The starting matrix $\mathbf{A}_{(0)} \in \mathbb{Z}_q^{n \times (k-h)}$ is 0-hollow, otherwise we return $\perp$, and each appended column $\mathbf{y}_{(i)}$ increments the hull dimension. Indeed, $\mathbf{y}_{(i)}$ is by Lemma 5 orthogonal to itself and every vector in the code given by $\mathbf{A}_{i-1}$, since it belongs to the dual code given by $\mathbf{A}_{(i-1)}^{\perp}$. In other words, $\mathbf{y}_{(i)}$ must be in the hull of $\mathbf{A}_{(i)}$. The rank condition ensures $\mathbf{y}_{(i)}$ is not already in the hull of $\mathbf{A}_{(i-1)}$, thus incrementing the hull dimension.

[12] We note that in our experiments, $\mathsf{SampleCode}$ also successfully generates self-dual codes, that is $n = 2 \cdot k$ and $h = k$.

Failure probability. We now analyse the probability that SampleCode returns $\perp$. The initial $\mathbf{A}_{(0)} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times(k-h)}$ will be of full rank $k-h$ with probability

$$\prod_{i=0}^{k-h-1}\left(1-q^{-n+i}\right) \geq 1-\frac{k-h}{q^{n-(k-h-1)}} \geq 1-\frac{\frac{n}{2}}{q^{\frac{n}{2}}}$$

overwhelming in n, where the bound is obtained using the Weierstrass product inequality. The matrix $\mathbf{A}_{(0)}$ now generates a uniformly random $[n, k-h]$-linear code, and using Lemma 1 gives the code generated by $\mathbf{A}_{(0)}$ is 0-hollow with probability at least $1-\frac{1}{q}-\frac{1}{q^2}$.

Next, SSO returns a uniformly random self-orthogonal vector $\mathbf{y}_{(i)}$ in $\mathbf{A}_{(i-1)}^{\perp}$ except with negligible probability in n by Lemma 5, since $\mathsf{halt} = n\cdot(q+1)$. Denoting $\gamma = \frac{q+1}{q}$ and $m = n\cdot(q+1)$, the sampler succeeds with probability at least $1-\frac{1}{\gamma^m}$ each of the h times. If $\mathbf{y}_{(i)}$ is already in $\mathrm{Span}(\mathbf{A}_{(i-1)})$ then it must by construction be in $\mathrm{hull}(\mathbf{A}_{(i-1)})$. Hence for $i = 1,\ldots,h$ in the algorithm, the matrix $\mathbf{A}_{(i-1)}$ generates a $[n, k-h+i-1]$-linear $(i-1)$-hollow code over $\mathbb{Z}_q$, and by Lemma 4 the probability we have sampled a vector already in the current hull is bounded by

$$\frac{q^{i-1}}{|\mathcal{V}_{q,\mathbf{A}_{(i-1)}^{\perp}}|} \leq \frac{q^{i-1}}{q^{n-(k-h+i-1)-2}} = q^{2\,i-(n-(k-h))} \leq q^{h-(n-k)}.$$

The probability that this does not happen in any iteration of the loop is therefore at least

$$\left(1-q^{h-(n-k)}\right)^h \geq 1-\frac{h}{q^{n-k-h}} \geq 1-\frac{\frac{n}{4}}{q^{\frac{n}{4}}},$$

since we need to be successful all h times, otherwise we return $\perp$. The bound is again obtained using the Weierstrass product inequality. Thus, the success probability that the algorithm returns a code generator matrix is at least

$$\left(1-\frac{1}{q}-\frac{1}{q^2}\right)\cdot\left(1-\frac{\frac{n}{2}}{q^{\frac{n}{2}}}\right)\cdot\left(1-\frac{\frac{n}{4}}{q^{\frac{n}{4}}}\right)\cdot\left(1-\frac{h}{\gamma^m}\right) = \left(1-\frac{1}{q}-\frac{1}{q^2}\right)\cdot(1-\mathsf{negl}(n)).$$

Uniform sampling. Assuming SampleCode outputs a matrix $\mathbf{A}_{(h)}$, observe that SampleCode first samples a uniformly random $[n, k-h]$-linear code generator matrix $\mathbf{A}_{(0)}$ with a trivial hull, and then samples a uniformly random self-orthogonal $[n, h]$-subcode generator matrix $\mathbf{Y}$ of the code given by $\mathbf{A}_{(0)}^{\perp}$, after which it returns $\mathrm{SF}([\mathbf{A}_{(0)}, \mathbf{Y}])$.

Consider instead an algorithm SampleUnifCode which first samples a uniformly random self-orthogonal $[n, h]$-linear code generator matrix $\mathbf{Y}_{(U)}$ and then samples a uniformly random $[n, k-h]$-subcode $\mathbf{A}_{(U)}$ of the code given by $\mathbf{Y}_{(U)}^{\perp}$ such that $\dim\mathrm{hull}(\mathbf{A}_{(U)}) = 0$ and $\mathrm{Span}(\mathbf{A}_{(U)}) \cap \mathrm{Span}(\mathbf{Y}_{(U)}) = \{0\}$. Recall that $\mathrm{Span}(\mathbf{Y}_{(U)}) \subseteq \mathrm{Span}(\mathbf{Y}_{(U)}^{\perp})$ since $\mathbf{Y}_{(U)}$ is self-orthogonal. Finally, SampleUnifCode outputs $\mathrm{SF}([\mathbf{A}_{(U)}, \mathbf{Y}_{(U)}])$. We have that SampleUnifCode samples uniformly random h-hollow codes, since, for any two codes $\mathfrak{C}_1, \mathfrak{C}_2$ with $\dim\mathrm{hull}(\mathfrak{C}_i) = h$, the probability of $\mathbf{Y}_{(U)}$ spanning $\mathrm{hull}(\mathfrak{C}_1)$ is the same as it spanning $\mathrm{hull}(\mathfrak{C}_2)$. Then the sampled $\mathbf{A}_{(U)}$ is subject only to linear constraints given by $\langle \mathbf{y}_i, \mathbf{a}_j\rangle = 0$ for all $i = 1,\ldots,h$ and $j = 1,\ldots,k-h$.

We now show that SampleCode and SampleUnifCode sample from the same distribution. Suppose $\mathbf{A}_{(U)}, \mathbf{Y}_{(U)}$ are as in SampleUnifCode, and consider a run of SampleCode where we fix $\mathbf{A}_{(0)} \coloneqq \mathbf{A}_{(U)}$, i.e. that output by SampleUnifCode. The algorithm SampleCode then samples $\mathbf{Y}$, generating a uniformly random self-orthogonal subcode of $\mathbf{A}_{(0)}^{\perp}$. We have $\mathrm{Span}(\mathbf{A}_{(0)}) \subseteq \mathrm{Span}(\mathbf{Y}_{(U)}^{\perp})$ and $\mathrm{Span}(\mathbf{A}_{(0)}) \cap \mathrm{Span}(\mathbf{Y}_{(U)}) = \{0\}$. Moreover, $\mathrm{Span}(\mathbf{Y}) \subseteq \mathrm{Span}(\mathbf{A}_{(0)}^{\perp})$ and also $\mathrm{Span}(\mathbf{A}_{(0)}) \cap \mathrm{Span}(\mathbf{Y}) = \{0\}$, i.e. both $\mathbf{Y}_{(U)}$ and $\mathbf{Y}$ span a uniformly random self-orthogonal $[n, h]$-subcode of the code given by $\mathbf{A}_{(0)}^{\perp}$.

Conversely, let $\mathbf{A}_{(0)}, \mathbf{Y}$ as in SampleCode, and consider a run of the algorithm SampleUnifCode where we fix $\mathbf{Y}_{(U)} \coloneqq \mathbf{Y}$, i.e. to that output by SampleCode. The algorithm SampleUnifCode then samples $\mathbf{A}$ generating a uniformly random subcode of $\mathbf{Y}^{\perp}$ with $\mathrm{Span}(\mathbf{A}) \cap \mathrm{Span}(\mathbf{Y}) = \{0\}$. But by construction also

$\mathrm{Span}(\mathbf{A}_{(0)}) \subseteq \mathrm{Span}(\mathbf{Y}^{\perp})$ and $\mathrm{Span}(\mathbf{A}_{(0)}) \cap \mathrm{Span}(\mathbf{Y}) = \{0\}$, i.e. both $\mathbf{A}_{(0)}$ and $\mathbf{A}$ span a uniformly random $[n, k-h]$-subcode of the code given by $\mathbf{Y}^{\perp}_{(U)}$ that has trivial intersection with $\mathbf{Y}_{(U)}$ and a trivial hull.

Summarising, let $\mathsf{D}_1^{\mathrm{rest}}, \mathsf{D}_1^{\mathrm{hull}}$ be distributions of $\mathbf{A}_{(0)}$ and $\mathbf{Y}$ in $\mathsf{SampleCode}$, and $\mathsf{D}_2^{\mathrm{rest}}, \mathsf{D}_2^{\mathrm{hull}}$ the distributions of $\mathbf{A}_{(U)}, \mathbf{Y}_{(U)}$ in $\mathsf{SampleUnifCode}$. We have shown that

$$\Pr\left[\mathbf{Y}_{(U)} \leftarrow\!\$\ \mathsf{D}_2^{\mathrm{hull}} \;\middle|\; \mathbf{A}_{(U)} \leftarrow\!\$\ \mathsf{D}_2^{\mathrm{rest}}\right] = \Pr\left[\mathbf{Y} \leftarrow\!\$\ \mathsf{D}_1^{\mathrm{hull}} \;\middle|\; \mathbf{A}_{(U)} \leftarrow\!\$\ \mathsf{D}_2^{\mathrm{rest}}\right] \text{ and}$$
$$\Pr\left[\mathbf{A}_{(0)} \leftarrow\!\$\ \mathsf{D}_1^{\mathrm{rest}} \;\middle|\; \mathbf{Y} \leftarrow\!\$\ \mathsf{D}_1^{\mathrm{hull}}\right] = \Pr\left[\mathbf{A}_{(U)} \leftarrow\!\$\ \mathsf{D}_2^{\mathrm{rest}} \;\middle|\; \mathbf{Y} \leftarrow\!\$\ \mathsf{D}_1^{\mathrm{hull}}\right].$$

By definition of conditional probability this implies

$$\Pr\left[\mathbf{Y}_{(U)} \leftarrow\!\$\ \mathsf{D}_2^{\mathrm{hull}} \wedge \mathbf{A}_{(U)} \leftarrow\!\$\ \mathsf{D}_2^{\mathrm{rest}}\right] = \Pr\left[\mathbf{Y} \leftarrow\!\$\ \mathsf{D}_1^{\mathrm{hull}} \wedge \mathbf{A}_{(U)} \leftarrow\!\$\ \mathsf{D}_2^{\mathrm{rest}}\right] \text{ and}$$
$$\Pr\left[\mathbf{A}_{(0)} \leftarrow\!\$\ \mathsf{D}_1^{\mathrm{rest}} \wedge \mathbf{Y} \leftarrow\!\$\ \mathsf{D}_1^{\mathrm{hull}}\right] = \Pr\left[\mathbf{A}_{(U)} \leftarrow\!\$\ \mathsf{D}_2^{\mathrm{rest}} \wedge \mathbf{Y} \leftarrow\!\$\ \mathsf{D}_1^{\mathrm{hull}}\right]. \qquad \square$$

Corollary 1. *If $\mathbf{A} \in \mathbb{Z}_q^{n\times k}$ generates a uniformly random $[n,k]$-linear h-hollow code over $\mathbb{Z}_q$ and $\mathbf{y} \leftarrow\!\$\ \mathcal{U}\left(\mathcal{V}_{q,\mathbf{A}^{\perp}}\right)$, then $[\mathbf{A}, \mathbf{y}]$ generates a uniformly random $[n, k+1]$-linear $(h+1)$-hollow code over $\mathbb{Z}_q$.*

Proof. Follows from the last part of the proof of Lemma 6. $\square$

4 Hollow Lattice Problems

We show that LWE with the promise that the instance matrices have a specified hull dimension h is as hard as standard LWE. We also show that a Leftover Hash Lemma variant applies to tall enough matrices with specific hull dimensions.

We begin with our LWE reduction. Roughly, the reduction works by translating an LWE sample $(\mathbf{A}, \mathbf{b})$, where $\mathbf{A}$ has dimension $n \times (k-h)$, into a new LWE sample $(\mathbf{A}', \mathbf{b}')$, where $\mathbf{A}'$ has dimension $n \times k$ and the $[n,k]$-linear code generated by the columns of $\mathbf{A}'$ is h-hollow. We consider first the special case of $h = 1$ to illustrate the reduction, then extend it with induction to a general h.

Definition 12 (Hollow LWE). *Assume the notation of Definitions 3 and 8. The hollow* LWE *problem with n samples, prime power q, and hull dimension $h \geq 0$ is denoted by $\mathrm{LWE}^{h}_{k,n,q,\chi}(\mathsf{D})$, and is the problem $\mathrm{LWE}_{k,n,q,\chi}(\mathsf{D})$ with the promise that the sample matrix $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]^{\mathsf{T}}$ is h-hollow.*

Lemma 7 ($\mathrm{LWE}_{k,n,q,\chi} \to \mathrm{LWE}^{0}_{k,n,q,\chi}$). *Assume the notation of Definition 12 and q prime. If there exists a (t,ε)-algorithm $\mathcal{A}$ for $\mathrm{LWE}^{0}_{k,n,q,\chi}$, then there exists a (t,ε')-algorithm $\mathcal{B}$ for $\mathrm{LWE}_{k,n,q,\chi}$ where*

$$\varepsilon' \geq \varepsilon \cdot \left(1 - \frac{1}{q} - \frac{1}{q^2}\right) \cdot \prod_{i=0}^{k-1} \left(1 - q^{i-n}\right).$$

Proof. The algorithm $\mathcal{B}$ just checks that the LWE instance matrix $\mathbf{A}$ is full-rank and has trivial hull, and forwards the instance to $\mathcal{A}$ if both checks pass. The matrix $\mathbf{A}$ is full-rank with probability $\prod_{i=0}^{k-1}\left(1-q^{i-n}\right)$, and it has trivial hull (provided it is full rank) with probability at least $1 - \frac{1}{q} - \frac{1}{q^2}$ by Lemma 1. $\square$

Lemma 8 ($\mathrm{LWE}^{0}_{k-1,n,q,\chi} \to \mathrm{LWE}^{1}_{k,n,q,\chi}$). *Assume the notation of Definition 12 and q odd prime. If there exists a (t,ε)-algorithm $\mathcal{A}$ for $\mathrm{LWE}^{1}_{k,n,q,\chi}$, then there exists a $(t + \mathsf{poly}(\lambda), \varepsilon')$-algorithm $\mathcal{B}$ for $\mathrm{LWE}_{k-1,n,q,\chi}$ where $\gamma = \frac{q+1}{q}$ and*

$$\epsilon' \geq \varepsilon \cdot \left(1 - \gamma^{-n\cdot(q+1)}\right) \cdot \left(1 - q^{k+1-n}\right).$$

Proof. The algorithm $\mathcal{B}$ receives an LWE instance $(\mathbf{A}, \mathbf{b})$ where $\mathbf{A} \in \mathbb{Z}_q^{n\times(k-1)}$ and $\mathbf{b} \leftarrow \mathbb{Z}_q^n$. The matrix $\mathbf{A}$ is 0-hollow by assumption. It computes the dual $[n, n-k+1]$-linear code $\mathbf{A}^{\perp}$ and samples a self-orthogonal vector $\mathbf{y} \leftarrow\!\$\ \mathcal{U}\left(\mathcal{V}_{q,\mathbf{A}^{\perp}}\right)$ using the algorithm from Definition 10 on $\mathbf{A}^{\perp}$ and $\mathsf{halt} = n \cdot (q+1)$. If $\mathbf{y}$ is $\perp$ or

linearly depends on the columns of $\mathbf{A}$, then $\mathcal{B}$ aborts. It also samples uniformly at random $s_0 \leftarrow\$ \mathbb{Z}_q$ and an invertible matrix $\mathbf{U} \leftarrow\$ \mathcal{GL}_k(\mathbb{Z}_q)$, and computes

$$\mathbf{A}' \coloneqq [\mathbf{A} \mid \mathbf{y}] \cdot \mathbf{U}, \qquad \mathbf{b}' \coloneqq \mathbf{b} + s_0 \cdot \mathbf{y}.$$

Lastly $\mathcal{B}$ calls $\mathcal{A}$ on $(\mathbf{A}', \mathbf{b}')$ and returns its output bit.

Suppose $\mathcal{B}$ does not abort. If $(\mathbf{A}, \mathbf{b})$ was sampled from $\mathcal{A}_{q,k-1,\chi}(\mathbf{s})$ with $\mathbf{A}$ 0-hollow for some uniformly random $\mathbf{s} \in \mathbb{Z}_q^{k-1}$, then $\mathbf{b}'$ is a real LWE sample for the secret $\mathbf{s}' = \mathbf{U}^{-1} \cdot [\mathbf{s}^\intercal, s_0]^\intercal$, which is distributed uniformly because s_0 was uniform and $\mathbf{U}$ is a uniformly random bijection on $\mathbb{Z}_q^k$. Indeed, for each row $\mathbf{A}'_i$ of $\mathbf{A}'$ we have that

$$\mathbf{A}'_i \cdot \mathbf{s}' = [\mathbf{A}_i \mid \mathbf{y}_i] \cdot \mathbf{U} \cdot \mathbf{U}^{-1} \cdot [\mathbf{s}^\intercal, s_0]^\intercal = \mathbf{A}_i \cdot \mathbf{s} + s_0 \cdot y_i = b_i + s_0 \cdot y_i.$$

If however $\mathbf{b}$ was sampled from the uniform distribution on $\mathbb{Z}_q^n$, then $\mathbf{b}'$ is also distributed according to the uniform distribution in $\mathbb{Z}_q^n$, since the map $\mathbf{x} \mapsto \mathbf{x} + s_0 \cdot \mathbf{y} \pmod q$ is a bijection on $\mathbb{Z}_q^n$. In both cases $\mathbf{A}'$ is a uniformly random 1-hollow matrix by Corollary 1. The algorithm $\mathcal{B}$ is therefore correct with the same probability ε as $\mathcal{A}$, and it is clearly efficient by Lemma 5.

We now evaluate the probability that $\mathcal{B}$ does not abort. Firstly, $\mathbf{A}$ is 0-hollow, in particular it is full-rank. Hence $\mathbf{A}$ generates an $[n, k-1]$-linear code over $\mathbb{Z}_q$ and has a dual $[n, n-k+1]$-linear code generated by $\mathbf{A}^\perp$. The algorithm from Definition 10 samples $\mathbf{y} \leftarrow\$ \mathcal{U}\left(\mathcal{V}_{q,\mathbf{A}^\perp}\right)$ with probability at least $1 - \frac{1}{\gamma^{n\cdot(q+1)}}$ which is negligible in n by Lemma 5, and $\mathbf{y}$ is linearly independent from the columns of $\mathbf{A}$ with probability at least $1 - q^{k+1-n}$ by Lemma 4. Thus ε' is as claimed. □

Corollary 2 ($\mathrm{LWE}^{i-1}_{k-1,n,q,\chi} \to \mathrm{LWE}^{i}_{k,n,q,\chi}$)**.** *Assume the notation of Definition 12 and q odd prime. If there exists a (t,ε)-algorithm $\mathcal{A}$ for $\mathrm{LWE}^{i}_{k,n,q,\chi}$, then there exists a $(t + \mathsf{poly}(\lambda), \varepsilon')$-algorithm $\mathcal{B}$ for $\mathrm{LWE}^{i-1}_{k-1,n,q,\chi}$ where $\gamma = \frac{q+1}{q}$ and*

$$\epsilon' \geq \varepsilon \cdot \left(1 - \gamma^{-n}\right) \cdot \left(1 - q^{k+i-n}\right).$$

Proof. The reduction proceeds exactly the same as the reduction in Lemma 8, but the probability that $\mathcal{B}$ aborts differs slightly. Concretely, by Lemma 4 the probability of sampling a vector which is linearly independent on the vectors forming the hull is at least $1 - q^{k+i-n}$. Again by Corollary 1 the output matrix $\mathbf{A}'$ is a uniformly random i-hollow matrix. □

Theorem 1. *Assume the notation of Definition 12 and q odd prime. If there exists a (t,ε)-algorithm $\mathcal{A}$ for $\mathrm{LWE}^{h}_{k,n,q,\chi}$ then there exists a $(t + \mathsf{poly}(\lambda), \varepsilon')$-algorithm $\mathcal{B}$ for $\mathrm{LWE}_{k-h,n,q,\chi}$ where $\gamma = \frac{q+1}{q}$ and*

$$\epsilon' \geq \varepsilon \cdot \left(1 - \frac{1}{q} - \frac{1}{q^2}\right) \cdot \left(1 - \frac{h}{\gamma^{n\cdot(q+1)}}\right) \cdot \prod_{i=0}^{k-h} \left(1 - q^{i-n}\right) \cdot \prod_{i=1}^{h} \left(1 - q^{k+i-n}\right).$$

Proof. The proof is a conjunction of the checks in Lemma 7, and using Lemma 8 once and Corollary 2 $(h-1)$ times. Namely, the algorithm $\mathcal{B}$, given $(\mathbf{A}, \mathbf{b})$, first runs the reduction in Lemma 7 to check if $\mathbf{A}$ is 0-hollow, and then follows Lemma 8 to produce $(\mathbf{A}', \mathbf{b})$ with $\mathbf{A}'$ being h-hollow. We note that this procedure, if successful, produces a full-rank $\mathbf{A}'$. Then it applies the procedure of Corollary 2 $(h-1)$ times, again incrementing the rank and hull dimension each time. □

We now proceed to prove that we may still apply the Leftover Hash Lemma even if the matrix whose rows we sum has a guaranteed hull dimension. Our reduction works by noticing that a sufficient number of entries per column are uniformly random regardless of hull, allowing us to appeal to the standard LHL.

Lemma 9 (Hollow LHL). *Let $n \geq (1+c)\cdot k \cdot \log_2(q) + k + h$ for a positive real constant $c > 0$, let $h \leq \frac{k}{2}$, and q an odd prime. Let $\mathbf{A} \in \mathbb{Z}_q^{n\times k}$ be a uniformly random h-hollow matrix, $\mathbf{r} \leftarrow\$ \{\pm 1\}^n$ uniformly at random, and $\mathbf{u} \leftarrow\$ \mathbb{Z}_q^k$ uniformly at random. Then the pairs $(\mathbf{A}, \mathbf{r}^\intercal \cdot \mathbf{A})$ and $(\mathbf{A}, \mathbf{u}^\intercal)$ are statistically close in k.*

Proof. For any $\mathbf{A}$, we can decompose it as $\mathbf{A} = [\mathbf{A}_{(0)}, \mathbf{Y}] \cdot \mathbf{U}$ where $\mathbf{Y}$ generates hull($\mathbf{A}$), $\mathbf{A}_{(0)}$ generates the rest of Span($\mathbf{A}$), and $\mathbf{U}$ is the corresponding basis change. Without loss of generality, we may assume that $\mathbf{A}$ is given in the form $\mathbf{A} = [\mathbf{A}_{(0)}, \mathbf{Y}]$, since multiplication by $\mathbf{U}$ is bijective and preserves the uniform distribution. Denote by $\mathbf{a}_i$ for $i = 1, \ldots, k-h$ the columns of $\mathbf{A}_{(0)}$, and by $\mathbf{y}_i$ for $i = 1, \ldots, h$ the columns of $\mathbf{Y}$. Observe that

$$\mathbf{r}^\intercal \cdot \mathbf{A} = [\langle \mathbf{r}, \mathbf{a}_1 \rangle, \ldots, \langle \mathbf{r}, \mathbf{a}_{k-h} \rangle, \langle \mathbf{r}, \mathbf{y}_1 \rangle, \ldots, \langle \mathbf{r}, \mathbf{y}_h \rangle].$$

We proceed by induction. Denote $\mathbf{y}_1 = \mathbf{G} \cdot \mathbf{x}$ for $\mathbf{G} \in \mathbb{Z}_q^{n \times (n-k+h)}$ the systematic form generator matrix of $\mathbf{A}_{(0)}^\perp$. We claim that all but two entries of $\mathbf{x}$ can be chosen uniformly at random. Indeed, let $\mathbf{g}_i, \mathbf{g}_j$ be any two columns of $\mathbf{G}$ satisfying the property of Lemma 3. Sample $\mathbf{x}[\ell]$ for $\ell \neq i, j$ uniformly from $\mathbb{Z}_q$, and set $\mathbf{x}[i] = v$ and $\mathbf{x}[j] = w$ for v, w variables over $\mathbb{Z}_q$. Define the function

$$F(v, w) = \mathbf{x}^\intercal \cdot \mathbf{G}^\intercal \cdot \mathbf{G} \cdot \mathbf{x}.$$

By Lemma 3, $F(v, w) = 0$ defines a smooth conic, since the coefficient a in front of v^2 is $\langle \mathbf{g}_i, \mathbf{g}_i \rangle$, the coefficient b in front of $v \cdot w$ is $2 \cdot \langle \mathbf{g}_i, \mathbf{g}_j \rangle$, the coefficient c in front of w^2 is $\langle \mathbf{g}_j, \mathbf{g}_j \rangle$, so the discriminant of $F(v, w)$ is

$$b^2 - 4 \cdot a \cdot c = 4 \cdot \langle \mathbf{g}_i, \mathbf{g}_j \rangle^2 - 4 \cdot \langle \mathbf{g}_i, \mathbf{g}_i \rangle \cdot \langle \mathbf{g}_j, \mathbf{g}_j \rangle \neq 0 \pmod{q},$$

since $q \neq 2$. The smooth affine conic $F(v, w)$ admits a solution, that is we may find $v_0, w_0 \in \mathbb{Z}_q$ such that setting $\mathbf{x}[i] = v_0$ and $\mathbf{x}[j] = w_0$ yields a self-orthogonal vector $\mathbf{y} = \mathbf{G} \cdot \mathbf{x}$, hence in the hull of $\mathbf{A}$. Note however that since $\mathbf{G}$ is in systematic form, all entries of $\mathbf{x}$ appear as entries in $\mathbf{y}_1$, wlog. as the first $n - k + h$ entries. Let $\mathcal{I}_1$ be the set of the first $n - k + h$ indices of $\mathbf{y}_1$ without i and j. Then since

$$n - k + h - 2 \geq (1 + c) \cdot (k - h) \cdot \log_2(q),$$

we have by Lemma 2 that $\langle \mathbf{r}[\mathcal{I}_1], \mathbf{y}_1[\mathcal{I}_1] \rangle$ is statistically close to uniform.

Further, take $i > 1$, assume $\langle \mathbf{r}[\mathcal{I}_j], \mathbf{y}_j[\mathcal{I}_j] \rangle$ is statistically close to uniform for $j < i$, and denote $\mathbf{y}_i = \mathbf{G} \cdot \mathbf{x}$ for $\mathbf{G}$ the systematic form generator matrix of $[\mathbf{A}_{(0)}, \mathbf{y}_1, \ldots, \mathbf{y}_{i-1}]^\perp$. Observe that $[\mathbf{A}_{(0)}, \mathbf{y}_1, \ldots, \mathbf{y}_{i-1}]$ is $(i-1)$-hollow, and thus so is its dual. Hence by Lemma 3 there exists a pair of columns $\mathbf{g}_{i'}, \mathbf{g}_{j'}$ satisfying the property of Lemma 3. As above, we define a smooth conic and obtain its root to construct $\mathbf{y}_i$ whose first $n - k + h$ entries are uniformly random, except the entries at indices i' and j', which are determined by the conic. Let $\mathcal{I}_i$ bet the set of the first $n - k + h$ indices of $\mathbf{y}_i$ without i' and j'. We have by Lemma 2 that $\langle \mathbf{r}[\mathcal{I}_i], \mathbf{y}_i[\mathcal{I}_i] \rangle$ is statistically close to uniform. By the same argument as in the proof of Lemma 6, the vectors $\mathbf{y}_1, \ldots, \mathbf{y}_h$ are linearly independent, and thus span hull($\mathbf{A}$).

Let $\mathcal{I} = \bigcap_{i=1}^h \mathcal{I}_i$ be the set of at least $n - k + h - 2 \cdot h = n - k - h$ indices. We have $\langle \mathbf{r}[\mathcal{I}], \mathbf{Y}[\mathcal{I}] \rangle$ is statistically close to uniform. Since in particular also $n \geq (1 + c) \cdot (k - h) \cdot \log_2(q) + k + h$, the product $\langle \mathbf{r}[\mathcal{I}], \mathbf{A}[\mathcal{I}] \rangle$ is statistically close to uniform by Lemma 2, and hence $\langle \mathbf{r}, \mathbf{A} \rangle$ is statistically close to uniform since

$$n - k - h \geq (1 + c) \cdot k \cdot \log_2(q).$$

□

5 Updatable Public-Key Encryption

We define our updatable public-key encryption scheme that relies on PCE to update dual-Regev keys with orthogonal matrices in $\mathcal{O}_n(\mathbb{Z})$, that is the automorphisms of $\mathbb{Z}^n$ which are precisely signed permutations. We first consider the standard definitions given in [DKW21], then present our scheme as a dual-Regev PKE scheme endowed with an update mechanism, and prove it secure in the Random Oracle Model. Finally, we select example parameters.

Definition 13. *Let $h, k, n, p, q \in \mathbb{N}$, with $h \leq k \leq n$, $p < q$ and q prime, distribution χ over $\mathbb{Z}$, and hash functions family $\mathcal{H}$ from $\{0,1\}^*$ to $\mathcal{O}_n(\mathbb{Z})$, be parametrised by λ. We construct an updatable public-key encryption scheme for the message space $\mathcal{M} = \mathbb{Z} \cap [-p/2, p/2)$ in Figure 4.*

$\mathsf{KGen}(1^\lambda)$

// try sampling at most $\log_2(q-1)$ times
$\mathbf{A} \leftarrow \mathsf{SampleCode}(n, k, h, q)$
$\mathbf{r} \leftarrow\!\!\$\ \{\pm 1\}^n$
$\mathbf{u}^\intercal \coloneqq \mathbf{r}^\intercal \cdot \mathbf{A} \bmod q$
$\mathsf{H} \leftarrow\!\!\$\ \mathcal{H}$
$\mathsf{pk} \coloneqq (\mathbf{A}, \mathbf{u}, \mathsf{H})$
$\mathsf{sk} \coloneqq \mathbf{r}$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ctxt})$

return $\left\lfloor \frac{p}{q} \cdot (c_1 - \langle \mathbf{r}, \mathbf{c}_0 \rangle \bmod q) \right\rceil$

$\mathsf{UpdSk}(\mathsf{sk}, \mathsf{up})$

// decrypt $\lfloor \log_2(p) \rfloor$ bits at a time
$\rho \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{up})$
$\mathbf{O} \coloneqq \mathsf{H}(\rho)$
$\mathbf{r}' \coloneqq \mathbf{O} \cdot \mathbf{r}$
return $\mathsf{sk}' \coloneqq \mathbf{r}'$

$\mathsf{Enc}(\mathsf{pk}, \mathsf{msg} \in \mathbb{Z} \cap [-p/2, p/2))$

$\mathbf{x} \leftarrow\!\!\$\ \mathbb{Z}_q^k$
$\mathbf{e} \leftarrow\!\!\$\ \chi^n$
$e' \leftarrow\!\!\$\ \chi$
$\mathbf{c}_0 \coloneqq \mathbf{A} \cdot \mathbf{x} + \mathbf{e} \bmod q$
$c_1 \coloneqq \langle \mathbf{u}, \mathbf{x} \rangle + e' + \left\lfloor \frac{q}{p} \right\rceil \cdot \mathsf{msg} \bmod q$
return $\mathsf{ctxt} \coloneqq (\mathbf{c}_0, c_1)$

$\mathsf{UpdPk}(\mathsf{pk})$

$\rho \leftarrow\!\!\$\ \{0,1\}^\lambda$
$\mathbf{O} \coloneqq \mathsf{H}(\rho)$
$(\mathbf{A}', \mathbf{U}) \coloneqq \mathrm{SF}(\mathbf{O} \cdot \mathbf{A})$
$\mathbf{u}'^\intercal \coloneqq \mathbf{u}^\intercal \cdot \mathbf{U} \bmod q$
$\mathsf{pk}' \coloneqq (\mathbf{A}', \mathbf{u}')$
// encrypt $\lfloor \log_2(p) \rfloor$ bits at a time
$\mathsf{up} \leftarrow \mathsf{Enc}(\mathsf{pk}, \rho)$
return $(\mathsf{pk}', \mathsf{up})$

Fig. 4: Construction of updatable encryption.

Theorem 2. *If p, q, s, χ is such that $\left\|\chi^{n+1}\right\| \leq s\sqrt{n+1}$ with overwhelming probability and $q > \frac{p^2}{2} + 2 \cdot p \cdot (n+1) \cdot s$, then the UPKE in Figure 4 is correct.*

Proof. By Lemma 6, $\mathsf{SampleCode}$ fails with probability negligibly close to at most $\frac{1}{q} + \frac{1}{q^2} \leq \frac{1}{q-1}$. Trying sampling $\log_2(q-1)$ times guarantees success of $\mathsf{KGen}(1^\lambda)$ except with probability at most $2^{-\lambda}$. The correctness of decryption follows immediately from Proposition 1. To verify correctness after updating, observe that if $((\mathbf{A}, \mathbf{u}), \mathbf{r}) \leftarrow \mathsf{KGen}(1^\lambda)$, we have for any $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$ and $\mathbf{U}$ the systematic form transform of $\mathbf{O} \cdot \mathbf{A}$, that

$$(\mathbf{O} \cdot \mathbf{r})^\intercal \cdot (\mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U}) = \mathbf{r} \cdot \mathbf{A} \cdot \mathbf{U} = \mathbf{u}^\intercal \cdot \mathbf{U}.$$

so the updated key-pair $((\mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U}, \mathbf{U}^\intercal \cdot \mathbf{u}), \mathbf{O} \cdot \mathbf{r})$ is valid. □

5.1 Security Proof

We prove the security of our construction from Definition 13 in the Random Oracle Model assuming the Learning with Errors problem and the Permutation Code Equivalence problem for h-hollow codes are hard.

Lemma 10. *Assume the notation of Figure 4. For any key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$, any orthogonal matrix $\mathbf{O} \in \mathcal{O}_n(\mathbb{Z})$ used to update $(\mathsf{pk}, \mathsf{sk})$ to $(\mathsf{pk}^*, \mathsf{sk}^*)$, and any message $\mathsf{msg} \in \mathcal{M}$, the following holds:*

(a) If $(\mathbf{c}_0, c_1) \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$, then $\mathbf{c}^* = (\mathbf{O} \cdot \mathbf{c}_0, c_1)$ is distributed according to $\mathsf{Enc}(\mathsf{pk}^*, \mathsf{msg})$.
(b) Conversely, if $\mathbf{c}^* = (c_0^*, c_1^*) \leftarrow \mathsf{Enc}(\mathsf{pk}^*, \mathsf{msg})$, then $\mathbf{c} = (\mathbf{O}^{-1} \cdot \mathbf{c}_0^*, c_1^*)$ is distributed according to $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$.

Proof. We have $\mathbf{c}_0 = \mathbf{A} \cdot \mathbf{x} + \mathbf{e}$ for $\mathbf{x} \leftarrow\!\!\$\ \mathbb{Z}_q^k$ and $\mathbf{e} \leftarrow\!\!\$\ \chi^n$, and thus

$$\mathbf{O} \cdot \mathbf{c}_0 = \mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U} \cdot \mathbf{U}^{-1} \cdot \mathbf{x} + \mathbf{O} \cdot \mathbf{e},$$

where $\mathbf{U} \in \mathcal{GL}_k(\mathbb{Z}_q)$ is the systematic form transformation matrix of $\mathbf{O} \cdot \mathbf{A}$. Observe that $\mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U}$ is precisely pk^*. Since $\mathbf{x}$ is uniformly random, so is $\mathbf{x}^* = \mathbf{U}^{-1} \cdot \mathbf{x}$. Since $\mathbf{e} \leftarrow \chi^n$, so is $\mathbf{O} \cdot \mathbf{e}$, since χ^n samples entry-wise and is thus not affected by permuting the entries, and χ is symmetric around $0 \in \mathbb{Z}_q$ and is thus not affected by changing the sign. This is consistent with

$$c_1 = \langle \mathbf{u}, \mathbf{x} \rangle + e' + \left\lfloor \frac{q}{p} \right\rceil \cdot \mathsf{msg} = \langle \mathbf{U}^\intercal \cdot \mathbf{u}, \mathbf{U}^{-1} \cdot \mathbf{x} \rangle + e' + \left\lfloor \frac{q}{p} \right\rceil \cdot \mathsf{msg} \pmod q.$$

The converse follows by remarking that if $\mathbf{O}$ updates $(\mathsf{pk}, \mathsf{sk})$ to $(\mathsf{pk}^*, \mathsf{sk}^*)$, then $\mathbf{O}^{-1}$ updates $(\mathsf{pk}^*, \mathsf{sk}^*)$ to $(\mathsf{pk}, \mathsf{sk})$. □

Theorem 3. *Let n, k, h, q be positive integers parametrised by λ with $n \geq (1+c) \cdot k \cdot \log_2(q) + k + h$ for a positive real constant $c > 0$, $2 \cdot h \leq k$ and q prime. Assuming the advantage of any PPT adversary in distinguishing $\mathrm{LWE}^h_{k,n,q,\chi}$ and in distinguishing $\Delta PCE^h_{n,k,q}$ is negligible in λ, the UPKE scheme from Definition 13 is IND-CR-CPA secure.*

Proof. We proceed through a sequence of hybrids. Throughout we denote by ε_i the advantage of the adversary $\mathcal{A}$ against game Game_i.

$\mathsf{Game}_0(\lambda)$**:** The game given in Figure 5 is the usual IND-CR-CPA game for our construction. The challenger runs $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KGen}$, sets time $T \coloneqq 0$ and samples $b \leftarrow\!\!\$\ \{0,1\}$, then sends pk_0 to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ with access to the update oracle UpdO sends back messages $\mathsf{msg}_0, \mathsf{msg}_1$. The challenger then runs $\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_T, \mathsf{msg}_b)$, and sends ctxt to $\mathcal{A}$. Once the adversary is done querying UpdO, the challenger performs an honest update by running $(\mathsf{pk}, \mathsf{up}) \leftarrow \mathsf{UpdPk}(\mathsf{pk}_T)$ and $\mathsf{sk} \leftarrow \mathsf{UpdSk}(\mathsf{sk}_T, \mathsf{up})$, and sends $(\mathsf{pk}, \mathsf{sk}, \mathsf{up})$ to $\mathcal{A}$, who returns a bit.

$\mathsf{Game}_0(\lambda)$	$\mathsf{Game}_1(\lambda)$
$T \coloneqq 0$; $b \leftarrow\!\!\$\ \{0,1\}$	$T \coloneqq 0$; $b \leftarrow\!\!\$\ \{0,1\}$
	$\rho^* \leftarrow\!\!\$\ \{0,1\}^\lambda$; $\mathcal{Q}[\rho^*] \coloneqq \mathbf{O}^* \leftarrow\!\!\$\ \mathcal{O}_n(\mathbb{Z})$
$(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KGen}(1^\lambda)$	$(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KGen}(1^\lambda)$
$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{UpdO},\mathsf{RO}}(\mathsf{pk}_0)$	$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{UpdO},\mathsf{RO}}(\mathsf{pk}_0)$
$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_T, \mathsf{msg}_b)$	$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{msg}_b)$
$\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{UpdO},\mathsf{RO}}(\mathsf{ctxt}, \mathsf{st})$	$\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{UpdO},\mathsf{RO}}(\mathsf{ctxt}, \mathsf{st})$
$(\mathsf{pk}^*, \mathsf{up}) \leftarrow \mathsf{UpdPk}^{\mathsf{RO}}(\mathsf{pk}_T)$	$(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}) \leftarrow \mathsf{Upd}^*(\mathsf{pk}_T, \mathsf{sk}_T, \mathbf{O}^*)$
$\mathsf{sk}^* \leftarrow \mathsf{UpdSk}^{\mathsf{RO}}(\mathsf{sk}_T, \mathsf{up})$	
$b' \leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}, \mathsf{st})$	$b' \leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}, \mathsf{st})$
return $b = b'$	**return** $b = b'$

$\mathsf{UpdO}(\rho)$	$\mathsf{Upd}^*(\mathsf{pk}, \mathsf{sk}, \mathbf{O}^*)$
$(\mathsf{pk}_{T+1}, \mathsf{up}_T) \leftarrow \mathsf{UpdPk}^{\mathsf{RO}}(\mathsf{pk}_T; \rho)$	**parse** $(\mathbf{A}, \mathbf{v}) \leftarrow \mathsf{pk}$
$\mathsf{sk}_{T+1} \leftarrow \mathsf{UpdSk}^{\mathsf{RO}}(\mathsf{sk}_T, \mathsf{up}_T)$	**parse** $\mathbf{r} \leftarrow \mathsf{sk}$
$T \coloneqq T + 1$	$\mathbf{A}^*, \mathbf{U} \coloneqq \mathrm{SF}(\mathbf{O}^* \cdot \mathbf{A})$
$\mathsf{RO}(\rho)$	$\mathsf{pk}^* \coloneqq \mathbf{A}^*$
if $\rho \notin \mathcal{Q}$: $\mathcal{Q}[\rho] \leftarrow\!\!\$\ \mathcal{O}_n(\mathbb{Z})$	$\mathsf{sk}^* \coloneqq \mathbf{O}^* \cdot \mathbf{r}$
return $\mathcal{Q}[\rho]$	$\mathsf{up} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{s})$
	return $(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up})$

Fig. 5: Game_0 and Game_1

$\mathsf{Game}_1(\lambda)$**:** This game given in Figure 5 is the same as Game_0, except that the challenge ciphertext ctxt and the final update token up are encrypted with respect to pk_0 instead of the public key of their respective epoch.

Denote by $\mathbf{O}_i$ the update transformation in epoch i, i.e. $\mathbf{O}_i$ updates pk_{i-1} to pk_i. By Lemma 10, knowing the cumulative $\mathbf{O} = \mathbf{O}_T \cdots \mathbf{O}_1$ in any epoch T, an adversary playing Game_0 upon receiving $\mathsf{ctxt} = (\mathbf{c}_0, c_1) \leftarrow \mathsf{Enc}(\mathsf{pk}_T, \mathsf{msg}_b)$ can always compute $\mathsf{ctxt}^* = (\mathbf{O}^{-1} \cdot \mathbf{c}_0, c_1)$ which is distributed exactly as $\mathsf{Enc}(\mathsf{pk}_0, \mathsf{msg}_b)$, and likewise with the encryptions in up (for a later epoch). The converse is also true, i.e. given a ciphertext $\mathsf{ctxt} = (\mathbf{c}_0, c_1) \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$, any adversary can always compute $\mathsf{ctxt}^* = (\mathbf{O} \cdot \mathbf{c}_0, c_1)$ for their chosen $\mathbf{O}$ that is distributed exactly as $\mathsf{Enc}(\mathsf{pk}^*, \mathsf{msg})$ where pk^* is pk updated with $\mathbf{O}$. Thus, for any adversary winning Game_0 with advantage ε_0 we can construct an adversary $\mathcal{B}$ winning Game_1 with advantage $\varepsilon_1 = \varepsilon_0$ (and vice versa).

$\mathsf{Game}_2(\lambda)$**:** This game given in Figure 6 is the same as Game_1, except that the adversary does not have access to UpdO.

$\mathsf{Game}_2(\lambda)$

~~$T \coloneqq 0$~~; $b \leftarrow\!\!\$\ \{0,1\}$; $\rho^* \leftarrow\!\!\$\ \{0,1\}^\lambda$; $\mathcal{Q}[\rho^*] \coloneqq \mathbf{O}^* \leftarrow\!\!\$\ \mathcal{O}_n(\mathbb{Z})$; $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KGen}(1^\lambda)$

$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pk}_0)$; $\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{msg}_b)$; $\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{ctxt}, \mathsf{st})$

$(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}) \leftarrow \mathsf{Upd}^*(\mathsf{pk}_0, \mathsf{sk}_0, \mathbf{O}^*)$; $b' \leftarrow \mathcal{A}^{\mathsf{RO}}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}, \mathsf{st})$

return $b = b'$

Fig. 6: Game_2

For any adversary $\mathcal{A}$ winning Game_1 with advantage ε_1 we can construct an adversary $\mathcal{B}$ winning Game_2 with advantage $\varepsilon_2 = \varepsilon_1$ by noticing that UpdO can be simulated using RO. Indeed, $\mathcal{B}$ is just $\mathcal{A}$ except that any time it wants to call UpdO with randomness ρ, it instead calls $\mathbf{O} \leftarrow \mathsf{RO}(\rho)$ and then manually updates the current key pk_T with $\mathbf{O}$ as in UpdPk. Notice the updated keys pk_T of any epoch T are not used anywhere in Game_1.

At this point, there is no need to keep the epoch counter T anymore. Simplifying notation, the view of the adversary is

$$\Big(\mathsf{pk}_0 = (\mathbf{A}, \mathbf{u}),\ \mathsf{pk} = (\mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U}, \mathbf{U}^{\mathrm{T}} \cdot \mathbf{u}),\ \mathsf{sk} = \mathbf{O} \cdot \mathbf{r},$$
$$\mathsf{ctxt} \leftarrow \mathsf{Enc}((\mathbf{A}, \mathbf{u}), \mathsf{msg}_b),\ \mathsf{up} = \mathsf{Enc}((\mathbf{A}, \mathbf{u}), \rho^*)\ ;$$
$$\mathbf{r}^{\mathrm{T}} \cdot \mathbf{A} \equiv \mathbf{u},\ \mathbf{O} = \mathsf{RO}(\rho^*)\Big),$$

where $\mathbf{r}$ is the initial secret key.

$\mathsf{Game}_3(\lambda)$**:** This game given in Figure 7 is the same as Game_2, except that if the adversary ever queries $\mathsf{RO}(\rho^\star)$, the game immediately returns 1, i.e. aborts with a win for the adversary. To denote this bad query we write RO_{ρ^*} instead of RO. Note the game still performs the honest update with $\mathbf{O} = \mathcal{Q}[\rho^\star]$ without aborting, but the adversary is now rewarded for querying the random seed $\rho^\star$ for $\mathbf{O}$.

We have $\varepsilon_2 \leq \varepsilon_3$ where ε_3 is the sum of the advantage of the adversary winning by guessing b and the probability of the adversary winning by querying $\mathsf{RO}(\rho^\star)$.

$\mathsf{Game}_4(\lambda)$**:** This game given in Figure 7 is the same as Game_3, except that $\mathbf{O} \leftarrow\!\!\$\ \mathcal{O}_n(\mathbb{Z})$ and not $\mathbf{O} \leftarrow \mathsf{RO}(\rho^*)$. In the Random Oracle Model $\mathbf{O}$ is independent of ρ^* until ρ^* is queried and we have $\varepsilon_4 = \varepsilon_3$. In more detail, we can separate two cases. If $\mathcal{A}$ never queries $\mathsf{RO}_{\rho^*}(\rho^*)$ then Game_4 is identical to Game_3 and the advantage is the same, since $\mathbf{O} \leftarrow\!\!\$\ \mathcal{O}_n(\mathbb{Z})$ in both cases. However, if $\mathcal{A}$ at any point queries $\mathsf{RO}_{\rho^*}(\rho^*)$, it automatically wins, so the advantage cannot decrease.

$\mathsf{Game}_3(\lambda)$	$\mathsf{Game}_4(\lambda)$
$b \leftarrow\!\$\ \{0,1\}$	$b \leftarrow\!\$\ \{0,1\}$
$\rho^* \leftarrow\!\$\ \{0,1\}^\lambda$; $\mathcal{Q}[\rho^*] \coloneqq \mathbf{O}^* \leftarrow\!\$\ \mathcal{O}_n(\mathbb{Z})$	$\rho^* \leftarrow\!\$\ \{0,1\}^\lambda$; ~~$\mathcal{Q}[\rho^*] \coloneqq \mathbf{O}^* \leftarrow\!\$\ \mathcal{O}_n(\mathbb{Z})$~~
$(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KGen}(1^\lambda)$	$(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KGen}(1^\lambda)$
$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}_0)$	$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}_0)$
$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{msg}_b)$	$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{msg}_b)$
$\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{ctxt}, \mathsf{st})$	$\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{ctxt}, \mathsf{st})$
$(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}) \leftarrow \mathsf{Upd}^*(\mathsf{pk}_0, \mathsf{sk}_0, \mathbf{O}^*)$	$(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}) \leftarrow \mathsf{Upd}^*(\mathsf{pk}_0, \mathsf{sk}_0, \mathbf{O}^*)$
$b' \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}, \mathsf{st})$	$b' \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}, \mathsf{st})$
return $(b = b') \vee$ `free_win`	**return** $(b = b') \vee$ `free_win`

$\mathsf{RO}_{\rho^*}(\rho)$

if $\rho = \rho^*$: `free_win` $\coloneqq 1$; **return** $\perp$
if $\rho \notin \mathcal{Q}$: $\mathcal{Q}[\rho] \leftarrow\!\$\ \mathcal{O}_n(\mathbb{Z})$
return $\mathcal{Q}[\rho]$

Fig. 7: Game_3 and Game_4

The view of the adversary is as follows

$$\Big(\mathsf{pk}_0 = (\mathbf{A}, \mathbf{u}),\ \mathsf{pk} = (\mathbf{O} \cdot \mathbf{A} \cdot \mathbf{U}, \mathbf{U}^{\mathrm{T}} \cdot \mathbf{u}),\ \mathsf{sk} = \mathbf{O} \cdot \mathbf{r},\ \mathsf{ctxt} \leftarrow \mathsf{Enc}((\mathbf{A}, \mathbf{u}), \mathsf{msg}_b),\ \mathsf{up} = \mathsf{Enc}((\mathbf{A}, \mathbf{u}), \rho^*)\ ;\ \mathbf{r}^{\mathrm{T}} \cdot \mathbf{A} \equiv \mathbf{u},\ \mathbf{O} \leftarrow\!\$\ \mathcal{O}_n(\mathbb{Z})\Big),$$

where $\mathbf{r}$ is the initial secret key.

$\mathsf{Game}_5(\lambda)$**:** This game given in Figure 8 is the same as Game_4, except that $(\mathsf{pk}, \mathsf{sk}) = ((\mathbf{B}, \mathbf{v}), \mathbf{r}')$ is a freshly sampled key unrelated to $((\mathbf{A}, \mathbf{u}), \mathbf{r})$. We show that Game_4 is indistinguishable from Game_5 under the PCE assumption (Definition 1) in Lemma 11. Then $\varepsilon_5 = \varepsilon_4 + \varepsilon_{\mathrm{PCE}}$, where $\varepsilon_{\mathrm{PCE}}$ is the advantage of the adversary in winning $\Delta\mathrm{PCE}^h_{n,k,q}$, negligible by assumption. The view of the adversary is

$$\Big(\mathsf{pk}_0 = (\mathbf{A}, \mathbf{u}),\ \mathsf{pk} = (\mathbf{B}, \mathbf{v}),\ \mathsf{sk} = \mathbf{r}',\ \mathsf{ctxt} \leftarrow \mathsf{Enc}((\mathbf{A}, \mathbf{u}), \mathsf{msg}_b),\ \mathsf{up} = \mathsf{Enc}((\mathbf{A}, \mathbf{u}), \rho^*)\ ;\ \mathbf{r}^{\mathrm{T}} \cdot \mathbf{A} \equiv \mathbf{u},\ \mathbf{r}'^{\mathrm{T}} \cdot \mathbf{B} \equiv \mathbf{v}\Big),$$

where $\mathbf{r}$ is the initial secret key.

$\mathsf{Game}_6(\lambda)$**:** This game given in Figure 8 is the same as Game_5, except that the h-hollow $\mathbf{A}$ and vector $\mathbf{u}$ are sampled uniformly at random, without knowing the new corresponding $\mathbf{r}$ such that $\mathbf{r}^{\mathrm{T}} \cdot \mathbf{A} = \mathbf{u}^{\mathrm{T}}$. The games Game_5 and Game_6 are indistinguishable by Lemma 9, and we get $\varepsilon_6 = \varepsilon_5 + \varepsilon_{\mathrm{LHL}}$, where $\varepsilon_{\mathrm{LHL}}$ is the negligible statistical distance from Lemma 9. The adversary's view is

$$\Big(\mathsf{pk}_0 = (\mathbf{A}, \mathbf{u}),\ \mathsf{pk} = (\mathbf{B}, \mathbf{v}),\ \mathsf{sk} = \mathbf{r}',\ \mathsf{ctxt} \leftarrow \mathsf{Enc}((\mathbf{A}, \mathbf{u}), \mathsf{msg}_b),\ \mathsf{up} = \mathsf{Enc}((\mathbf{A}, \mathbf{u}), \rho^*)\ ;\ \mathbf{r}'^{\mathrm{T}} \cdot \mathbf{B} \equiv \mathbf{v}\Big).$$

$\mathsf{Game}_7(\lambda)$**:** This game given in Figure 9 is the same as Game_6, except that $\mathsf{ctxt} \leftarrow\!\$\ \mathbb{Z}_q^{n+1}$ and $\mathsf{up} = [\mathbf{u}_i \leftarrow\!\$\ \mathbb{Z}_q^{n+1}]_{i=1}^{\lambda/\log_2(p)}$. The games Game_6 and Game_7 are indistinguishable by using Theorem 1 in a standard hybrid argument, and we get $\varepsilon_7 = \varepsilon_6 + h \cdot \varepsilon_h$, where ε_h is the advantage of winning $\mathrm{LWE}^h_{k,n,q,\chi}$ from Definition 12, negligible by the LWE assumption.

$\mathsf{Game}_5(\lambda)$	$\mathsf{Game}_6(\lambda)$
$b \leftarrow\$ \{0,1\}$; $\rho^* \leftarrow\$ \{0,1\}^\lambda$	$b \leftarrow\$ \{0,1\}$; $\rho^* \leftarrow\$ \{0,1\}^\lambda$
$(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KGen}(1^\lambda)$	$\mathsf{pk}_0 \leftarrow \mathsf{SampleCode}(n,k,h,q) \times \mathcal{U}\left(\mathbb{Z}_q^k\right)$
$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}_0)$	$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}_0)$
$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{msg}_b)$	$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{msg}_b)$
$\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{ctxt}, \mathsf{st})$	$\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{ctxt}, \mathsf{st})$
$(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KGen}(1^\lambda)$	$(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KGen}(1^\lambda)$
$\mathsf{up} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \rho^*)$	$\mathsf{up} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \rho^*)$
$b' \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}, \mathsf{st})$	$b' \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}, \mathsf{st})$
return $(b = b') \vee$ `free_win`	**return** $(b = b') \vee$ `free_win`

Fig. 8: Game_5 and Game_6

$\mathsf{Game}_7(\lambda)$
$b \leftarrow\$ \{0,1\}$; $\rho^* \leftarrow\$ \{0,1\}^\lambda$; $\mathsf{pk}_0 \leftarrow \mathsf{SampleCode}(n,k,h,q) \times \mathcal{U}\left(\mathbb{Z}_q^k\right)$
$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}_0)$; $\mathsf{ctxt} \leftarrow\$ \mathbb{Z}_q^{n+1}$; $\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{ctxt}, \mathsf{st})$
$(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KGen}(1^\lambda)$; $\mathsf{up} \leftarrow\$ \mathbb{Z}_q^{(n+1)\times(\lambda/\log_2(p))}$; $b' \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}, \mathsf{st})$
return $(b = b') \vee$ `free_win`

Fig. 9: Game_7

The advantage of the adversary of guessing $b' = b$ in Game_7 is clearly 0 since ctxt is uniformly random with no relation to msg_b. The adversary can therefore only win Game_7 by querying $\mathsf{RO}(\rho^*)$, that is with negligible probability $2^{-\lambda}$. □

Lemma 11. *If there exists a (t,ϵ)-algorithm $\mathcal{A}$ that distinguishes Game_4 from Game_5, then there exists a $(\mathsf{poly}(\lambda)+t,\epsilon)$-algorithm $\mathcal{B}$ for $\Delta PCE^h_{n,k,q}$.*

Proof. The algorithm $\mathcal{B}$ receives a PCE instance $\mathbf{A}, \mathbf{B}$ where either $\mathbf{B} = \mathbf{P} \cdot \mathbf{A} \cdot \mathbf{U}$ for a secret permutation matrix $\mathbf{P}$ and $\mathbf{U}$ wlog. the systematic form transformation for $\mathbf{P} \cdot \mathbf{A}$, or random with respect to the hull dimension h. It then computes

$$\mathbf{a}^\intercal = \sum_{i=1}^{n} \mathbf{A}_i, \qquad \mathbf{b}^\intercal = \sum_{i=1}^{n} \mathbf{B}_i,$$

where $\mathbf{A}_i$ and $\mathbf{B}_i$ are rows of $\mathbf{A}$ and $\mathbf{B}$, respectively. Observe that multiplication of the all 1's row $([1]^n)^\intercal$ with $\mathbf{A}$ and $\mathbf{B}$ gives precisely $\mathbf{a}^\intercal$ and $\mathbf{b}^\intercal$.

The algorithm $\mathcal{B}$ then randomises the instance. It samples two uniformly random signed permutations $\mathbf{O_A} = \mathbf{P_A} \cdot \mathbf{D_A}, \mathbf{O_B} = \mathbf{P_B} \cdot \mathbf{D_B} \leftarrow\$ \mathcal{O}_n(\mathbb{Z})$, two uniformly random basis changes $\mathbf{U_A}, \mathbf{U_B} \leftarrow\$ \mathcal{GL}_k(\mathbb{Z}_q)$, and computes

$$\tilde{\mathbf{A}} = \mathbf{O_A} \cdot \mathbf{A} \cdot \mathbf{U_A}, \qquad \tilde{\mathbf{a}}^\intercal = \mathbf{a}^\intercal \cdot \mathbf{U_A}, \qquad \mathbf{r} = \mathbf{O_A} \cdot [1]^n,$$
$$\tilde{\mathbf{B}} = \mathbf{O_B} \cdot \mathbf{B} \cdot \mathbf{U_B}, \qquad \tilde{\mathbf{b}}^\intercal = \mathbf{b}^\intercal \cdot \mathbf{U_B}, \qquad \mathbf{r}' = \mathbf{O_B} \cdot [1]^n.$$

Observe $\mathbf{r}$ is a uniformly random element of $\{\pm 1\}^n$ and $\mathbf{r}^\intercal \cdot \tilde{\mathbf{A}} = \tilde{\mathbf{a}}^\intercal$, and that $\mathbf{r}'$ is the same for $\tilde{\mathbf{B}}$.

Let us show that if $(\mathbf{A}, \mathbf{B})$ is a real PCE instance, then $((\tilde{\mathbf{A}}, \tilde{\mathbf{a}}), \mathbf{r})$ is distributed according to $\mathsf{KGen}(\lambda)$, and $((\tilde{\mathbf{B}}, \tilde{\mathbf{b}}), \mathbf{r}')$ is exactly the key pair updated with $\mathbf{O} = \mathbf{O_B} \cdot \mathbf{P} \cdot \mathbf{O_A}^{-1}$, which is a uniformly random signed permutation, since the vector $[1]^n$ is fixed by any permutation, including $\mathbf{P}$. Similarly, if $(\mathbf{A}, \mathbf{B})$ is a random

instance, then $((\tilde{\mathbf{A}}, \tilde{\mathbf{a}}), \mathbf{r})$ and $((\tilde{\mathbf{B}}, \tilde{\mathbf{b}}), \mathbf{r}')$ are both distributed independently according to $\mathsf{KGen}(\lambda)$. Indeed, define the sets

$$\mathcal{T} = \left\{(\mathbf{A}, \mathbf{r}) \; ; \; \mathbf{r} \in \{\pm 1\}^n \text{ s.t. } \mathbf{A}^{\mathrm{T}} \cdot \mathbf{r} = \mathbf{a}\right\}, \qquad \mathcal{S} = \left\{\mathbf{A} \; ; \; \mathbf{A}^{\mathrm{T}} \cdot [1]^n = \mathbf{a}\right\},$$

where $\mathbf{A}$ ranges over $[n, k]$-linear code generator matrices over $\mathbb{Z}_q$ with hull dimension h and $\mathbf{a}$ is some fixed vector. We show there exists a surjective function $f \colon \mathcal{T} \to \mathcal{S}$ that is regular in the sense that there exists a constant C depending only on the parameters such that for every $\mathbf{A} \in \mathcal{S}$ we have $|f^{-1}(\mathbf{A})| = C$. For a pair $(\mathbf{A}, \mathbf{r}) \in \mathcal{T}$ let $\mathbf{D}_{\mathbf{r}} \in \mathrm{diag}(\{\pm 1\}^n)$ be the unique diagonal matrix such that $\mathbf{D}_{\mathbf{r}} \cdot \mathbf{r} = [1]^n$. The function f maps $(\mathbf{A}, \mathbf{r}) \mapsto \mathbf{D} \cdot \mathbf{A}$. The function f is surjective since for every $\mathbf{A} \in \mathcal{S}$ we have $(\mathbf{A}, [1]^n) \in f^{-1}(\mathbf{A})$. For regularity note that if $f(\mathbf{A}, \mathbf{r}) = f(\mathbf{A}', \mathbf{r}')$ then $\mathbf{A} = \mathbf{D}_{\mathbf{r}}^{-1} \cdot \mathbf{D}_{\mathbf{r}'} \cdot \mathbf{A}'$, that is $\mathbf{A}$ and $\mathbf{A}'$ have the same rows up to a sign, so for every $\mathbf{A} \in \mathcal{S}$ we have $|f^{-1}(\mathbf{A})| = 2^n$, concluding the proof.

In particular, this means that the uniform distribution on $\mathcal{S}$ factors through f to the uniform distribution on $\mathcal{T}$, and vice versa. Given a random element $\mathbf{A} \in \mathcal{S}$, picking a uniformly random $\mathbf{D} \in \mathrm{diag}(\{\pm 1\}^n)$ and outputting $(\mathbf{D} \cdot \mathbf{A}, \mathbf{D} \cdot [1]^n)$ gives a random element of $\mathcal{T}$. Observe also that applying a permutation before or after the diagonal matrix does not change the distribution, since permutations are bijections on $\mathcal{T}$ (applied to both entries of the pair) and on $\mathcal{S}$.

Thus $\mathcal{B}$ proceeds as shown in Figure 10. It sets $(\mathsf{pk}_0, \mathsf{sk}_0) = ((\tilde{\mathbf{A}}, \tilde{\mathbf{a}}), \mathbf{r})$ and $(\mathsf{pk}, \mathsf{sk}) = ((\tilde{\mathbf{B}}, \tilde{\mathbf{b}}), \mathbf{r}')$, and samples $b \leftarrow\!\$\; \{0,1\}$ and $\rho^* \leftarrow\!\$\; \{0,1\}^\lambda$, which it encrypts into $\mathsf{up} = [u_i \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \rho_i^*)]_{\rho_i^* \in \rho^*}$. It then plays the game as $\mathcal{A}$ expects, also simulating the RO queries, and returns the final output of $\mathcal{A}$. If $(\mathbf{A}, \mathbf{B})$ is a real

$\mathcal{B}(\mathbf{A}, \mathbf{B})$	$\mathsf{RO}_{\rho^*}(\rho)$
$\rho^* \leftarrow\!\$\; \{0,1\}^\lambda$; $b \leftarrow\!\$\; \{0,1\}$	**if** $\rho \notin \{0,1\}^\lambda$: **abort**
$(\mathsf{pk}_0, \mathsf{sk}_0) \coloneqq ((\tilde{\mathbf{A}}, \tilde{\mathbf{a}}), \mathbf{r})$	**if** $\rho = \rho^*$: **abort**
$(\mathsf{st}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}_0)$	**if** $\rho \in \mathcal{Q}$:
$\mathsf{ctxt} \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \mathsf{msg}_b)$	**return** $\mathcal{Q}[\rho]$
$\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{ctxt}, \mathsf{st})$	**else** :
$(\mathsf{pk}, \mathsf{sk}) \coloneqq ((\tilde{\mathbf{B}}, \tilde{\mathbf{b}}), \mathbf{r}')$	$\mathbf{O} \leftarrow\!\$\; \mathcal{O}_n(\mathbb{Z})$
$\mathsf{up} \coloneqq [\mathbf{u}_i \leftarrow \mathsf{Enc}(\mathsf{pk}_0, \rho_i^*)]_{\rho_i^* \in \rho^*}$	$\mathcal{Q}[\rho] \coloneqq \mathbf{O}$
$b' \leftarrow \mathcal{A}^{\mathsf{RO}_{\rho^*}}(\mathsf{pk}, \mathsf{sk}, \mathsf{up}, \mathsf{st})$	**return** $\mathbf{O}$
return $(b = b')$	

Fig. 10: An adversary $\mathcal{B}$ against PCE.

PCE instance, then we are simulating Game_4, and if $(\mathbf{A}, \mathbf{B})$ is a random instance, then we are simulating Game_5. The success probability of $\mathcal{B}$ is the same as that of $\mathcal{A}$, and all additional operations $\mathcal{B}$ performs use only linear algebra which are efficient. □

5.2 Parameter Selection

We select parameters for our scheme based on a security parameter λ. There are multiple constraints we must take into account. First, for our construction to be correct we require $q > \frac{p^2}{2} + p \cdot s \cdot \sqrt{(\lambda+1) \cdot (n+1)}$ where s is the Gaussian parameter of the error distribution χ. Second, for the underlying dual-Regev to be secure, we require $\mathrm{LWE}_{k,n,q,\chi}$ to be hard, and the particular case of Lemma 2 satisfied. Third, for the update mechanism to be secure, we require $\mathrm{PCE}^h_{n,k,q}$ to be hard and by Lemma 9 a slightly higher bound on n than in Lemma 2.

Regarding the hardness of $\mathrm{PCE}^h_{n,k,q}$, we observe in Appendix A that the hull dimension h needs to be large enough to mitigate traditional hull attacks [Sen00,BOST19], and collision-based attacks on the hull [Sen00,

Prop. 12].[13] For the hull attacks, we make a conservative estimate that the SSA algorithm [Sen00] runs in $\mathcal{O}(q^h)$ time, and that the BOS algorithm runs in $\mathcal{O}(n^h)$ time, and so we require $h \geq \max\left\{\frac{\lambda}{\log_2(q)}, \frac{\lambda}{\log_2(n)}\right\}$. Since $q \geq n$ in LWE, we take $h \geq \left\lceil \frac{\lambda}{\log_2(n)} \right\rceil$. For the hull collision attack, we observe in Appendix A that we require $\frac{1}{2} \cdot \mathrm{H}\left(\frac{h}{n}\right) \cdot n \geq \lambda$, where H is the binary entropy function.

For the Hollow LHL bound from Lemma 9, observe we require

$$n - \frac{\lambda}{\log_2(n)} \geq (1+c) \cdot k \cdot \log_2 q + k = \left(1 + c + \frac{1}{\log_2(q)}\right) \cdot k \cdot \log_2(q),$$

thus it is enough to consider a slightly larger constant c than in Lemma 2. It remains only to set k large enough for $\mathrm{LWE}_{k,n,q,\chi}$ to be hard and the above bound is satisfied. Our Python script estimating parameters is attached to this document[14] and uses the Lattice Estimator [APS15][15] to construct parameters based on the security parameter λ, the LHL constant c, the Gaussian parameter s, and the message space size p. Note that the required hull dimension h is low compared to k and n, so it does not play a significant role in the above LHL bound. We present our parameters along with ciphertext and update token sizes in KiB in Table 1.

References

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. 2

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. 22

Bab16. László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 684–697. ACM Press, June 2016. 27

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, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, pages 23–43. Springer, Cham, 2021. 2, 24, 25, 27

BBPS23. Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. On the computational hardness of the code equivalence problem in cryptography. *Advances in Mathematics of Communications*, 17(1):23–55, 2023. 24, 26

BDR. Jongmin Baek, Anand Deopurkar, and Katherine Redfield. Points on Conics Modulo p. 28, 29

Ber15. Elwyn Berlekamp. *Algebraic Coding Theory*. World Scientific, revised edition, 2015. 25

Beu20a. Ward Beullens. Not enough LESS: An improved algorithm for solving code equivalence problems over $\mathbb{F}_q$. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 387–403. Springer, Cham, October 2020. 22, 24, 28

Beu20b. Ward Beullens. Not enough LESS: An improved algorithm for solving code equivalence problems over $\mathbb{F}_q$. Cryptology ePrint Archive, Report 2020/801, 2020. 24, 25

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. 2, 21, 24, 27, 28, 31, 32

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. 1

Coh93. Henri Cohen. *A Course in Computional Algebraic Number Theory*. Springer, 1993. Third, Corrected Printing 1996. 10

CPS23. Tung Chou, Edoardo Persichetti, and Paolo Santini. On linear equivalence, canonical forms, and digital signatures. Cryptology ePrint Archive, Report 2023/1533, 2023. 22, 24, 26, 28

[13] We ran the estimator provided in [Beu20a] and updated it to include CF attacks [CPS23,Now24].

[14] Also provided in Appendix D.

[15] Commit `431ee7d` of https://github.com/malb/lattice-estimator/.

CW35. Herrn Chevalley and Ewald Warning. Bemerkung zur vorstehenden Arbeit von Herrn Chevalley. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 11(1):76–83, 1935. 9

DG23. Léo Ducas and Shane Gibbons. Hull attacks on the lattice isomorphism problem. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 177–204. Springer, Cham, May 2023. 6

DJK22. Yevgeniy Dodis, Daniel Jost, and Harish Karthikeyan. Forward-secure encryption with fast forwarding. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 3–32. Springer, Cham, November 2022. 1

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. 2, 3, 7, 8, 15

DvW22. Léo Ducas 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. 3, 25

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. 2, 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. 2, 3, 5

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. 2

Kat71. Nicholas M. Katz. On a Theorem of Ax. *American Journal of Mathematics*, 93(2):485–499, 1971. 9

LB88. Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 275–280. Springer, Berlin, Heidelberg, May 1988. 24

Leo82. Jeffrey Leon. Computing automorphism groups of error-correcting codes. *IEEE Transactions on Information Theory*, 28(3):496–511, 1982. 24, 25

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. 4

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *DCC*, 75(3):565–599, 2015. 5

MP13. 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. 7

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. 7

Now24. Julian Nowakowski. An improved algorithm for code equivalence. Cryptology ePrint Archive, Report 2024/1272, 2024. 22, 24, 26, 28

Pet10. Christiane Peters. Information-set decoding for linear codes over F_q. In Nicolas Sendrier, editor, *The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010*, pages 81–94. Springer, Berlin, Heidelberg, May 2010. 24

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962. 24

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. 6

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, 2009. 6, 7

Sen97. Nicolas Sendrier. On the Dimension of the Hull. *SIAM Journal on Discrete Mathematics*, 10(2):282–293, 1997. 2, 5, 24, 27

Sen00. Nicolas Sendrier. Finding the permutation between equivalent linear codes: the support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000. 2, 21, 22, 24, 27, 28

SS13. Nicolas Sendrier and Dimitris E. Simos. The hardness of code equivalence over and its application to code-based cryptography. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 203–216. Springer, Berlin, Heidelberg, June 2013. 6

A Hardness of (Permutation) Code Equivalence

This section provides a short survey on the hardness of solving PCE. We consider two attacks that rely on finding codewords with with low hamming weight, namely Leon's algorithm [Leo82] and its improvement in the case of large fields by Beullens [Beu20a]. Both focus on finding codewords with colliding multisets, hence we call them *collision* attacks. Another collision-based attack was recently introduced by Chou, Persichetti, and Santini [CPS23], and further improved by Nowakowski [Now24]. We then consider *hull* attacks, namely Sendrier's Support Splitting Algorithm [Sen00] which excels when the hull dimension is small, and the so-called BOS algorithm by Bardet, Otmani and Saeed-Taha [BOST19], which exploits a trivial hull (and can also be extended to any small hull dimension). Hull attacks can be especially devastating, since random codes typically have small dimension that asymptotically depends only on q [Sen97], forcing us to restrict ourselves to weakly self-dual codes or similar families of codes with a large hull dimension. For a more thorough survey on the hardness of code equivalence, one should see [BBPS21,BBPS23] and [CPS23, App. A].

By the weight of the codeword $\omega(\mathbf{x})$ we mean its Hamming weight, i.e. the number of its non-zero entries as a vector. More generally, we denote the support of a set of codewords A as $\mathcal{S}(A)$ and define it as the subset of indices $\mathcal{S}(A) \subseteq \{1, \dots, n\}$ such that $i \in \mathcal{S}(A)$ if and only if A has a codeword with a non-zero entry at i. Clearly $\omega(\mathbf{x}) = |\mathcal{S}(\{\mathbf{x}\})|$. We denote by $B^w(\mathfrak{C})$ the set of all codewords of $\mathfrak{C}$ with weight w, and denote by $A^w(\mathfrak{C})$ the same set up to multiplication by scalars, i.e. if $\omega(\mathbf{x}) = w$ for $\mathbf{x} \in \mathfrak{C}$ then there exists precisely one $a \in \mathbb{Z}_q^*$ such that $a \cdot x \in A^w(\mathfrak{C})$.

Proposition 3 ([BBPS21, Prop. 1]). *For a random code C and any weight $w \leq n$, the expected cardinality of $A^w(\mathfrak{C})$ is given by*

$$N_w \approx \binom{n}{w} \cdot (q-1)^{w-2} \cdot q^{k-n+1}.$$

A.1 Information Set Decoding

For the first two collision-based algorithms, we will need an algorithm for Information Set Decoding (ISD) as a subroutine to find low-weight codewords. These are many ISD algorithm, e.g. Beullens uses the Lee-Brickell algorithm [LB88] in [Beu20a] because of the simplified heuristic analysis, while in [BBPS21] the authors use the Peters algorithm [Pet10] and the authors of [CPS23] compare their algorithm to Prange's [Pra62]. We summarise Beullens' estimates for Lee-Brickell made in [Beu20b, Sec. 2.2].

Beullens estimates that the Lee-Brickell algorithm (with parameter $p = 2$) finds a distinguished codeword x of target weight w after approximately

$$C_\infty(q, n, k, w) = \mathcal{O}\left(\frac{q \cdot \binom{n}{w}}{\binom{n-k}{w-2}}\right)$$

row operations. If the number $N = (q-1) \cdot N_w$ of codewords of weight w is small enough (i.e. the weight is sufficiently small), the cost of finding (any) one codeword of weight w is then estimated as $C_1(q, n, k, w) \approx \frac{C_\infty(q,n,k,w)}{N}$. Finding L distinct codewords of weight w therefore costs

$$C_L(q, n, k, w) \approx C_\infty(q, n, k, w) \cdot \left(\sum_{i=0}^{L-1} \frac{q}{L-i}\right),$$

which simplifies to

$$C_L(q, n, k, w) \approx \frac{L}{N} \cdot C_\infty(q, n, k, w)$$

if L is much smaller than N. If we wish to find all codewords, i.e. $L = N$, a better cost to note is then

$$C_N(q, n, k, w) \approx C_\infty(q, n, k, w) \cdot \ln(N),$$

since $\sum_{i=1}^{N} \frac{1}{i} \approx \ln(N)$.

A.2 Leon's Algorithm

The first algorithm we cover is due to Leon [Leo82], and the main idea is somewhat similar to the approach of solving LIP using invariants in [DvW22]. Suppose we have two codes $\mathfrak{C}_1$ and $\mathfrak{C}_2$ for which $\pi(\mathfrak{C}_1) = \mathfrak{C}_2$. Then for each subset $X \subseteq \mathfrak{C}_1$ there exists a subset $Y \subseteq \mathfrak{C}_2$ of the same cardinality such that $\pi(X) = Y$. The first thing to note is that among all the mappings between X and Y, π (restricted to X) must be one of them. The second thing to notice is that the map π preserves "geometric" invariants, namely that $\omega(x) = \omega(\pi(x))$ for all $x \in \mathfrak{C}_1$ (cf. [DvW22] on geometric invariants). For this reason, Leon's algorithm focuses on finding sets of low-weight codewords that span each code, finding permutations between them, and checking whether these permutations are exactly π on the entire code $\mathfrak{C}_1$. Denote by $\mathcal{S}(X, Y)$ the subgroup of permutations $\sigma \in \mathcal{P}_n$ such that $\sigma(X) = Y$.

For a target weight w, the algorithm thus proceeds roughly as follows:

1. Compute $B_i^w \coloneqq B^w(\mathfrak{C}_i)$ for $i = 1, 2$.
2. Find (the group generators of) $\mathcal{S}(B_1^w, B_2^w)$.
3. Check whether $\sigma(\mathfrak{C}_1) = \mathfrak{C}_2$ for $\sigma \in \mathcal{S}(B_1^w, B_2^w)$.

Leon proved in [Leo82] that the complexity of the last two steps is polynomial in the cardinality of B_i^w, which can be estimated as $(q-1) \cdot N_w$ by Proposition 3. Hence w needs to be small enough (slightly larger than the minimum distance of the code, estimated by the Gilbert-Varshamov bound for linear codes [Ber15, Thm. 13.74]), lest the sets B_i^w are too large. It w is too small, then there may be too many permutations to efficiently check, i.e. the subgroup $\mathcal{S}(B_1^w, B_2^w)$ is too large. For finding low-weight codewords, the standard is to use an algorithm based on ISD. Denote by $C_{\mathrm{ISD}}(q, n, k, w)$ the time complexity of an ISD algorithm searching for a codeword of weight w in an $[n, k]$-linear code over $\mathbb{Z}_q$. The complexity of the attack is then estimated from below as follows.

Proposition 4 ([BBPS21, Prop. 3]). *Let $\mathfrak{C}_1$ be a random $[n, k]$-linear code over $\mathbb{Z}_q$, and $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$ for a random permutation $\pi \in \mathcal{P}_n$. Then time required for Leon's algorithm with the target weight parameter w to find the permutation π given $\mathfrak{C}_1$ and $\mathfrak{C}_2$, i.e. to solve sPCE, is at least*

$$\mathcal{O}\left(C_{ISD}(q, n, k, w) \cdot 2 \sum_{i=1}^{N_w} \frac{1}{i}\right),$$

where w is such that $N_w \geq 2$. We can estimate this further [Beu20b] by

$$\mathcal{O}(C_{ISD}(q, n, k, w) \cdot \ln(N_w)).$$

A.3 Beullens' Algorithm

Beullens [Beu20b] refines Leon's algorithm by observing that not only does a permutation preserve the Hamming weight of a codeword, it also preserves its (non-zero) multiset of entries. The main improvement comes from the second observation that if the field is large enough (i.e. there are a lot of possible entries), the converse holds as well with large probability. That is, if $x \in \mathfrak{C}_1$ and $y \in \mathfrak{C}_2$ are low-weight codewords with the same multiset of entries, then with large probability $\pi(x) = y$. Namely, suppose we have B_1^w and B_2^w for w small, and $x \in B_1^w$ with a unique multiset, then we can immediately see which $y \in B_2^w$ it gets mapped to by π. The main idea is that if $\mathbb{Z}_q$ is sufficiently large, then with large probability a lot of multisets of the codewords in B_1^w will be unique, therefore we can gather pairs $(x, \pi(x))$, from which we recover π. Heuristically, at least $\Omega(\log(n))$ such pairs suffice to recover π.

Another improvement Beullens' algorithm makes is that it avoids computing the entirety of the sets B_i^w for $i = 1, 2$, using the fact that ISD-based algorithms for finding low-weight codewords can be tuned to how many codewords they output, making this algorithm probabilistic. Instead of computing the entirety of B_i^w, Beullens' algorithm computes $\Theta\left(\sqrt{|B_1^w| \log(n)}\right) = \Theta\left(\sqrt{(q-1)N_w \log(n)}\right)$ elements of B_1^w and B_2^w, from which one expects to find $\Theta(\log(n))$ pairs $(x, \pi(x))$ with unique multisets, enough to recover π.

The algorithm proceeds roughly as follows:

1. Set w maximal such that $\frac{n!}{(n-w)!}q^{-n+k} < \frac{1}{4\log(n)}$ and $w \leq n-k+1$.
2. Use an ISD algorithm repeatedly to generate a list L that contains

$$\ell = \sqrt{|B_1^w| q^{-n+k-1} 2\log(n)}$$

pairs of the form $(x, \mathsf{lex}(x))$ where $x \in B_1^w$ and $\mathsf{lex}(x)$ is the lexicographically first element of the set $\{\sigma(a \cdot x)\ ;\ \sigma \in \mathcal{P}_n, a \in \mathbb{Z}_q^*\}$ (notice again only non-zero entries play a role, since permutations that put the $n-w$ zero entries in x to the beginning are lexicographically below others, so we may just ignore them).
3. Initialize an empty list P and again use an ISD algorithm repeatedly to generate $y \in B_2^w$. If there is a $(x, \mathsf{lex}(x)) \in L$ such that $\mathsf{lex}(x) = \mathsf{lex}(y)$ (i.e. they have the same multiset of entries up to scalar multiplication), then append (x, y) to P. Stop when P has $2 \cdot \log(n)$ elements.
4. Iterate over all permutations $\sigma \in \mathcal{P}_n$ that satisfy $\langle \sigma(x) \rangle = \langle y \rangle$ (recall we considered codewords up to multiplication by a scalar) for all $(x, y) \in P$, until a permutation is found such that $\sigma(\mathfrak{C}_1) = \mathfrak{C}_2$.

Step 1 is optional and can be replaced by giving w as a parameter as in Leon's algorithm. The number of entries ℓ generated by an ISD algorithm in step 2 can also be adjusted.

Beullens estimates the complexity of his algorithm is dominated by the cost of computing $|L|$ low-weight codewords using an ISD algorithm (concretely Lee-Brickell), which is $2 \cdot C_{\mathrm{ISD}}(q, n, k, w)$, which is according to the above estimated as

$$2 \cdot C_{|L|}(q, n, k, w) \approx C_\infty(q, n, k, w) \cdot \frac{|L|}{N_w}.$$

We note that the authors of [BBPS23] also improve the LCE version of Beullens' algorithm, but we are mainly focused on PCE.

A.4 Algorithm from Canonical Forms

Recently, Chou, Persichetti, and Santini [CPS23] introduced a new version of code equivalence they call Canonical Form Linear (resp. Permutation) Equivalence Problem, abbreviated CF-LEP (resp. CF-PEP). They prove it computationally equivalent to LCE (resp. PCE), and while the upshot is that this allows them to significantly reduce the size of LESS signatures, the reduction also yields a new algorithm for LCE and PCE for large finite fields $\mathbb{Z}_q$. Nowakowski [Now24] then extended the algorithm to smaller fields $\mathbb{Z}_q$, and both algorithms have the following complexity.

Proposition 5 ([Now24, Thm. 4.4]). *Let* $\mathrm{H}\colon [0,1] \to \mathbb{R}$ *be the binary entropy function* $\mathrm{H}(x) = -x \cdot \log_2(x) - (1-x) \cdot \log_2(x)$*, and let* $r = \frac{k}{n}$ *be the rate of the codes. Then the complexity of the Canonical Forms algorithm in solving LCE is estimated as*

$$\Theta\Big(2^{\frac{1}{2} \cdot \mathrm{H}(r) \cdot n}\Big).$$

A.5 Support Splitting Algorithm

Neither of the above attacks depends in complexity on the dimension of the hull. This next attack provides a polynomial solver for PCE for any fixed hull dimension, but is exponential in the hull dimension. As we mention above, the hull dimension of a random $[n,k]$-linear code is typically a small constant, meaning this attack efficiently solves PCE unless we limit ourselves to a subset of codes with large hull dimension, such as weakly self-dual codes.

We say a function $\mathfrak{S}\colon \mathfrak{C}_{n,k,q} \times \{1,\dots,n\} \to T$ mapping to some set T, where $\mathfrak{C}_{n,k,q}$ is the set of all $[n,k]$-linear codes over $\mathbb{Z}_q$, is a signature function for a code $\mathfrak{C}$, if for every $i \in \{1,\dots,n\}$ and every $\pi \in \mathcal{P}_n$ we have that $\mathfrak{S}(\mathfrak{C}, i) = \mathfrak{S}(\pi(\mathfrak{C}), \pi(i))$. We say a signature function is fully discriminant, if we also have that $\mathfrak{S}(\mathfrak{C}, i) \neq \mathfrak{S}(\mathfrak{C}, j)$ for $i \neq j$. Clearly, given two permutation equivalent codes $\mathfrak{C}_1$ and $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$ and a fully discriminant signature $\mathfrak{S}$ (for either one) we can recover the permutation π, since $\mathfrak{S}(\mathfrak{C}_1, i) = \mathfrak{S}(\mathfrak{C}_2, j) \iff$

$j = \pi(i)$. If one has such a signature function, the Support Splitting Algorithm (SSA) essentially consists of finding collisions between $\mathfrak{S}(\mathfrak{C}_1, i)$ and $\mathfrak{S}(\mathfrak{C}_2, j)$.

The signature function proposed by Sendrier [Sen00] for use in SSA is based on the hull space of a code, which is founded in his earlier observation [Sen97] that the hull dimension of a random code will be small. In particular, denote by $\mathfrak{C}^i$ the code obtained by taking a code $\mathfrak{C}$ and puncturing it at position $i \in \{1, \ldots, n\}$, i.e. simply forgetting that entry. We recall that for a code $\mathfrak{C}$, its weight enumerator is a bivariate polynomial defined as

$$\omega(\mathfrak{C})(x, y) = \sum_{w=0}^{n} N_w \cdot x^w y^{n-w}.$$

Then the signature function $\mathfrak{S}$ proposed is the following

$$\mathfrak{S}(\mathfrak{C}, i) = \left\{\omega\left(\mathrm{hull}(\mathfrak{C}^i)\right), \omega\left(\mathrm{hull}((\mathfrak{C}^\perp)^i)\right)\right\}.$$

Since computing the hull of a code is efficient, requiring only linear algebra with a cost of about $\mathcal{O}(n^\omega)$ operations in $\mathbb{Z}_q$ for ω the linear algebra constant, the computational bottleneck is the weight enumerator function, as it usually entails enumerating all of the codewords. Denoting by h the dimension of the hull of an $[n, k]$-linear code, one can estimate [BBPS21] a cost of $\mathcal{O}\big(n \cdot q^h\big)$ for each computation of the weight enumerator. Again observing heuristically that using around $\ln(n)$ punctures is enough to obtain a fully discriminant signature for a code, the complexity estimate of the SSA algorithm is the following.

Proposition 6 ([BBPS21, Prop. 7]). *Let $\mathfrak{C}_1$ be a random $[n, k]$-linear code over $\mathbb{Z}_q$ with hull dimension $h \leq \min\{k, n-k\}$, and let $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$ for a random permutation π. Then the complexity of using SSA to recover π given $\mathfrak{C}_1$ and $\mathfrak{C}_2$ is estimated as*

$$\mathcal{O}\big(n^3 + n^2 \cdot q^h \cdot \ln(n)\big).$$

A.6 The BOS Algorithm

While practically speaking, SSA is a polynomial PCE solver for random codes, it clearly fails when the hull of the code is trivial. In this case, Bardet, Otmani, and Saeed-Taha [BOST19] showed that solving PCE essentially amounts to solving the weighted version of the graph isomorphism problem (WGI). Consider undirected weighted graphs and represent them as they adjacency matrices, i.e. the entry at position (i, j) is equal to ξ if and only if the vertices respectively labeled i and j are connected by a (directed) edge of weight ξ. Two graphs are (permutation) isomorphic if one can be obtained from the other by permuting its vertices while preserving the edge weights. In terms of adjacency matrices, two graphs represented by $\mathbf{A}_1$ and $\mathbf{A}_2$ are isomorphic if and only if there exists a permutation matrix $\mathbf{P}$ such that $\mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^\mathsf{T}$. The main observation of [BOST19] is the following.

Proposition 7 ([BBPS21, Thm. 2]). *Let $\mathfrak{C}_1$ and $\mathfrak{C}_2$ be $[n, k]$-linear codes over $\mathbb{Z}_q$ with trivial hulls. For $i = 1, 2$ denote by $\mathbf{G}_i$ a generator matrix (of columns) for $\mathfrak{C}_i$ and define $\mathbf{A}_i = \mathbf{G}_i \cdot (\mathbf{G}_i \cdot \mathbf{G}_i^\mathsf{T})^{-1} \cdot \mathbf{G}_i^\mathsf{T}$. Then $\mathfrak{C}_1$ and $\mathfrak{C}_2$ are permutation equivalent, i.e. $\mathfrak{C}_2 = \pi(\mathfrak{C}_1)$, if and only if $\mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^\mathsf{T}$ where $\mathbf{P}$ is the permutation matrix associated to π.*

Given that the graph adjacency matrices can be computed in $\mathcal{O}(n^\omega)$ (the cost of matrix multiplication and inversion), the cost of the BOS algorithm is

$$\mathcal{O}(n^\omega \cdot C_{\mathrm{WGI}}(n)),$$

where $C_{\mathrm{WGI}}(n)$ is the cost of solving WGI for graphs with n vertices. Note that WGI can be solved in quasi-polynomial time in the worst case [Bab16]. More generally, if the hull dimension is non-trivial and denoted by h, the complexity of the BOS algorithm is asymptotically estimated as

$$\mathcal{O}\big(h \cdot n^{\omega+h+1} \cdot C_{\mathrm{WGI}}(n)\big).$$

A.7 Summary

We ran the attack estimator provided with [Beu20a] on our parameters, and extended it to also consider the new attack reported in [CPS23,Now24]. We observe that the three collision-based attacks do not impact lattice parameters when run on the parameters of the whole code (the estimator reports upwards of 1700 bits of complexity for each collision-based attack). However, since our codes are not self-orthogonal, one may reduce the complexity of these attacks by consider only the hulls of the instance codes.

Indeed, for two permutation equivalent codes $\mathfrak{C}_1 = \pi(\mathfrak{C}_2)$, we have that $\mathfrak{H}_1 = \pi(\mathfrak{H}_2)$, where $\mathfrak{H}_i = \mathrm{hull}(\mathfrak{C}_i)$ for $i = 1, 2$ [Sen00, Prop. 12]. It is well-known [Sen00] that a generator matrix for $\mathfrak{H}_i$ can be computed efficiently using echelon forms. Since the complexity of ISD depends on the rate $r = \frac{k}{n}$ of the code, the worst case being $r = \frac{1}{2}$, one may therefore reduce the complexity of collision-finding by considering just the hulls of the codes, having a strictly smaller rate $\frac{h}{n} < \frac{k}{n}$.

We observe that the complexity of canonical forms algorithms was always the lowest, leading to the requirement that

$$\frac{1}{2} \cdot \mathrm{H}\left(\frac{h}{n}\right) \cdot n \geq \lambda,$$

where H is the binary entropy function, h, n the dimension and length of the hull, and λ the security parameter. Notice that the complexity of running collision attacks on the hull is indeed lower than on the entire code, since $\mathrm{H}\left(\frac{h}{n}\right) \leq \mathrm{H}\left(\frac{k}{n}\right)$ as soon as $0 < h < k \leq \frac{n}{2}$. We report minimal hull dimensions required for particular security parameters and parameter sets in Table 1. Table 2 compares the hull dimension h required by both hull attacks and hull collision attacks. We observe the latter is always greater for our parameter sizes, thus in our parameter selection in Section 5.2, we consider both the hull collision attack and the usual hull attacks [Sen00,BOST19].

Table 2: Parameters for the given λ and p with $c = 0.25$ and $s = 8$ noting $h_{\mathbf{coll}}$ and $h_{\mathbf{hull}}$ coming from hull collision and the usual hull attacks, respectively.

λ	p	n	k	$\log_2(q)$	$h_{\mathbf{coll}}$	$h_{\mathbf{hull}}$
128	2	7313	450	13	27	10
128	16	11000	550	16	26	10
192	32	20250	900	18	37	14
256	32	29688	1250	19	48	18

B Solving Affine Conics Over $\mathbb{F}_q$

In this section, we discuss solving affine conic equations, and construct a provably uniform sampler for solutions to conic equations using rejection sampling.

B.1 Number of Solutions to Affine Conic Equations

This section summarises [BDR]. An affine conic over a field $\mathbb{F}$ is a quadratic equation in 2 variables of the form

$$F(x_1, x_2) = a\,x_1^2 + b\,x_1 \cdot x_2 + c\,x_2^2 + d\,x_1 + e\,x_2 + f.$$

If $\mathrm{char}(\mathbb{F}) \neq 2$, we may conveniently write the variables, the quadratic form, and the linear form as matrices

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \mathbf{Q} = \begin{bmatrix} a & \frac{b}{2} \\ \frac{b}{2} & c \end{bmatrix}, \qquad \mathbf{l} = \begin{bmatrix} d \\ e \end{bmatrix},$$

and obtain an equivalent formulation of an affine conic

$$F(\mathbf{x}) = \mathbf{x}^{\mathsf{T}} \cdot \mathbf{Q} \cdot \mathbf{x} + \mathbf{l}^{\mathsf{T}} \cdot \mathbf{x} + f.$$

A conic is called *smooth*, or non-degenerate, if its discriminant $D = b^2 - 4 \cdot a \cdot c$ is non-zero. Equivalently, its quadratic form $\mathbf{Q}$ is invertible. From now on let $\mathbb{F} = \mathbb{Z}_q$ for an odd prime q, and let conics be smooth.

It is well-known that since $\mathrm{char}(\mathbb{F}) \neq 2$, any symmetric quadratic form over $\mathbb{F}$ is diagonalisable in the sense of quadratic forms, that is there exists an invertible matrix $\mathbf{A}$ such that $\mathbf{A}^{\mathsf{T}} \cdot \mathbf{Q} \cdot \mathbf{A} = \mathbf{D}$ where $\mathbf{D}$ is a diagonal conjugate to $\mathbf{Q}$. Further, define the vector

$$\mathbf{s} = -\frac{1}{2} \cdot \mathbf{Q}^{-1} \cdot \mathbf{l}.$$

Since $\mathbf{A}$ is invertible, the transformation $\tilde{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{s}$ corresponds to a change a coordinates, hence the number of solutions to $F(\mathbf{x}) = 0$ is equal to the number of solutions of $F(\tilde{\mathbf{x}}) = 0$. Indeed, there is a bijective correspondence between the two solution sets that maps a root $\mathbf{r}$ of $F(\mathbf{x})$ to $\tilde{\mathbf{r}} = \mathbf{A} \cdot \mathbf{r} + \mathbf{s}$ which is a root of $F(\tilde{\mathbf{x}})$, and the map $\mathbf{r} = \mathbf{A}^{-1} \cdot (\tilde{\mathbf{r}} - \mathbf{s})$ that maps back.

Observe that the change of coordinates $\tilde{\mathbf{x}}$ precisely annihilates the linear component, that is

$$\begin{aligned}
&F(\mathbf{A} \cdot \mathbf{x} + \mathbf{s}) \\
&= (\mathbf{A} \cdot \mathbf{x} + \mathbf{s})^{\mathsf{T}} \cdot \mathbf{Q} \cdot (\mathbf{A} \cdot \mathbf{x} + \mathbf{s}) + \mathbf{l} \cdot (\mathbf{A} \cdot \mathbf{x} + \mathbf{s}) + f \\
&= \mathbf{x}^{\mathsf{T}} \cdot \mathbf{A}^{\mathsf{T}} \cdot \mathbf{Q} \cdot \mathbf{A} \cdot \mathbf{x} + 2 \cdot \mathbf{s}^{\mathsf{T}} \cdot \mathbf{Q}^{\mathsf{T}} \cdot \mathbf{A} \cdot \mathbf{x} + \mathbf{l}^{\mathsf{T}} \cdot \mathbf{A} \cdot \mathbf{x} + \mathbf{s}^{\mathsf{T}} \cdot \mathbf{Q} \cdot \mathbf{s} + \mathbf{l}^{\mathsf{T}} \cdot \mathbf{s} + f \\
&= \mathbf{x}^{\mathsf{T}} \cdot \mathbf{D} \cdot \mathbf{x} + \mathbf{s}^{\mathsf{T}} \cdot \mathbf{Q} \cdot \mathbf{s} + \mathbf{l}^{\mathsf{T}} \cdot \mathbf{s} + f \\
&= \mathbf{x}^{\mathsf{T}} \cdot \mathbf{D} \cdot \mathbf{x} + f - \mathbf{s}^{\mathsf{T}} \cdot \mathbf{Q} \cdot \mathbf{s}
\end{aligned}$$

Denote the diagonal entries of $\mathbf{D}$ by d_1 and d_2, the entries of $\tilde{\mathbf{x}}$ by $\tilde{x}_1$ and $\tilde{x}_2$, and the new constant by $f' = f - \mathbf{s}^{\mathsf{T}} \cdot \mathbf{Q} \cdot \mathbf{s}$. We have obtained a new affine conic

$$F(\tilde{x}_1, \tilde{x}_2) = d_1 \cdot \tilde{x}_1^2 + d_2 \cdot \tilde{x}_2^2 + f'$$

whose solutions bijectively correspond to the solutions of $F(x_1, x_2)$ via the above change of coordinates. Recall that since $\mathbf{D}$ is full-rank, we have that $d_1 \cdot d_2 \neq 0$. We have shown that every smooth conic can be equivalent via a change of coordinates to a smooth conic of such form. The number of solutions of an affine conic equation is then characterised completely by the following statement.

Lemma 12 ([BDR, Sec. 2.1]). *Let $F(x_1, x_2) = d_1 \cdot x_1^2 + d_2 \cdot x_2^2 + f'$ with $d_1 \cdot d_2 \neq 0$ be a smooth conic over $\mathbb{Z}_q$ for q odd prime, and denote by $\mathcal{V}_F$ the set of its points $\mathcal{V}_F = \big\{(x_1, x_2) \in \mathbb{Z}_q^2 \,;\, F(x_1, x_2) = 0\big\}$. Then*

$$|\mathcal{V}_F| = \begin{cases} 1 & \text{if } f' = 0 \text{ and } -d_1 \cdot d_2 \notin \mathrm{QR}(\mathbb{Z}_q) \\ q - 1 & \text{if } f' \neq 0 \text{ and } -d_1 \cdot d_2 \in \mathrm{QR}(\mathbb{Z}_q) \\ q + 1 & \text{if } f' \neq 0 \text{ and } -d_1 \cdot d_2 \notin \mathrm{QR}(\mathbb{Z}_q) \\ 2 \cdot q - 1 & \text{if } f' = 0 \text{ and } -d_1 \cdot d_2 \in \mathrm{QR}(\mathbb{Z}_q). \end{cases}$$

Corollary 3. *Let $F(x_1, x_2) = a \cdot x_1^2 + b \cdot x_1 x_2 + c \cdot x_2^2 + d \cdot x_1 + e \cdot x_2 + f$ with $D = b^2 - 4 \cdot a \cdot c \neq 0$ be a smooth conic over $\mathbb{Z}_q$ for q odd prime, and denote by $\mathcal{V}_F$ the set of its points $\mathcal{V}_F = \big\{(x_1, x_2) \in \mathbb{Z}_q^2 \,;\, F(x_1, x_2) = 0\big\}$. Then*

$$|\mathcal{V}_F| = \begin{cases} 1 & \text{if } f' = 0 \text{ and } D \notin \mathrm{QR}(\mathbb{Z}_q) \\ q - 1 & \text{if } f' \neq 0 \text{ and } D \in \mathrm{QR}(\mathbb{Z}_q) \\ q + 1 & \text{if } f' \neq 0 \text{ and } D \notin \mathrm{QR}(\mathbb{Z}_q) \\ 2 \cdot q - 1 & \text{if } f' = 0 \text{ and } D \in \mathrm{QR}(\mathbb{Z}_q), \end{cases}$$

where

$$f' = f + \frac{c \cdot d^2 - b \cdot d \cdot e + a \cdot e^2}{D}.$$

Proof. Since

$$d_{x_1} \cdot d_{x_2} = \det(\mathbf{D}) = \det(\mathbf{A})^2 \cdot (a \cdot c - \frac{b^2}{4})$$

and 4 is always a quadratic residue, we have that checking whether $-d_{x_1} \cdot d_{x_2}$ is a quadratic residue is equivalent to checking whether $D = b^2 - 4 \cdot a \cdot c$ is a quadratic residue.

A quick calculation shows that

$$\mathbf{s}^\mathsf{T} \cdot \mathbf{Q} \cdot \mathbf{s} = \frac{c \cdot d^2 - b \cdot d \cdot e + a \cdot e^2}{4 \cdot a \cdot c - b^2} = \frac{c \cdot d^2 - b \cdot d \cdot e + a \cdot e^2}{-D}.$$

Then we have

$$f' = f - \mathbf{s}^\mathsf{T} \cdot \mathbf{Q} \cdot \mathbf{s}^\mathsf{T} = f + \frac{c \cdot d^2 - b \cdot d \cdot e + a \cdot e^2}{D}.$$ □

B.2 Rejection Sampling for the Conic Solver

We next apply the above observations in the context of the SSO algorithm defined in Definition 10 and its analysis in the proof of Lemma 5. Adopting the notation there, observe that $D = b^2 - 4 \cdot a \cdot c$ being a quadratic residue depends only on the matrix $\mathbf{G}$ which generates the code from which we wish to sample self-orthogonal vectors.[16]

If we sample the first $k-2$ coefficients $c_1, \ldots, c_{k-2}$ uniformly at random from $\mathcal{U}(\mathbb{Z}_q)$, and then a uniformly random solution to the resulting smooth conic, the probabilities sampling each self-orthogonal vector of $\mathrm{Span}(\mathbf{G})$ are not the same. Indeed, if $D \in \mathrm{QR}(\mathbb{Z}_q)$ we have that

- if there are $q-1$ solutions, the probability of each one being sampled is $\frac{1}{q^{k-2}} \cdot \frac{1}{q-1}$, and
- if there are $2 \cdot q - 1$ solutions, the probability of each one being sampled is $\frac{1}{q^{k-2}} \cdot \frac{1}{2 \cdot q - 1}$.

To make the probabilities equal, the sampler must only accept the first case with probability $\frac{q-1}{2 \cdot q - 1}$, and always accept the second case, making the probability of each self-orthogonal vector being sampled exactly $\frac{1}{q^{k-2}} \cdot \frac{1}{2 \cdot q - 1}$.

If $D \notin \mathrm{QR}(\mathbb{Z}_q)$ we have that

- if there is only 1 solution, the probability of it being sampled is $\frac{1}{q^{k-2}}$,
- if there are $q+1$ solutions, the probability of each one being sampled is $\frac{1}{q^{k-2}} \cdot \frac{1}{q+1}$.

To make the probabilities equal, the sampler must only accept the first case with probability $\frac{1}{q+1}$, and always accept the second case, making the probability of each self-orthogonal vector being sampled exactly $\frac{1}{q^{k-2}} \cdot \frac{1}{q+1}$.

The sampler from Figure 3 thus correctly samples from $\mathcal{U}(\mathcal{V}_{q,\mathbf{G}})$. When $D \in \mathrm{QR}(\mathbb{Z}_q)$, the failure probability is bounded from above by $\left(\frac{q}{2 \cdot q - 1}\right)^{\mathsf{halt}}$, and by $\left(\frac{q}{q+1}\right)^{\mathsf{halt}}$ when $D \notin \mathrm{QR}(\mathbb{Z}_q)$. Indeed, rejection can only happen in the case of $q-1$ solutions (resp. 1 solution), which we conservatively bound by 1, and when $u > \frac{q-1}{2 \cdot q - 1}$ (resp. $u > \frac{1}{q+1}$), which happens with probability $\frac{q}{2 \cdot q - 1}$ (resp. $\frac{q-1}{q}$).

Remark 3. We can determine the expected failure probability under the heuristic assumption that for a uniformly random code generator matrix $\mathbf{G}$ we have that f' is a uniformly random field element in $\mathbb{Z}_q$ if $(c_1, \ldots, c_{k-2})$ are uniformly random. The case of $q-1$ solutions (resp. 1 solution) then happens with probability exactly $\frac{q-1}{q}$ (resp. $\frac{1}{q}$), thus

$$\mathbb{E}[\bot \leftarrow \mathsf{SSO}(\mathbf{G}, \mathsf{halt})] = \begin{cases} \left(\frac{(q-1)^2}{2 \cdot q^2 - q}\right)^{\mathsf{halt}} & D \in \mathrm{QR}(\mathbb{Z}_q), \\ \left(\frac{1}{q+1}\right)^{\mathsf{halt}} & D \notin \mathrm{QR}(\mathbb{Z}_q) \end{cases} < \frac{1}{2^{\mathsf{halt}}},$$

[16] There is some choice there since multiple pairs of columns could satisfy the smoothness condition, however once we pick a pair this property is fixed.

over the internal randomness of the algorithm.

It is worth noting that it is **not** true that for any code generator matrix $\mathbf{G}$, the value f' follows the uniform distribution over the randomness of $(c_1, \ldots, c_{k-2})$. This would imply that every code $[n,k]$-linear code over $\mathbb{Z}_q$ has the same number of self-orthogonal codewords, which is not the case. For example, the $[n,2]$-linear code $\mathfrak{C}_{i,j}$ over $\mathbb{Z}_3$ generated by any two standard unit vectors $\mathbf{e}_i, \mathbf{e}_j$ with $i \neq j$ has only one self-orthogonal codeword, the all-zeroes vector $[0]^n$, in other words the inner product on $\mathfrak{C}_{i,j}$ is always non-degenerate. Take for contrast any self-orthogonal $[n,2]$-linear code over $\mathbb{Z}_3$, where every codeword is self-orthogonal, perhaps the $[6,2]$-linear code generated by $\mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3$ and $\mathbf{e}_4 + \mathbf{e}_5 + \mathbf{e}_6$. The above assumption instead states that this holds on average, for a random code generator matrix $\mathbf{G}$.

C Dehulling Linear Codes

In this section, we explicitly write down the algorithms for finding the dual, the hull, and an algebraically complementary 0-hollow subcode to the hull of a linear code. We also sketch out why finding permutation equivalent 0-hollow complementary subspaces to the hulls of the permutation equivalent linear codes is a hard computational problem despite the efficiency of finding an algebraically complementary subspace. We call these two complementary spaces *algebraic* and *geometric dehullings*, respectively.

Let $\mathfrak{C}$ be a $[n,k]$-linear h-hollow code over $\mathbb{Z}_q$ for $h < k$ with hull $\mathfrak{H} \subset \mathfrak{C}$. Algebraically speaking, there exists a 0-hollow subcode $\mathfrak{T} \subset \mathfrak{C}$ of dimension $k-h$ such that $\mathfrak{C} = \mathfrak{H} \oplus \mathfrak{T}$ (that is each $\mathbf{v} \in \mathfrak{C}$ can be written as $\mathbf{v} = \mathbf{h} + \mathbf{t}$ where $\mathbf{h} \in \mathfrak{H}$ and $\mathbf{t} \in \mathfrak{T}$, and $\mathfrak{H} \cap \mathfrak{T} = \{0\}$), and $\mathfrak{T} \cong \mathfrak{C}/\mathfrak{H}$. Here the elements of the quotient space are the equivalence classes of the equivalence relation on $\mathfrak{C}$ defined as $\mathbf{v} \equiv \mathbf{w} \iff \mathbf{v} - \mathbf{w} \in \mathfrak{H}$, and denoted by $\mathbf{v} + \mathfrak{H}$ for any representative $\mathbf{v}$. The code $\mathfrak{T}$ is necessarily 0-hollow, since any vector orthogonal to the entirety of $\mathfrak{T}$ would immediately fall in $\mathfrak{H}$, contradicting the maximality of $\mathfrak{H}$.

Note, however, that this correspondence is not geometric. In an inner product space equipped with a *non-degenerate* bilinear form $\langle \cdot, \cdot \rangle$, the orthogonal complement space $\mathfrak{T}$ would be defined by $\mathfrak{T} = \{\mathbf{v} \in \mathfrak{C} \,;\, \mathbf{v} \perp \mathfrak{H}\}$, and found by considering the left kernel of any generating matrix $\mathbf{H}$ for $\mathfrak{H}$. The inner product on $\mathfrak{C}$ admits many self-orthogonal vectors by Lemma 4, and the hull $\mathfrak{H}$ of $\mathfrak{C}$ is defined precisely as the largest subspace of $\mathfrak{C}$ wrt. inclusion such that $\{\mathbf{v} \in \mathfrak{C} \,;\, \mathbf{v} \perp \mathfrak{H}\} = \mathfrak{C}$, so this approach fails here.

$\mathsf{Dual}(\mathbf{A})$

$\mathbf{A}' \coloneqq$ reduced echelon form of $\mathbf{A}$
$\mathbf{P} \coloneqq$ perm. s.t. $(\mathbf{P} \cdot \mathbf{A}')[1, \ldots, k] = \mathbf{I}_k$
$\mathbf{M} \coloneqq (\mathbf{P} \cdot \mathbf{A}')[k+1, \ldots, n]$
$\mathbf{A}^\perp \coloneqq \mathbf{P}^\mathbf{T} \cdot [-\mathbf{M}^\mathbf{T} || \mathbf{I}_{n-k}]$
return $\mathbf{A}^\perp$

$\mathsf{Hull}(\mathbf{A})$

$\mathbf{M} = [\mathbf{A} | \mathsf{Dual}(\mathbf{A})]$
$\mathbf{H} =$ basis of left kernel of $\mathbf{M}$
return $\mathbf{H}$

$\mathsf{Dehull}(\mathbf{A})$

$\mathbf{H} \coloneqq \mathsf{Hull}(\mathbf{A})$
$\mathbf{B} \coloneqq$ basis of $\mathrm{Span}(\mathbf{A}) / \mathrm{Span}(\mathbf{H})$
$\mathbf{T} \coloneqq$ lift $\mathbf{B}$ to $\mathrm{Span}(\mathbf{A})$
return $\mathbf{T}$

Fig. 11: Algorithms for computing the dual and the hull of a code.

C.1 Geometric Dehulling is Hard

If one could efficiently find 0-hollow subcodes $\mathfrak{T}_i$ such that $\mathfrak{C}_i = \mathfrak{H}_i \oplus \mathfrak{T}_i$ and also $\mathfrak{T}_1 = \pi(\mathfrak{T}_2)$, then [BOST19] can be used to mount an attack against sPCE for $\mathfrak{C}_1, \mathfrak{C}_2$ using the $\mathsf{DehullAttack}$ algorithm from Figure 12. While we can efficiently "dehull" a code algebraically, that is find a 0-hollow $\mathbf{T}$ such that $\mathfrak{C} = \mathrm{hull}(\mathfrak{C}) \oplus \mathrm{Span}(\mathbf{T})$,

finding geometric dehullings of permutation equivalent codes that are permutation equivalent themselves is hard if PCE is hard, as seen in Figure 12.

The main idea of the algebraic dehulling algorithm is to simply construct a vector space basis for $\mathfrak{T} \cong \mathfrak{C}/\operatorname{hull}(\mathfrak{C})$ from distinct equivalence classes to get a subcode of $\mathfrak{C}$ of dimension $n-k$. One may optionally randomise to get a random basis. The algorithm is described in Figure 11, with a reduction sketch from sPCE to the problem of finding geometric dehullings in Figure 12.

$\mathsf{DehullAttack}(\mathbf{A}, \mathbf{B})$

// $\mathfrak{C}_1 = \mathrm{Span}(\mathbf{A})$ and $\mathfrak{C}_2 = \mathrm{Span}(\mathbf{B})$

$\mathbf{H}_1 \coloneqq \mathsf{Hull}(\mathbf{A})$; $\mathbf{H}_2 \coloneqq \mathsf{Hull}(\mathbf{B})$

$\mathbf{T}_1, \mathbf{T}_2 \leftarrow$ find complements to $\mathbf{H}_1, \mathbf{H}_2$ such that $\mathrm{Span}(\mathbf{T}_1) = \pi(\mathrm{Span}(\mathbf{T}_2))$

$\mathit{\Pi} \coloneqq \{\pi' \ ; \ \mathrm{Span}(\mathbf{T}_1) = \pi'(\mathrm{Span}(\mathbf{T}_2))\}$ using [BOST19]

for $\pi' \in \mathit{\Pi}$ **if** $\mathfrak{C}_1 = \pi'(\mathfrak{C}_2)$: **return** π'

return $\perp$

Fig. 12: A potential dehull attack on PCE with the geometric dehulling step marked in gray.

D Code for Parameter Selection

The script is also attached.

```
from dataclasses import dataclass
from estimator.estimator import LWE, ND
from sage.all import log, ceil, floor, sqrt, RR
import sys
import os


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

    def __exit__(self, exc_type, exc_val, exc_tb):
        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):
    def entropy(x):
        return RR(-x * log(x, 2) - (1 - x) * log(1 - x, 2))

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

    # hull collision attacks min h
    h_collision = k
    for h in range(1, k + 1):
        cf = CF_attack(n, h, 2**log_q)
        if cf >= secpar:
            h_collision = h
            break
    # hull attacks min h
    h_hull = max(ceil(secpar / log_q), ceil(secpar / log(n, 2)))
    return max(h_collision, h_hull)


@dataclass
class UPKEParams:
    """
    All solvers return an instance of this class.
    """

    secpar: int
    k: int
    n: int
    log_q: int
    p: int

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

    def ct(self):
        return _kb((self.n + 1) * self.log_q)

    def update(self):
        return _kb(self.secpar / floor(log(self.p, 2)) * (self.n + 1) * self.log_q)

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

    def display(self):
        print(f"{self}: h = {self.hulldim()}, |ct| = {self.ct()}, |upd| = {self.update()}")


def upke_params(secpar=128, c=0.25, sigma=3.2, p=32, lwe_kwds=None):
```

```
    """
    Estimate UPKE parameters and sizes.
    """
    for k in range(300, 1500, 50):
        # LWE correctness condition
        for log_q in range(10, 30):
            if 2**log_q / sqrt(log_q) >= p**2 / 2 + p * sigma * sqrt((secpar + 1) * (1 + c) * k):
                break

        # LHL condition
        # NOTE ignoring - secpar/log(n)
        n = ceil((1 + c) * k * log_q)

        lwe = LWE.Parameters(
            n=k,
            m=n,
            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_hyrbid", "bdd_mitm_hybrid")
        with HiddenPrints():
            if lwe_kwds is None:
                lwe_kwds = {}
            costs = LWE.estimate(lwe, deny_list=deny_list, **lwe_kwds)

        if min(cost["rop"] for cost in costs.values()) > 2**secpar:
            upke = UPKEParams(secpar, k, n, log_q, p)
            return upke


def table1():
    """
    [EC:AlbBenLai25]: Table 1
    """
    rows = [
        {"secpar": 128, "p": 2},
        {"secpar": 128, "p": 16},
        {"secpar": 192, "p": 32},
        {"secpar": 256, "p": 32},
    ]
    for row in rows:
        upke = upke_params(secpar=row["secpar"], p=row["p"])
        upke.display()
```