Cryptanalysis of rank-2 module-LIP: a single real embedding is all it takes

Bill Allombert[1], Alice Pellet-Mary[1], and Wessel van Woerden[1,2]

[1] Univ. Bordeaux, CNRS, Inria, Bordeaux INP, IMB, Talence, France
bill.allombert@math.u-bordeaux.fr,
alice.pellet-mary@math.u-bordeaux.fr
[2] PQShield
wessel.vanwoerden@pqshield.com

Abstract. The rank-2 module-LIP problem was introduced in cryptography by (Ducas, Postlethwaite, Pulles, van Woerden, Asiacrypt 2022), to construct the highly performant HAWK scheme. A first cryptanalytic work by (Mureau, Pellet–Mary, Pliatsok, Wallet, Eurocrypt 2024) showed a heuristic polynomial time attack against the rank-2 module-LIP problem over *totally real* number fields. While mathematically interesting, this attack focuses on number fields that are not relevant for cryptography. The main families of fields used in cryptography are the highly predominant cyclotomic fields (used for instance in the HAWK scheme), as well as the NTRU Prime fields, used for instance in the eponymous NTRU Prime scheme (Bernstein, Chuengsatiansup, Lange, van Vredendaal, SAC 2017).

In this work, we generalize the attack of Mureau et al. against rank-2 module-LIP to the family of all number fields with at least one real embedding, which contains the NTRU Prime fields. We present three variants of our attack, firstly a heuristic one that runs in quantum polynomial time. Secondly, under the extra assumption that the defining polynomial of K has a 2-transitive Galois group (which is the case for the NTRU Prime fields), we give a provable attack that runs in quantum polynomial time. And thirdly, with the same 2-transitivity assumption we give a heuristic attack that runs in classical polynomial time. For the latter we use a generalization of the Gentry–Szydlo algorithm to any number field which might be of independent interest.

1 Introduction

The recent introduction of the Lattice Isomorphism Problem (LIP) as a new hardness assumption in cryptography [2, 11, 12] has lead to an exciting new line of cryptanalysis [7, 10, 14, 22, 23, 25]. LIP asks if two lattices are isomorphic, i.e., if one is an orthonormal transformation of the other. On the constructive side the *module*-lattice isomorphism problem (module-LIP) has been introduced to increase the efficiency of LIP-based schemes. In particular, the highly performant scheme HAWK [11] relies on the hardness of module-LIP. And while LIP has a

long history of cryptanalytic effort, the hardness of the module variant, especially for modules of low rank, is yet unclear.

Let us consider the case of rank-r module lattices over the ring of integers $\mathcal{O}_K$ of a number field K. The rank-r module-LIP problem asks, given two rank-r $\mathcal{O}_K$-modules, to determine whether there exists a K-*linear* orthonormal transformation mapping one of the module onto the other one. Just as for the Shortest Vector Problem (SVP), the rank $r = 1$ case of *ideal* lattices is considered insecure. Namely, the Gentry–Szydlo algorithm can (depending on the number field) recover the isomorphism in classical polynomial time [17]. Therefore, schemes like HAWK propose the usage of modules of rank 2. Recently however, it was shown that rank-2 module-LIP can heuristically be solved in classical polynomial time when the number field is totally real [25], i.e., if all its complex embeddings are real. This is achieved by reducing the initial problem to an instance of the principal ideal problem over a degree-2 extension of K, and applying the Gentry–Szydlo algorithm in this extension [17]. This attack did not affect HAWK as it is instantiated over the *totally imaginary* cyclotomic field $\mathbb{Q}[x]/(x^n + 1)$, where n is a power of two. In fact, a similar approach as in the totally real case here reduces the rank-2 module-LIP problem in cyclotomic fields to an instance of the principal ideal problem [7, 14, 23], but in a quaternion algebra over the maximal totally real subfield of K. Quaternion algebras are significantly harder to handle than number fields (in particular, they are non-commutative), and it is unclear so far if the Gentry–Szydlo algorithm can be generalized to this setting.

Given that for rank-2 modules the totally real case is easy, and the totally imaginary case is seemingly unaffected, a natural question is what happens for intermediate fields. Can rank-2 module-LIP be secure if there is a mixture of real and complex embeddings? For example, the number field $K = \mathbb{Q}[x]/(x^p - x - 1)$ used in NTRU Prime [3] has one real and $p-1$ complex embeddings. This field is one of the only non-cyclotomic field used in cryptographic constructions. Hence, it is natural to wonder whether the rank-2 module-LIP problem would be as secure over this class of number fields, as it seems to be over cyclotomic fields.

In this work we answer this question to the negative. We show that rank-2 module-LIP is heuristically easy once the underlying number field has at least one real embedding. Note that this is always the case if the number field has odd degree.

1.1 Contributions

We describe three attacks against rank-2 module-LIP, when the number field K has at least one real embedding. The three attacks share the same core idea, but the end of the attacks differs, which has an impact on some properties of the attacks (heuristic/provable, classical/quantum, more restriction on the number fields). Our three algorithms and their properties are summarized in Fig. 1 below.

The first attack we present is a heuristic polynomial-time quantum algorithm solving rank-2 module-LIP in case the number field K has at least one real embedding. The heuristic depends on the geometric behavior of the Log-unit

lattice if we scale up one coordinate, and we experimentally verify this heuristic to be true for NTRU Prime fields.

Secondly, we present a fully provable polynomial-time quantum algorithm if we require in addition that the Galois group $\mathrm{Gal}(\Phi)$ of a defining polynomial Φ of K acts 2-transitively on its roots. This is a relatively weak assumption as generally most number fields of degree n have $\mathrm{Gal}(\Phi) \cong S_n$ [9], the symmetric group on a set of size n, which is even n-transitive. In particular this is the case for NTRU Prime fields.

Lastly, we give a heuristic polynomial-time classical algorithm, again with the additional requirement that $\mathrm{Gal}(\Phi)$ is 2-transitive. The heuristic here is a light number theoretic assumption that the field $K(i)$ is what we call a *Gentry–Szydlo friendly* field (see Definition 1). We conjecture that this is true for all fields, and we verify it experimentally for cyclotomic fields, a collection of random fields and for NTRU Prime fields K.

As a side contribution, we provide a generalization of the Gentry–Szydlo algorithm which heuristically works in *any* number field. So far, the Gentry–Szydlo algorithm was only known to work in cyclotomic fields [17] and in CM fields (a larger class of fields containing cyclotomic fields) [21]. We believe that this generalization to any number field can be of independent interest.

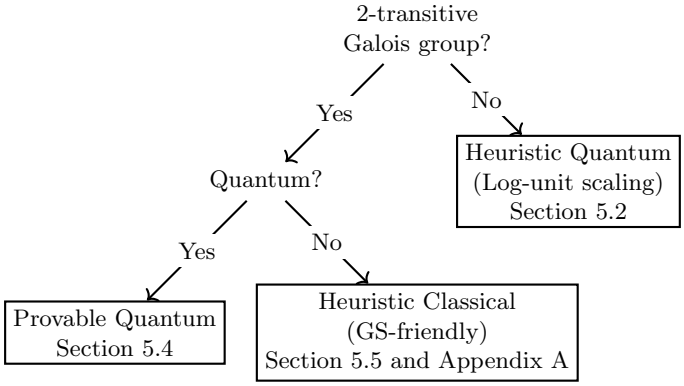

Fig. 1: Overview of the three algorithms described in this article for solving rank-2 module-LIP. They all require that the number field K has at least one real embedding, and they all run in polynomial time.

1.2 Technical overview

Let K be any number field, and $\mathcal{M} \subset \mathcal{O}_K^2$ be a (free) rank-2 module with public basis B_0 and secret basis B. The rank-2 module-LIP problem asks to recover the secret basis B from the knowledge of B_0 and $G = B^* \cdot B$ (the Gram matrix associated to the basis B). Here, $B^* = \overline{B}^\top$ is the transpose conjugate of the matrix

B. One should note that conjugation is not always well defined over a number field K. If K is totally real, then conjugation is simply the identity. If K is a CM number field (i.e., a totally imaginary number field which is a degree 2 extension of a totally real field; for instance a cyclotomic field), complex conjugation is still well defined over K. However, for number fields that are neither CM nor totally real, there exists no complex conjugation in K. To circumvent this issue, we will work in a larger set named $K_{\mathbb{R}}$. This set $K_{\mathbb{R}}$, which will be formally defined in preliminaries, is a ring containing K. In this ring, complex conjugation is always well defined, and so there exists a subring $\overline{K}$ of $K_{\mathbb{R}}$ consisting of all complex conjugates of elements of K. The matrix $G = B^*B$ can then be defined for any field K, albeit in the subring $K\overline{K}$ of $K_{\mathbb{R}}$, and not directly in the field K.

Similarities and differences with the totally real case. Let $B = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right) \in \mathcal{O}_K^{2\times 2}$ be the secret matrix we want to recover, and let $G = B^*B = \left(\begin{smallmatrix} g_1 & g_2 \\ g_3 & g_4 \end{smallmatrix}\right)$ be the public Gram matrix. One of the key point of the algorithm from [25], solving rank-2 module-LIP in totally real number fields, is the simple observation that $g_1 = \overline{a}a + \overline{b}b = a^2 + b^2$ is a sum of two squares in $\mathcal{O}_K$. Note that here, we used the fact that the field is totally real to argue that $\overline{a} = a$ and $\overline{b} = b$.

When the field K is not totally real, this is not the case anymore. Yet, one of our key result is to observe that when K has at least one real embedding $\sigma_1 : K \to \mathbb{R}$, then one can efficiently reconstruct the element $a^2 + b^2$ from the element $\overline{a}a + \overline{b}b$. Indeed, when evaluating the real embedding σ_1 at the element $\overline{a}a + \overline{b}b$, one obtains

$$\sigma_1(\overline{a}a + \overline{b}b) = \overline{\sigma_1(a)} \cdot \sigma_1(a) + \overline{\sigma_1(b)} \cdot \sigma_1(b) = \sigma_1(a)^2 + \sigma_1(b)^2 = \sigma_1(a^2 + b^2),$$

where the second equality comes from the fact that $\sigma_1(a)$ and $\sigma_1(b)$ are in $\mathbb{R}$. Moreover, it is known (see, e.g. [26, Lemma 2.5]) that knowing a single embedding of some element $x \in K$ with enough precision is sufficient to recover the element x exactly. Hence, from $\sigma_1(\overline{a}a+\overline{b}b) = \sigma_1(a^2+b^2)$, one can reconstruct $a^2+b^2 \in \mathcal{O}_K$.[3] More generally, one can reconstruct the matrix $B^\top B \in \mathcal{O}_K^{2\times 2}$ from the knowledge of $G = B^*B$.

We are then back to a situation similar to the totally real case: we know the element $q_1 := a^2 + b^2 \in \mathcal{O}_K$ for some secret $(a, b) \in \mathcal{O}_K^2$, and we want to recover a and b. In the totally real case, the algorithm from [25] then proceeds by constructing the field $K(i)$, and considering the secret element $z = a+ib \in K(i)$. This element z has a so-called relative norm $\mathcal{N}_{K(i)/K}(z) = (a + ib) \cdot (a - ib) = a^2 + b^2 = q_1$ which is known. Hence, recovering z (or equivalently a and b) amounts to solving a relative norm equation in the extension $K(i)/K$. To do so, one first constructs the ideal $I = z\mathcal{O}_{K(i)}$ from the knowledge of q_1.[4] Then, it

[3] Note that $\overline{a}a + \overline{b}b$ is in $K\overline{K}$ but not in K, so the fact that $\sigma_1(\overline{a}a + \overline{b}b) = \sigma_1(a^2 + b^2)$ does not contradict the injectivity of σ_1 on K.

[4] A better strategy using the knowledge of the full matrix $B^\top B$ to construct I is proposed in the recent work [7]. This allows to avoid the rerandomization procedure from [25] and the restriction to a certain class of modules. We use this improved strategy in our work.

remains to call the Gentry–Szydlo algorithm on input I and $q_1 = \mathcal{N}_{K(i)/K}(z)$ to recover z (up to a root of unity) in classical polynomial time [17]. Note that the Gentry–Szydlo algorithm applies here because K is totally real, which implies that $\mathcal{N}_{K(i)/K}(z) = (a+ib)(a-ib) = (a+ib)\overline{(a+ib)} = z\overline{z}$ (where in the second equality we used the fact that $\overline{a} = a$ and $\overline{b} = b$).

Almost all steps of this strategy can be adapted to the case of a number field K with at least one real embedding, once the matrix $B^\top B$ is known: we can create the field $K(i)$, consider the secret element $z = a + ib$, construct the ideal $I = z\mathcal{O}_{K(i)}$ from public information, and we have the equality $q_1 = \mathcal{N}_{K(i)/K}(z)$. The main difficulty arises in the last step: from the knowledge of $z\mathcal{O}_{K(i)}$ and $\mathcal{N}_{K(i)/K}(z)$, we have no efficient algorithm to recover z.[5] Even worse, we have infinitely many solutions $\tilde{z} \in \mathcal{O}_{K(i)}$ satisfying $\tilde{z}\mathcal{O}_{K(i)} = I$ and $\mathcal{N}_{K(i)/K}(\tilde{z}) = q_1$, whereas in the totally real case we had only a small finite number of solutions.

Recovering z from I and q_1. The last technical question that remains to be solved is then: how do we recover z from $I = z\mathcal{O}_{K(i)}$ and $q_1 = \mathcal{N}_{K(i)/K}(z)$, when we have infinitely many solutions satisfying these two equations. This is a time to look back an remember that we were also given the element $\overline{a}a + \overline{b}b \in K\overline{K}$. Using this element, together with the element $a^2 + b^2$, we show that one can reconstruct some coefficients of the vector $(|\sigma_j(z)|)_j$ exactly (and others up to pairwise swaps), where σ_j ranges over all complex embeddings of the field $K(i)$. From there, we propose three different strategies to recover z, leading to our three algorithms.

The first strategy uses that we know at least one coefficient $|\sigma_{j_0}(z)|$, and we can try to use this information to recover z exactly from I. Using a quantum computer, one can reduce the problem of recovering z given $z\mathcal{O}_{K(i)}$ and $|\sigma_{j_0}(z)|$ (up to a root of unity) to the problem of recovering a vector $\boldsymbol{v}$ in the so-called Log-unit lattice with a given value for v_{j_0}, its j_0-th coordinate. Our heuristic is that the map sending vectors of the Log-unit lattice to their j_0-th coordinate is injective, and that by scaling this coordinate by a large factor, one can efficiently invert this map using the LLL algorithm. We verify this heuristic experimentally when the field K is an NTRU Prime field. This gives us a first algorithm recovering z, which is heuristic and quantum polynomial time.

For the second and third algorithm, we work a little bit more, and we show that if the Galois group of Φ is 2-transitive, then one can recover the full vector $(|\sigma_j(z)|)_j$ exactly (and not only a few coordinates or up to pairwise swaps). Knowing $|\sigma_j(z)|$ for all the complex embeddings of $K(i)$ allows to improve the previous strategy: we now have to recover a vector $\boldsymbol{v}$ in the Log-unit lattice, knowing the value of *all* its coordinates. This is simple linear algebra, and we

[5] There are two limitations for applying the Gentry–Szydlo algorithm as in the totally real case. A first limitation, which can be easily circumvented (see Appendix A), is the fact that the field $K(i)$ is not CM anymore. The second and main limitation is that q_1 is not equal anymore to $z\overline{z}$, which is a crucial quantity needed by the algorithm.

do not need a heuristic anymore. This gives us a provable quantum polynomial time algorithm.

Finally, the last algorithm uses the fact that knowing $(|\sigma_j(z)|)_j$ means that we know $z\overline{z} \in K(i)\overline{K(i)}$, in addition to the ideal $I = z\mathcal{O}_{K(i)}$. Recovering z from these two quantities is exactly what the Gentry–Szydlo algorithm does in cyclotomic fields (and what its generalization to CM fields by Lenstra and Silverberg [21] does). The (minor) issue here is that $K(i)$ is not a CM field, hence, we generalized the Gentry–Szydlo algorithm to all number fields, and used it to conclude our third algorithm. Our generalization of the Gentry–Szydlo algorithm requires the existence of some prime numbers with good properties with respect to the field $K(i)$. We call number fields for which such good primes exist *Gentry–Szydlo friendly*. We conjecture that any number field is Gentry–Szydlo friendly (and we verify it experimentally for various families of number fields, including for NTRU Prime fields K), yet we are unable to prove it. This is why we overall obtain a *heuristic* classical polynomial time algorithm for rank-2 module-LIP.

1.3 Related works and discussion

It is interesting to observe that, with our current knowledge, the hardness of the rank-2 module-LIP problem seems to depend on the choice of the number field K. When K is a totally real number field, the work of Mureau at al. [25] showed that the rank-2 module-LIP problem was solvable heuristically in polynomial time. This, however, has little impact on cryptography since totally real number fields are not used in actual cryptographic constructions: the two main families of fields used in lattice-based cryptography are cyclotomic fields, and NTRU Prime fields. Our work extends the result of [25] to all number fields with at least one real embedding. This shows in particular that instantiating the rank-2 module-LIP problem over NTRU Prime fields would be insecure.[6]

Regarding cyclotomic fields, three recent concurrent works [7, 14, 23] tried to generalize the result of [25] to CM fields, which includes cyclotomic fields. Interestingly though, none of these articles managed to provide a polynomial time attack against rank-2 module-LIP in cyclotomic fields. The three works reduce the rank-2 module-LIP problem over CM fields to another problem which might be easier, but they are unable to conclude the attack and prove that the problem they reduce to is indeed easier that the problem they started from. So far, the best known algorithm solving rank-2 module-LIP in cyclotomic fields hasn't changed, and still consists in ignoring the module structure and running standard unstructured LIP-solving algorithms.

To the best of our knowledge, this is the first time that we are faced with an algorithmic problem where NTRU Prime fields seem to lead to weaker instances than cyclotomic fields. It would be interesting to investigate further this

[6] We are not aware of any construction based on rank-2 module-LIP in NTRU Prime fields, so our attack does not break any concrete scheme. But it is a warning against using NTRU Prime fields when relying on the hardness of module-LIP.

discrepancy: can this weakness of NTRU Prime fields be exploited also for other algorithmic problems? Or, on the other hand, is it only a matter of time before someone manages to efficiently solve rank-2 module-LIP over cyclotomic fields?

1.4 Experimental code and data

The code used to generate the data for the figures in this work is made available online[7] and will be submitted to the call for artifacts of Eurocrypt 2025. All the data to generate the plots has also been added there.

Acknowledgments

All the authors were supported by the CHARM ANR-NSF grant (ANR-21-CE94-0003). A. Pellet-Mary was supported by the PEPR quantique France 2030 programme (ANR-22-PETQ-0008) and by the TOTORO ANR grant (ANR-23-CE48-0002). Most of the work on this paper was executed while W. van Woerden was under employment at IMB in Bordeaux. The authors would like to thank Guilhem Mureau, for pointing us to the notion of exponent of a group G.

2 Preliminaries

2.1 Notation

For a polynomial $\Phi = \sum_{j=0}^{k} a_j X^j$ we denote $\|\Phi\| := \|(a_0, \ldots, a_k)\|$ for any norm $\|.\|$. Vectors $\boldsymbol{v} \in R^k$ are *column* vectors and are denoted in lower-case bold. For a finite multiplicative group G and $x \in G$, we let $o(x)$ denote the order of x in G. We also let $o_{\max}(G)$ be the smallest positive integer such that $x^{o_{\max}(G)} = 1$ for all $x \in G$ (this is known as the *exponent* of the group G). Equivalently, $o_{\max}(G)$ is the least common multiple of all the $o(x)$ for $x \in G$.

2.2 Precision

In this work we require the approximate representation of real numbers by floating point numbers. This approximation can have a certain precision λ, where a higher precision gives a better approximation. We discuss here some properties of such approximate representations and computations with them.

Floating point numbers generally consist of a base b, a precision λ and an exponent range $(e_{\min}, e_{\max})$. For this work we fix $b = 2$ and ignore any issues regarding the exponent and assume these have an infinite range. For a real number $y \in \mathbb{R}$ we write its floating-point approximation by $\hat{y} \in \mathbb{R}$. For a floating-point approximation with precision λ we assume the guarantee that

$$|y - \hat{y}| \leq 2^{-\lambda-1} \cdot |y|.$$

[7] Available at `https://github.com/WvanWoerden/modLIP_real_embedding`.

In other words, we assume that the relative error is bounded by $2^{-\lambda-1}$. Absolute errors play no role by our assumption that the exponent range is unlimited.

For basic operations or functions $f : \mathbb{R}^k \to \mathbb{R}$ we furthermore assume that on input floating point numbers $\hat{x}_1, \ldots, \hat{x}_k$ with precision λ we get a floating point result $\hat{y} \in \mathbb{R}$ that satisfies

$$|f(\hat{x}_1, \ldots, \hat{x}_k) - \hat{y}| \leq 2^{-\lambda-1} \cdot |f(\hat{x}_1, \ldots, \hat{x}_k)|.$$

These guarantees are given for most functions by most arbitrary precision floating point implementations.

2.3 Lattices

A lattice $\mathcal{L} \subset \mathbb{R}^n$ is a discrete subgroup of $\mathbb{R}^n$. We call the dimension k of $\mathrm{span}_{\mathbb{R}}(\mathcal{L}) \subset \mathbb{R}^n$ the *rank* of the lattice. We will also consider lattices in $\mathbb{C}^n$ simply by identifying $\mathbb{C}^n \cong \mathbb{R}^{2n}$. Any lattice can be represented as $\mathcal{L} = B \cdot \mathbb{Z}^k$ by a basis $B \in \mathbb{R}^{n \times k}$ consisting of k linearly independent columns, where k is its rank. We call $G = B^\top B = (\langle \boldsymbol{b}_j, \boldsymbol{b}_l \rangle)_{j,l}$ the *gram* matrix of a basis B of a lattice $\mathcal{L}$. The *volume* $\mathrm{vol}(\mathcal{L})$ of a lattice $\mathcal{L}$ is given by $\mathrm{vol}(\mathcal{L}) := \sqrt{\det(B^\top B)}$ for any basis, which is independent of the chosen basis. A lattice has a minimum distance $\lambda_1(\mathcal{L})$ between any two distinct lattice point, by linearity this is equivalent to the length of the shortest nonzero vector, i.e., $\lambda_1(\mathcal{L}) := \min_{\boldsymbol{v} \in \mathcal{L} \setminus \{0\}} \|v\|$.

Given a target $\boldsymbol{t} \in \mathbb{R}^n$ close to the lattice, the LLL algorithm and Babai's nearest plane algorithm can recover the closest lattice point in polynomial time.

Lemma 1 (LLL+Babai). *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice. Let $\boldsymbol{a} \in \mathbb{R}^n$ be such that $\|\boldsymbol{a}\| < 2^{-n} \cdot \lambda_1(\mathcal{L})$. There exists an algorithm that given (a basis of) $\mathcal{L}$ and any target element $\boldsymbol{t} \in \boldsymbol{a} + \mathcal{L}$, recovers $\boldsymbol{a}$ in polynomial time.*

Proof. Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ be an LLL-reduced basis of $\mathcal{L}$ which can efficiently be obtained from any other basis. This basis has Gram-Schmidt vectors $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*$ which satisfy $\|\boldsymbol{b}_j^*\| \geq 2^{1-j} \cdot \|\boldsymbol{b}_1\|$. Babai nearest plane algorithm decodes a target $\boldsymbol{t} \in \boldsymbol{a} + \mathcal{L}$ correctly if $\|\boldsymbol{a}\| < \frac{1}{2} \min_j \|\boldsymbol{b}_j^*\|$. But indeed $\|\boldsymbol{a}\| < 2^{-n} \cdot \lambda_1(\mathcal{L}) \leq 2^{-n} \|\boldsymbol{b}_1\| \leq \frac{1}{2} \min_j \|\boldsymbol{b}_j^*\|$.

2.4 Number fields

Notations and basic results. Let K be any number field of degree n and discriminant Δ_K. When $|\Delta_K|$ tends to infinity, it holds that $n = O(\log |\Delta_K|)$ [24, pp. 261-264], hence, many of our complexity statements will involve $\log |\Delta_K|$ but not n. We let $\mu(K)$ be the group of roots of unity of K. We know that $|\mu(K)| \leq 2n^2$, and that $\mu(K)$ can be computed in polynomial time [25, Corollary 2.11]. We let $\boldsymbol{\sigma} : K \to \mathbb{C}^n$ represents the canonical embedding of K, i.e., it is the concatenation $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ of the r_1 real and $2r_2$ complex embeddings $\sigma_j : K \to \mathbb{C}$. In this work we will fix some ordering such that $\sigma_1, \ldots, \sigma_{r_1}$ are the

real embeddings, and $\sigma_{r_1+j}, \sigma_{r_1+r_2+j} = \overline{\sigma_{r_1+j}}$ are the conjugate pairs of complex embeddings, for $1 \leq j \leq r_2$. Let $\mathcal{O}_K$ be the ring of integers of K. The image $\boldsymbol{\sigma}(\mathcal{O}_K)$ of $\mathcal{O}_K$ by the canonical embedding is a rank-n lattices included in $\mathbb{C}^n$. We assume that we are given a $\mathbb{Z}$-basis $(o_1, \ldots, o_n)$ of $\mathcal{O}_K$, which is LLL-reduced for the canonical embedding (i.e., $(\boldsymbol{\sigma}(o_1), \ldots, \boldsymbol{\sigma}(o_n))$ is an LLL-reduced basis of the lattice $\boldsymbol{\sigma}(\mathcal{O}_K)$).

For any $x \in K$, we let $\mathcal{N}(x) = \prod_j \sigma_j(x)$ denote the algebraic norm of x. By the inequality of arithmetic and geometric means, we know that $\|\boldsymbol{\sigma}(x)\| \geq \sqrt{n} \cdot \mathcal{N}(x)^{1/n}$ for all $x \in K$. In particular, since any $x \in \mathcal{O}_K$ satisfies $\mathcal{N}(x) \geq 1$, it holds that $\lambda_1(\boldsymbol{\sigma}(\mathcal{O}_K)) \geq \sqrt{n}$.

Both in our algorithms and the final step of the adapted Gentry–Szydlo algorithm will require to factor a polynomial over K. This can be performed in polynomial time, for instance thanks to the following lemma.

Lemma 2 (Factoring polynomials in K [1]). *There is a polynomial time algorithm that, given a number field K and a polynomial P in $K[X]$, factors P in $K[X]$.*

Defining complex conjugation in K. Unless K is totally real or a CM field there is not a natural definition of complex conjugation within K. In particular, for an element $a \in K$ there does not necessarily exists an element $\overline{a} \in K$ such that $\sigma_j(\overline{a}) = \overline{\sigma_j(a)}$ for all complex embeddings σ_j. We resolve that here by extending K to a larger ring that does include conjugate elements.

We let $K_{\mathbb{R}} = \mathrm{Span}_{\mathbb{R}}(\boldsymbol{\sigma}(K))$ be the $\mathbb{R}$-vector subspace of $\mathbb{C}^n$ spanned by the image of K via the canonical embedding. This is a dimension n vector space, with basis $(\boldsymbol{\sigma}(o_1), \ldots, \boldsymbol{\sigma}(o_n))$. We also have the following explicit definition $K_{\mathbb{R}} = \{\boldsymbol{x} \in \mathbb{C}^n \,|\, x_j \in \mathbb{R} \text{ for } 1 \leq j \leq r_1,\, x_{j+r_2} = \overline{x_j} \text{ for } r_1 < j \leq r_1 + r_2\}$, where r_1 is the number of real embeddings of K and r_2 is the number of pairs of complex embeddings. We view $K_{\mathbb{R}}$ as a ring, where addition and multiplication are performed coordinate-wise. We also define complex conjugation over $K_{\mathbb{R}}$, which is also defined coordinate-wise: if $\boldsymbol{x} = (x_1, \ldots, x_n) \in K_{\mathbb{R}}$, then $\overline{\boldsymbol{x}} := (\overline{x_1}, \ldots, \overline{x_n})$.

We let $\boldsymbol{\sigma}(K)$ denote the image of K in $K_{\mathbb{R}}$ via the canonical embedding. This is a field for the addition and multiplication of $K_{\mathbb{R}}$ (since σ is a ring homomorphism). We also define $\overline{\boldsymbol{\sigma}(K)} := \{\overline{\boldsymbol{x}} \,|\, \boldsymbol{x} \in \boldsymbol{\sigma}(K)\}$. This is a $\mathbb{Q}$-vector space of dimension n with basis $(\overline{\boldsymbol{\sigma}(o_1)}, \ldots, \overline{\boldsymbol{\sigma}(o_n)})$, and also a field. Finally, we define $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)} := \{\sum_{j=1}^{r} \boldsymbol{x}_j \boldsymbol{y}_j \in K_{\mathbb{R}} \,|\, r \geq 1,\, \boldsymbol{x}_j \in \boldsymbol{\sigma}(K),\, \boldsymbol{y}_j \in \overline{\boldsymbol{\sigma}(K)}\}$, the smallest subring of $K_{\mathbb{R}}$ containing both $\boldsymbol{\sigma}(K)$ and $\overline{\boldsymbol{\sigma}(K)}$. This is a ring by definition, and also a $\mathbb{Q}$-vector space of dimension $\leq n^2$, generated by the elements $(\boldsymbol{\sigma}(o_k)\overline{\boldsymbol{\sigma}(o_l)})_{1 \leq k,l \leq n}$. We note that $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ is not necessarily a field, but the so-called *elementary products*, i.e., the elements of the form $\boldsymbol{x} \cdot \boldsymbol{y}$ with $\boldsymbol{x} \in \boldsymbol{\sigma}(K)$ and $\boldsymbol{y} \in \overline{\boldsymbol{\sigma}(K)}$, are always invertible in $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ (with inverse $\boldsymbol{x}^{-1} \cdot \boldsymbol{y}^{-1}$).

Representation of objects and computation. Recall that we assumed that an LLL-reduced (for the canonical embedding) $\mathbb{Z}$-basis $O = (o_1, \ldots, o_n)$ of $\mathcal{O}_K$

has been fixed. All the objects will be represented with respected to this fixed basis. The elements of K are represented by their vector of coefficients in the basis O, i.e., $x \in K$ is represented by the vector $(x_1, \ldots, x_n) \in \mathbb{Q}^n$ such that $x = \sum_j x_j o_j$. We call the size of x the quantity $\mathrm{size}(x) := \sum_j \mathrm{size}(x_j)$, where the size of a rational number is the sum of the bit-length of its (coprime) numerator and denominator. Similarly, elements of $\boldsymbol{\sigma}(K)$ (resp. $\overline{\boldsymbol{\sigma}(K)}$) are represented by the vector of their rational coefficients in the basis $(\boldsymbol{\sigma}(o_1), \ldots, \boldsymbol{\sigma}(o_n))$ (resp. $(\overline{\boldsymbol{\sigma}(o_1)}, \ldots, \overline{\boldsymbol{\sigma}(o_n)})$), and the size of these elements corresponds to the size of their vector of rational coefficients. For $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$, we fix a $\mathbb{Q}$-basis of this vector space by taking a subset of appropriate size of the generating set $(\boldsymbol{\sigma}(o_k)\overline{\boldsymbol{\sigma}(o_l)})_{1 \leq k,l \leq n}$. Again, vectors of $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ are represented by the vector of dimension $\leq n^2$ of their rational coefficients in this basis. Given such a representation of an element x in $\boldsymbol{\sigma}(K), \overline{\boldsymbol{\sigma}(K)}$ or $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ we can efficiently compute any precision $\lambda = \mathrm{poly}(n)$ floating-point representation $\hat{\boldsymbol{x}}$ in $K_{\mathbb{R}}$ (i.e., we can compute floating-point representations of the coordinates of the vector in $\mathbb{C}^n$).

Since O is LLL-reduced for the canonical embedding, it holds that $\mathrm{size}(x) = \mathrm{poly}(\log \|\boldsymbol{\sigma}(x)\|, n)$ for any $x \in \mathcal{O}_K$ [15, Lemma 2]. Reciprocally, using the fact that $\|\lambda_n(\boldsymbol{\sigma}(\mathcal{O}_K))\| = O(\Delta_K^{1/n})$ [4, Theorem 1.6], we see that $\|\boldsymbol{\sigma}(o_k)\| = \mathrm{poly}(2^n, |\Delta_K|^{1/n})$ and so $\log \|\boldsymbol{\sigma}(x)\| = \mathrm{poly}(\mathrm{size}(x), \log |\Delta_K|)$ for all $x \in \mathcal{O}_K$.

Integral ideals of K are represented by a matrix in $\mathbb{Z}^{n \times n}$ corresponding to their HNF-basis when seen as $\mathbb{Z}$-modules and using the fixed basis B of $\mathcal{O}_K$ to get integer coefficients. Fractional ideals are scaled by an integer until they are integral, and are then represented by the corresponding integral ideal together with the scaling factor. The size of a matrix in $\mathbb{Z}^{n \times n}$ is defined as the sum of the bit-length of its coefficients. The size of an integral ideal is the size of the integer matrix representing it, and for a fractional ideal we also add the bit-length of the (integer) scaling denominator. For any $x \in K$, it holds that $\mathrm{size}(x\mathcal{O}_K) = \mathrm{poly}(\mathrm{size}(x), \log |\Delta_K|)$. This is because $xb_1, \ldots, xb_n$ forms a $\mathbb{Z}$-basis of $x\mathcal{O}_K$, and the rational matrix associated to this basis has a size $\mathrm{poly}(\mathrm{size}(x), \log |\Delta_K|)$. Computing the HNF of this matrix cannot increase its size more than polynomially.

Basic operation on elements, such that addition, multiplication and division (restricted to inversion of elementary products for $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$) of elements of K, $\boldsymbol{\sigma}(K)$, $\overline{\boldsymbol{\sigma}(K)}$, and $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ can be performed in polynomial time in their size and in $\log |\Delta_K|$. Basic operations on ideals, such as multiplication and inversion of fractional ideals of K can be performed in polynomial time in their size and in $\log |\Delta_K|$.

Principal ideal problem. Principal ideals $g\mathcal{O}_K \subset \mathcal{O}_K$ play an important role in this work. As discussed above, these ideals are generally represented by their HNF basis and recovering a principal generator of the ideal is classically a hard problem. There does however exist a polynomial-time quantum algorithm to recover a principal generator.

Theorem 1 (Principal ideal recovery [6]). *Let K be a number field, $\mathcal{O}_K$ its ring of integers, and $I \subset \mathcal{O}_K$ a principal $\mathcal{O}_K$-ideal. Given (the HNF basis of) I there exists a quantum algorithm that recovers a principal generator $h \in I$ such that $I = h\mathcal{O}_K$ in polynomial time.*

Log-unit lattice. Let K be a number field of degree n with ring of integers $\mathcal{O}_K$. Let $\sigma_1, \ldots, \sigma_n : K \to \mathbb{C}$ be the complex embeddings of K. We define the *logarithmic embedding* $\mathbf{Log} : K^* \to \mathbb{R}^n$ by

$$\mathbf{Log}(x) := (\log(|\sigma_1(x)|), \ldots, \log(|\sigma_n(x)|)) \quad \text{for all } x \in K^*.$$

Note that this is a group morphism mapping the multiplicative group K^* onto an additive subgroup of $\mathbb{R}^n$.

We denote the *unit group* of $\mathcal{O}_K$ by $\mathcal{O}_K^\times$. This is a multiplicative group, which by the logarithmic embedding maps to an additive discrete sub-group of $\mathbb{R}^n$, i.e., a lattice. We call the lattice $\mathbf{Log}(\mathcal{O}_K^\times)$ the *Log-unit lattice* of K. The Log-unit lattice is useful for relating distinct generators g, h of the same principal ideal $I = g\mathcal{O}_K = h\mathcal{O}_K \subset \mathcal{O}_K$ to each-other. Indeed, in that case $u = gh^{-1} \in \mathcal{O}_K^\times$ and the difference $\mathbf{Log}(g) - \mathbf{Log}(h) = \mathbf{Log}(u)$ lies in the Log-unit lattice.

The Log embedding $\mathcal{O}_K^\times \to \mathbf{Log}(\mathcal{O}_K^\times)$ is not injective, more precisely its kernel consists of all roots of unity y in $\mathcal{O}_K$, as those satisfy $|\sigma_j(y)| = 1$ for all $1 \leq j \leq n$. This implies that from knowing $\mathbf{Log}(u) \in \mathbf{Log}(\mathcal{O}_K^\times)$ one can only uniquely recover u up to a root of unity. Additionally because complex embeddings come in pairs with the same norm the rank of the Log-unit lattice equals $r_1 + r_2 - 1 < n = r_1 + 2r_2$ where r_1 is the number of real and r_2 the number of totally imaginary embeddings.

Computing generators for the unit group, and thereby a basis for the Log-unit lattice can be done in classical sub-exponential time or quantum polynomial time.

Theorem 2 (Basis of Log-unit lattice [5, 13]). *Given a number field K of degree n, there exists a classical sub-exponential time or a quantum polynomial-time algorithm (in $\log|\Delta_K|$) that computes a set S of generators for the unit group $\mathcal{O}_K^\times$.*[8] *Furthermore, one can compute an LLL-reduced basis of the Log-unit lattice $\mathbf{Log}(\mathcal{O}_K^\times)$ up to precision λ in time $\mathrm{poly}(\log|\Delta_K|, \lambda)$.*

Furthermore, the first minimum of the Log-unit lattice cannot be too small.

Lemma 3 (First minimum Log-unit lattice [18]). *Let K be any number field of degree n and let $\mathcal{L}_K := \mathbf{Log}(\mathcal{O}_K^\times) \subset \mathbb{R}^n$ be the Log-unit lattice of K. Then for any $\varepsilon > 0$, the first minimum $\lambda_1(\mathcal{L}_K)$ is lower bounded by $\lambda_1(\mathcal{L}_K) \geq \Omega(n^{-\frac{1}{2}-\varepsilon})$.*

[8] These generators are usually represented in the so-called compact representation, which allows to maintain a representation with bit-length polynomial in $\log|\Delta_K|$, whereas the standard representation as vectors of coefficients in the known basis of $\mathcal{O}_K$ may lead to non-polynomial bit-length.

2.5 Module lattices and the module-lattice isomorphism problem

Free module lattices. Let K be a degree n number field with a ring of integers $\mathcal{O}_K$. A module lattice over $\mathcal{O}_K$ is an $\mathcal{O}_K$-module $\mathcal{M} \subset K^r$ along with an embedding σ into a real or complex vector space. For simplicity and clarity we restrict ourselves to *free* module lattices of rank $r \geq 1$ over $\mathcal{O}_K$ by considering full-rank free modules $\mathcal{M} = B \cdot \mathcal{O}_K^r$ given by some basis $B \in K^{r \times r}$. Usually we rescale the coefficients and assume that $B \in \mathcal{O}_K^{r \times r}$.

As an embedding we consider the canonical embedding $\sigma : K \to \mathbb{C}^n$ and simply extend it to K^r by concatenation, i.e., we define $\boldsymbol{\sigma}(\boldsymbol{x}) = (\boldsymbol{\sigma}(a_1)| \dots |\boldsymbol{\sigma}(a_r)) \in \mathbb{C}^{nr}$ for $\boldsymbol{x} = (a_1, \dots, a_r) \in K^r$. The geometry of the module lattice is then determined via the embedding, i.e., for vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{M}$ their (Hermitian) inner product equals $\langle \boldsymbol{\sigma}(\boldsymbol{x}), \boldsymbol{\sigma}(\boldsymbol{y}) \rangle \in \mathbb{C}$ (where $\langle \cdot, \cdot \rangle$ denotes the standard Hermitian inner product over $\mathbb{C}^{rn}$). Additionally we define the module dot product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_K := \sum_{j=1}^{r} \overline{\boldsymbol{\sigma}(a_j)} \boldsymbol{\sigma}(b_j) \in \boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)} \subset \mathbb{C}^n$ for $\boldsymbol{x}, \boldsymbol{y} \in K^r$. For a basis $B \in K^{r \times r}$ with columns $\boldsymbol{b}_1, \dots, \boldsymbol{b}_r$ we then define the *module Gram matrix* $G := (\langle \boldsymbol{b}_j, \boldsymbol{b}_k \rangle_K)_{j,k} \in (\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)})^{r \times r}$ consisting of all pairwise module dot products. One can view B and G as the structured representation of a full basis and gram matrix of the module lattice embedded in $\mathbb{C}^{nr}$. Note that when K is totally real or a CM field, then $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)} = \boldsymbol{\sigma}(K)$ and thus the module Gram matrix G can simply be represented by elements in K. In this work this will generally not be the case though and we refer back to Section 2.4 on how to represent elements of $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$.

The module-lattice isomorphism problem. Two $\mathbb{Z}$-lattices $\mathcal{L}_1, \mathcal{L}_2 \subset \mathbb{R}^n$ are *isomorphic* if there exists a (linear) isometry $O \in \mathcal{O}_n(\mathbb{R})$ sending $\mathcal{L}_1$ to $\mathcal{L}_2$. In terms of bases $B_1, B_2 \in \mathbb{R}^{n \times n}$ for $\mathcal{L}_1, \mathcal{L}_2$ respectively this implies that there exists an orthonormal transformation (isometry) $O \in \mathcal{O}_n(\mathbb{R})$ and a unimodular (basis transformation) $U \in \mathrm{GL}_n(\mathbb{Z})$ such that $B_2 = OB_1U$. One can rephrase this in terms of their gram matrices $G_j := B_j^\top B_j$ as

$$G_2 = (OB_1U)^\top OB_1U = U^\top B_1^\top (O^\top O) B_1 U = U^\top G_1 U,$$

taking the orthonormal transformation O out of the picture. We can thus focus on recovering U and assume for now that B_1, B_2 are bases of the same lattice. One can then rephrase the LIP problem as: given B_1 and $G_2 = B_2^\top B_2$, recover B_2 up to an *automorphism* $V \in \mathrm{GL}_n(\mathbb{Z})$ such that $V^\top G_2 V = G_2$.

We now generalize this to free $\mathcal{O}_K$-module lattices of rank r as defined in Section 2.5. Let $\mathcal{M}_1, \mathcal{M}_2 \subset \mathcal{O}_K^r$ be two module lattices of rank r over $\mathcal{O}_K$ given by bases $B_1, B_2 \in \mathcal{O}_K^{r \times r}$. Again we look for a K-linear isometry between them and thus we must preserve all pairwise dot products $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_K \in \boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$. Just as before we can rephrase the LIP problem in terms of (module) gram matrices and basis transformations $U \in \mathrm{GL}_n(\mathcal{O}_K)$.

Problem 1 (module-LIP). Let K be a number field. Let $B \in \mathcal{O}_K^r$ be a (secret) basis of a free module lattice $\mathcal{M} = B \cdot \mathcal{O}_K^r$. Given the module gram matrix

$G_1 = \overline{\boldsymbol{\sigma}(B^\top)}\boldsymbol{\sigma}(B)$ and any other basis of $\mathcal{M}$, recover B up to an automorphism $V \in \mathrm{GL}_2(\mathcal{O}_K)$ such that $V^\top G_1 V = G_1$.

In particular, in this work, we only consider the module-LIP problem on rank-2 integer $\mathcal{O}_K$-module lattices.

Problem 2 (rank-2 module-LIP). Let $B = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ with $a, b, c, d \in \mathcal{O}_K$ be a (secret) basis of a module lattice $\mathcal{M} = B\mathcal{O}_K^2$. Given $G := \overline{\boldsymbol{\sigma}(B^\top)}\boldsymbol{\sigma}(B)$ and any basis of $\mathcal{M}$, recover B (up to an automorphism).

Note that the recovery of B is only unique up to automorphisms. For example, if $\mathcal{M} = \mathcal{O}_K^2$, then (a, b) and (c, d) can be swapped to (b, a) and (d, c), or multiplied by a root of unity $(u_1 a, u_2 b), (u_1 c, u_2 d)$ for $u_1, u_2 \in \mu(K)$, without changing G or the generated lattice.

We would like to remark that using fields that are not totally real or a CM field are probably an inefficient choice for constructions based on module-LIP (besides the attacks we present). The gram matrix has elements in $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$, which generally could require a representation of size as large as $\Theta(n^2)$ in the degree n, compared to $\Theta(n)$ for elements in K.

2.6 Our setting: number fields with a real embedding.

In this section we set up the stage and notation for the rest of this work. Our goal is to solve the module-LIP problem for a rank-2 module $\mathcal{M} \subset \mathcal{O}_{K_1}^2$ over a number field K_1 with at least one real embedding. We also ignore the totally real case as it has been treated by [25]. We will define some notations for K_1 and some extension fields that will be used in the later sections.

Let Φ be an irreducible monic polynomial of degree $n = r_1 + 2r_2$ with $r_1 > 0$ real and $r_2 > 0$ pairs of conjugate complex roots. We denote the real roots of Φ in $\mathbb{C}$ by $\alpha_1, \ldots, \alpha_{r_1}$, and the complex roots by $\alpha_{r_1+1}, \ldots, \alpha_{r_1+r_2}, \ldots, \alpha_{r_1+r_2+1}, \ldots, \alpha_{r_1+2r_2}$ where $\alpha_{r_1+r_2+j} = \overline{\alpha_{r_1+j}}$ for $j = 1, \ldots, r_2$.

Without loss of generality we fix the conjugate pair of roots $\alpha_{r_1+1}, \alpha_{r_1+r_2+1} = \overline{\alpha_{r_1+1}}$, and we consider the number field $K_1 := \mathbb{Q}(\alpha_{r_1+1})$. Then K_1 has degree n over $\mathbb{Q}$ and has n complex embeddings $\sigma_1, \ldots, \sigma_n$ given by $\sigma_k : \alpha_{r_1+1} \mapsto \alpha_k$.

Since K_1 has at least one real embedding $\sigma_1 : K_1 \to \mathbb{R}$ we know that $i = \sqrt{-1} \notin K_1$ as otherwise $\sigma_1(i)^2 = \sigma_1(i^2) = -1$ which is impossible for $\sigma_1(i) \in \mathbb{R}$. Let $L_1 := \mathbb{Q}(\alpha_{r_1+1}, i)$. Then $L_1 = K_1(i)$ is a degree-2 extension of K_1, and a degree-$2n$ extension of $\mathbb{Q}$. It has $2n$ complex embeddings $\sigma_{k,\pm}$ given by

$$\begin{aligned} \sigma_{k,+}(x + yi) &= \sigma_k(x) + i \cdot \sigma_k(y) && \text{for } k = 1, \ldots, n \\ \sigma_{k,-}(x + yi) &= \sigma_k(x) - i \cdot \sigma_k(y) && \text{for } k = 1, \ldots, n, \end{aligned}$$

for all $x, y \in K_1$.

The Galois group $\mathrm{Gal}(\Phi)$ of the polynomial Φ is the Galois group of the splitting field $\widetilde{K} := \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ (where $\alpha_1, \ldots, \alpha_n$ are the roots of Φ). In other words, let $\mathrm{Aut}(\widetilde{K})$ be the set of automorphisms of $\widetilde{K}$ (i.e., the set of all $\mathbb{Q}$-linear

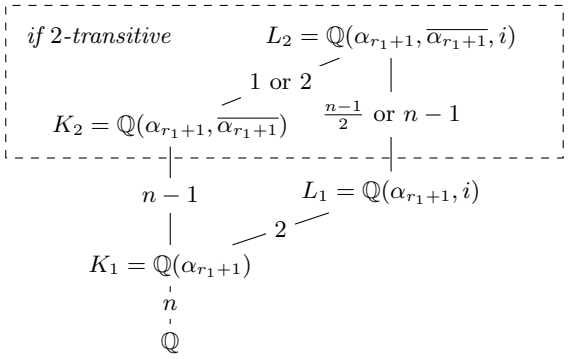

Fig. 2: Diagram of number fields.

bijective field morphisms from $\widetilde{K}$ to $\widetilde{K}$). This group is called the Galois group of $\widetilde{K}$, and this is also, by definition, the Galois group $\mathrm{Gal}(\Phi)$ of Φ. Let $\varphi \in \mathrm{Gal}(\Phi)$ be an element of the Galois group of Φ. As we have seen, $\varphi : \widetilde{K} \to \widetilde{K}$ is an automorphism of $\widetilde{K}$. In particular, for any root α_j of Φ, the image $\varphi(\alpha_j)$ must be another root of Φ. Hence, we can make the Galois group $\mathrm{Gal}(\Phi)$ act on the set of roots of Φ via the action $(\varphi, \alpha_j) \in \mathrm{Gal}(\Phi) \times \{\text{roots of } \Phi\} \mapsto \varphi(\alpha_j) \in \{\text{roots of } \Phi\}$. We say that $\mathrm{Gal}(\Phi)$ acts k-transitively on the roots of Φ (for some $k \in \{1, \ldots, n\}$) if, for any two k-tuples $(\beta_1, \ldots, \beta_k)$ and $(\beta'_1, \ldots, \beta'_k)$ of roots of Φ, where the roots are distinct within each tuple, there exists $\varphi \in \mathrm{Gal}(\Phi)$ such that $\varphi(\beta_j) = \beta'_j$ for all $1 \leq j \leq k$.

To define the remaining extensions K_2 and L_2 we assume that $\mathrm{Gal}(\Phi)$ acts 2-transitively on the roots $\alpha_1, \ldots, \alpha_n$. This will be a requirement for two out of three of our attacks. This assumption does not exclude many number fields. In fact (for a certain notion of randomness) most number fields of degree n have a Galois group isomorphic with the full permutation group S_n [9], which is k-transitive for all $k = 1, \ldots, n$.

Let $K_2 = \mathbb{Q}(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}})$. By the 2-transitivity this is a degree-$n(n-1)$ extension of $\mathbb{Q}$ (because it admits $n(n-1)$ complex embeddings, sending $(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}})$ to any pair of distinct roots of Φ), and so a degree-$(n-1)$ extension of K_1. As mentioned before we have $n(n-1)$ complex embeddings $\sigma_{k,l}$ for $k, l = 1, \ldots, n$ with $k \neq l$ generated by

$$\sigma_{k,l}(\alpha_{r_1+1}) = \alpha_k \in \mathbb{C} \text{ and } \sigma_{k,l}(\overline{\alpha_{r_1+1}}) = \alpha_l \in \mathbb{C}.$$

In particular we will be interested in the embeddings $\sigma_{r_1+j, r_1+r_2+j}$ which map the conjugate pair $(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}})$ to the conjugate pair $(\alpha_{r_1+j}, \overline{\alpha_{r_1+j}})$ for $j = 1, \ldots, r_2$. Additionally, we consider the map $\phi : K_2 \mapsto K_2$ which maps $\alpha_{r_1+1} \mapsto \overline{\alpha_{r_1+1}}$ and $\overline{\alpha_{r_1+1}} \mapsto \alpha_{r_1+1}$ as the natural conjugation map on K_2.

Lastly, let $L_2 := \mathbb{Q}(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}}, i)$. In case $i \in K_2$ we have $K_2 = L_2$ and these number fields collide. Otherwise, L_2 is a degree-2 extension of K_2 and a degree-$(n-1)$ extension of L_1. Then L_2 has $2n(n-1)$ complex embeddings $\sigma_{k,l,\pm}$

for $k, l = 1, \ldots, n$ distinct and $\pm \in \{+, -\}$, given by

$$\sigma_{k,l,\pm}(x + iy) = \sigma_{k,l}(x) \pm i \cdot \sigma_{k,l}(y),$$

for all $x, y \in K_2$.

NTRU Prime fields. As a concrete example throughout this work we consider the NTRU Prime field K_1 with defining polynomial $\Phi = x^p - x - 1$ where p is an odd prime (these polynomial a-re known as Artin-Schreier polynomials). This field is used in the NTRU Prime key encapsulation mechanism [3], which is why we give it this name. This field has precisely 1 real embedding $\sigma_1 : K \to \mathbb{R}$, and $p - 1$ complex embeddings $\sigma_2, \ldots, \sigma_p$. Using the notation $\alpha_1 \in \mathbb{R}$ and $\alpha_2, \ldots, \alpha_p \in \mathbb{C}$ as before for the roots of Φ we have $K_1 = \mathbb{Q}(\alpha_2)$. Furthermore, we have $\mathcal{O}_K = \mathbb{Z}[\alpha_2]/(\Phi)$ and $\mathrm{Gal}(\Phi) \cong S_p$. The Galois group $\mathrm{Gal}(\Phi)$ acts p-transitively and in particular 2-transitively on the roots.

3 Reconstructing $B^\top B$ from a single real embedding

Recall that we try to recover $B = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ from the Hermitian form $G = \overline{\boldsymbol{\sigma}(B^\top)}\boldsymbol{\sigma}(B)$ and any other basis of $\mathcal{M} = B\mathcal{O}_K^2$. In this section, we show that the Hermitian form $\overline{\boldsymbol{\sigma}(B^\top)}\boldsymbol{\sigma}(B)$ can be transformed into a quadratic form $\boldsymbol{\sigma}(B^\top)\boldsymbol{\sigma}(B) = \boldsymbol{\sigma}(B^\top B)$, by reconstructing $B^\top B$. The key observation to do so is the fact that, for any real embedding, e.g., for $\sigma_1 : K_1 \to \mathbb{R}$ we have $\overline{\sigma_1(B^\top)}\sigma_1(B) = \sigma_1(B^\top)\sigma_1(B) = \sigma_1(B^\top B)$. Knowing the value of $\sigma_1(B^\top B)$ (at sufficient precision) allows us to recover $B^\top B \in K$.

3.1 Reconstruction

The idea is as follows. Each of the embedding maps σ_k is injective on K_1, and therefore in theory one can recover any preimage $y \in K_1$ from its image $\sigma_1(y) \in \mathbb{R}$. In particular, one should recover the $x_j \in \mathbb{Q}$ such that $\sum_{j=1}^{d} x_j \sigma_1(o_j) = \sigma_1(y)$ over the reals. In practice however, one only obtains an approximation of $\sigma_1(y)$ and for precise recovery one needs to assume that y is part of some discrete set and that the x_j are bounded. In our setting the discreteness is obtained from knowing that $y \in \mathcal{O}_{K_1}$ and thus that the x_j we are looking for are integer.

We can now apply a standard trick to find a small integer combination of the $\sigma_1(o_j)$ that is close to (our approximation of) $\sigma_1(y)$. We do this by constructing a basis A as follows where $\tilde{\gamma}$ is an approximation of $\sigma_1(y)$ such that $|\tilde{\gamma} - \sigma_1(y)| \leq 2^{-\lambda}$:

$$A = \begin{pmatrix} 2^\lambda \cdot \tilde{\gamma} & 2^\lambda \cdot \sigma_1(o_1) & \ldots & 2^\lambda \cdot \sigma_1(o_n) \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}.$$

Note that

$$\begin{aligned}\|A \cdot (-1, x_1, \ldots, x_n)\|^2 &= 2^{2\lambda} \cdot \left(\tilde{\gamma} - \sum_j x_j \sigma_1(o_j)\right)^2 + \sum_j x_j^2 \\ &= 2^{2\lambda} \cdot (\tilde{\gamma} - \sigma_1(y))^2 + \sum_j x_j^2 \leq 1 + \sum_j x_j^2,\end{aligned}$$

and thus if the x_j's are small, the solution we are looking for corresponds to a short vector in the lattice generated by the columns of A. Note that this lattice is of rank $n+1$ in $\mathbb{R}^{n+2}$.

The remaining question is how large λ has to be for this solution vector to be sufficiently shorter than all the other vectors in the lattice, such that LLL will recover it in polynomial time. For a full analysis of this we refer to Appendix A of [26]. From the same work we take the following complexity statement.

Lemma 4 (Reconstruction [26, Lemma 2.5]). *Let $O = (o_1, \ldots, o_n)$ be an LLL-reduced $\mathbb{Z}$-basis of $\mathcal{O}_{K_1}$. There exists an algorithm that, given O and $\tilde{\gamma}_1$ such that $|\tilde{\gamma}_1 - \sigma_1(y)| \leq 2^{-\lambda}$ for some $y = \sum_{j=1}^{d} x_j o_j \in \mathcal{O}_{K_1}$ and some*

$$\lambda \geq \mathrm{poly}(\log|\Delta_K|, \log\|\Phi\|, \mathrm{size}(y)),$$

recovers $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ in probabilistic polynomial time with respect to λ.

In the statement above, we simplified the bound on λ from [26], which was originally $\mathrm{poly}(n, \log\max_j\|o_j\|_\infty, \log\|\Phi\|, \mathrm{size}(y))$. The simplification comes from the fact that the basis O is LLL-reduced, and so we have seen in preliminaries that $\log\|o_j\| = \mathrm{poly}(\log|\Delta_K|)$ for all j's. For a more precise expression of the needed precision see [26, Lemma A.10]. Generally, the needed precision when only knowing one embedding is at least $\Omega(n^2)$ in the degree.

3.2 Reconstruction example: NTRU Prime

Let p be an odd prime and let $\Phi = x^p - x - 1$ such that K_1 is an NTRU Prime field. Then $\|\Phi\| = O(1)$, and $\log|\Delta_{K_1}| = \mathrm{poly}(p)$. Now for any $y = \sum_j x_j o_j$ with $|x_j| \leq 2^{O(n)}$ we can efficiently recover x from $\sigma_1(y)$ with precision $\lambda = \mathrm{poly}(n) = \mathrm{poly}(p)$. We will observe that in practice, the precision needs only to be quadratic in p, and the constant in front of p^2 is relatively small, see Fig. 3. The quadratic growth of these experimental results is consistent with the more precise expression from [26, Lemma A.10].

Setup. For the experiment we sample $B = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathcal{O}_{K_1}^{2\times 2}$ in the following way, similar to how HAWK proceeds. First, we sample a, b as polynomials with uniform random ternary coefficients. Then, using a Hermite Normal Form computation we try to find c, d such that we have a full basis of $\mathcal{O}_{K_1}$. If this is not possible we start again by resampling a, b. Lastly, we reduce the size of c, d by a simple

application of Babai's basis rounding algorithm. HAWK uses the better Babai's nearest plane algorithm for this but we ignore this difference for simplicity.

Given $\sigma_1(B^\top B)$ we now try to recover the elements of $B^\top B = \begin{pmatrix} q_1 & q_2 \\ q_3 & q_4 \end{pmatrix}$, where notably $q_3 = q_2$. In terms of size we roughly have $\|q_1\| < \|q_2\| < \|q_4\|$. Given that the recovery of q_j from $\sigma_1(q_j)$ is generally influenced by the norm we run different experiments for q_1, q_2 and q_4. For each experiment we find the needed precision λ to recover q_j in the procedure explained in Section 3.1 by a binary search. We use `fplll` [27] as our LLL implementation. We run the experiment with 128 trials for each prime $30 < p < 300$ and for each coefficient q_1, q_2, q_4.

Results. The experimental results are presented in Fig. 3. Indeed we observe a required precision of $\lambda = \Theta(n^2)$, but with a relatively small constant in front of n^2, making the procedure quite practical even for high degrees. The binary search for each case took between a few seconds to at most a day on a single core for each experiment, and should scale to cryptographic dimensions relatively easily. As expected, we see in Fig. 3 that the required precision scales with the norms of q_j, in particular q_4 requires a higher precision to recover than q_1. We also ran additional experiments where instead of LLL we used BKZ with blocksize 20 and 40. This reduced the required precision by roughly 31% and 38% respectively over all cases.

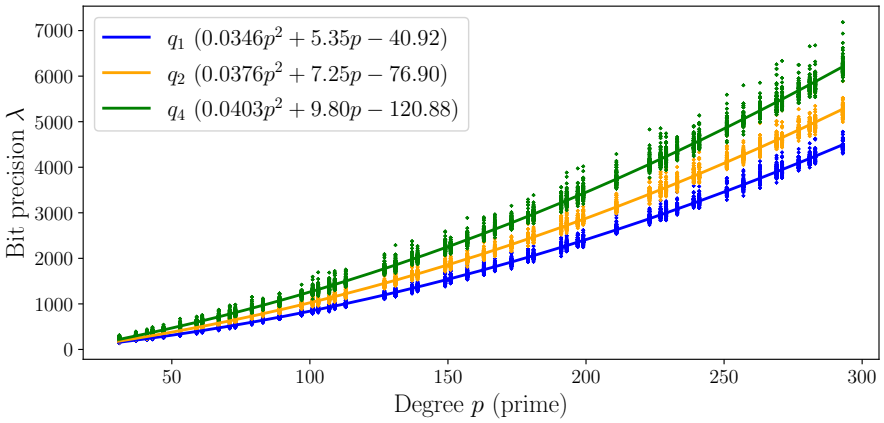

Fig. 3: Required precision λ for reconstruction of $q_1 = a^2 + b^2, q_2 = ac + bd$ and $q_3 = c^2 + d^2$ from $\sigma_1(q_j)$ in the NTRU Prime case. Demonstrates quadratic growth $\lambda = \Theta(p^2)$. The experiment consisted of 128 trials per prime $30 < p < 300$ and coefficient q_1, q_2, q_4.

Precision of LLL. The algorithm behind Lemma 4 runs LLL on a basis with entries of size at least $2^\lambda \geq 2^{\Omega(n^2)}$. Generally, for an integer basis, the size of the entries gives an indication of the required precision needed for floating-point implementations of LLL. It is however not always needed to also run the floating-point LLL algorithm with the same precision. E.g. for the NTRU Prime case experiment in Fig. 3, running `fplll` for all cases $p \leq 300$ required at most 160 bits of precision, indicating more of a linear growth $O(p)$ than quadratic.

One explanation for that is that the basis A with $\lambda = O(n^2)$ has volume $2^{O(n^2)}$, which after LLL reduction implies that all coefficients have size at most $2^{O(n)} \cdot 2^{O(n^2)/n} = 2^{O(n)}$. By increasing λ incrementally alternated by LLL reducing the basis one could even provably keep the coefficients of size $2^{O(n)}$.

Secondly, the incremental nature of `fplll` is helpful, i.e., it LLL reduces the first k rows before considering row $k+1$. In particular, the first k rows form a lattice of determinant roughly $2^{O(n^2)}$, and thus once those k rows are LLL reduced they heuristically have entries bounded by $2^{O(n^2/k)}$. Furthermore the $k+1$-th Gram-Schmidt vector of the initial basis is significantly smaller than that and thus after a single size-reduction step the $(k+1)$-th basis vector also has its entries bounded by $2^{O(n^2/k)}$ before any of the other LLL operations on the first $k+1$ vectors takes place.

4 Ideal recovery

Recall that $L_1 = \mathbb{Q}(\alpha_{r_1+1}, i) = K_1(i)$. In this section we explain how to use $B^\top B$ to recover the ideal $(a+bi)\mathcal{O}_{L_1}$, where (a, b) is the first column of the secret basis B. This follows essentially the ideas from [7, Lemma 3.5, Proposition 3.6]. In Section 5 we will then extract (a, b) from this ideal.

4.1 Computing the ideal

To set things up let $B = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right) \in \mathcal{O}_{K_1}^{2\times 2}$ be the secret basis. From the real embedding we have seen in the previous section how to recover the matrix $Q = B^\top B = \left(\begin{smallmatrix} q_1 & q_2 \\ q_3 & q_4 \end{smallmatrix}\right)$ and thus $q_1 := a^2 + b^2$ and $q_2 := ac + bd$. Generators for the ideal $(a+bi)\mathcal{O}_{L_1}$ now follow from Lemma 5.

Lemma 5 (Adaptation [7, Lemma 3.5, Proposition 3.6]). *Let $z_1 = a+bi$ and $z_2 = c+di$ be elements in $\mathcal{O}_{L_1}$ and $I_\mathcal{M} = z_1\mathcal{O}_{L_1} + z_2\mathcal{O}_{L_1}$ be an integral ideal in $\mathcal{O}_{L_1}$. Then*

$$z_1(\det(B)i + q_2) = q_1 z_2,$$

and $z_1\mathcal{O}_{L_1} = I_\mathcal{M} \cap z_1 z_2^{-1} I_\mathcal{M} = I_\mathcal{M} \cap q_1(\det(B)i + q_2)^{-1} I_\mathcal{M}$.

Proof. For the first part it is easy to verify that

$$\begin{aligned} z_1(\det(B)i + q_2) &= (a+bi)((ad-bc)i + ac + bd) \\ &= a^2c + abd - abd + b^2c + i(a^2d - abc + abc + b^2d) \end{aligned}$$

$$= (a^2 + b^2)c + id(a^2 + b^2) = (a^2 + b^2)(c + id) = q_1 z_2.$$

For the second part note that for any two ideals $I_1, I_2 \subset \mathcal{O}_{L_1}$ we have $(I_1 + I_2)^{-1} = I_1^{-1} \cap I_2^{-1}$. This gives

$$I_{\mathcal{M}}^{-1} = (z_1\mathcal{O}_{L_1} + z_2\mathcal{O}_{L_1})^{-1} = (z_1\mathcal{O}_{L_1})^{-1} \cap (z_2\mathcal{O}_{L_1})^{-1} = z_1^{-1}\mathcal{O}_{L_1} \cap z_2^{-1}\mathcal{O}_{L_1}.$$

To conclude we get

$$z_1\mathcal{O}_{L_1} = z_1 I_{\mathcal{M}}^{-1} I_{\mathcal{M}} = z_1 z_1^{-1} I_{\mathcal{M}} \cap z_1 z_2^{-1} I_{\mathcal{M}} = I_{\mathcal{M}} \cap z_1 z_2^{-1} I_{\mathcal{M}}.$$

The last equality finally follows from the fact that $z_1 z_2^{-1} = q_1(\det(B)i + q_2)^{-1}$, as we have seen before. □

Note that the ideal $I_{\mathcal{M}} = z_1\mathcal{O}_{L_1} + z_2\mathcal{O}_{L_1}$ does not depend on the choice of the basis of $\mathcal{M} = B \cdot \mathcal{O}_{K_1}^2$ we consider. Hence, $I_{\mathcal{M}}$ can be computed in polynomial time from any (public) basis of the underlying module lattice $\mathcal{M}$, which is assumed to be known. For example, in the case that $\mathcal{M} = \mathcal{O}_{K_1}^2$ we simply have $I_{\mathcal{M}} = \mathcal{O}_{L_1}$.

Corollary 1. *Consider the (secret) module basis* $B = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathcal{O}_{K_1}^{2\times 2}$. *There exists a polynomial-time algorithm that given* $\det(B)$, $Q = B^\top B \in \mathcal{O}_{K_1}^{2\times 2}$, *and any public basis of* $\mathcal{M} = B \cdot \mathcal{O}_{K_1}^2$, *recovers the HNF basis of the ideal* $z_1\mathcal{O}_{L_1}$ *where* $z_1 = a + bi$.

Proof. The ideal $I_{\mathcal{M}}$ from Lemma 5 is independent from the choice of the basis. Indeed, let $B' = \left(\begin{smallmatrix} a' & c' \\ b' & d' \end{smallmatrix}\right)$ be the public basis of $\mathcal{M}$ and consider the ideal $I'_{\mathcal{M}} = (a' + b'i)\mathcal{O}_{L_1} + (c' + d'i)\mathcal{O}_{L_1}$. Because $(a, b) \in \mathcal{M}$ there exist $s, t \in \mathcal{O}_{K_1}$ such that $(a, b) = s \cdot (a', b') + t \cdot (c', d')$. But then also $z_1 = a + bi = s \cdot (a' + b'i) + t \cdot (c' + d'i) \in I'_{\mathcal{M}}$. Similarly $z_2 \in I'_{\mathcal{M}}$ and thus $I_{\mathcal{M}} \subset I'_{\mathcal{M}}$, and the other inclusion work similarly by considering the inverse transformation. We can thus compute (the HNF basis of) the ideal $I_{\mathcal{M}}$ in polynomial time from any public basis of $\mathcal{M}$. Following Lemma 5 one then computes $q_1(\det(B)i + q_2)^{-1} I_{\mathcal{M}} \cap I_{\mathcal{M}}$ from the public information, giving the ideal $z_1\mathcal{O}_{L_1}$. These are all standard ideal computations that run in polynomial time. □

Note that the previous corollary requires the knowledge of $\det(B)$. Since we know $Q = B^\top B$, we can compute $\det(Q) = \det(B)^2$, and then compute in polynomial time the roots of the polynomial $X^2 - \det(Q)$ in K_1, which are $\{\det(B), -\det(B)\}$. This allows us to efficiently recover $\det(B)$ up to its sign. In the final attack we can simply try both options and continue the other steps with both results (at least one of which is correct).

4.2 Ideal recovery example: NTRU Prime

We implemented the ideal recovery method from Lemma 5 and Corollary 1 for the NTRU Prime case for several bases B. The implementation was done directly in `PARI/GP` as `sagemath` was significantly slower.

Setup. We aim to verify both the correctness and the running time of the ideal recovery step for the NTRU Prime case. For the experiment we sample a random basis $B \in \mathcal{O}_{K_1}^{2\times 2}$, for which all elements have uniform ternary coefficients in the power basis $1, \alpha_2, \ldots, \alpha_2^{p-1}$. We continue by computing $B^\top B$ and $\det(B)$. Then we confirm that indeed $z_1(\det(B)i + q_2) = q_1 z_2$ as in Lemma 5. To conclude we compute $I_{\mathcal{M}}$ from B and and verify that $z_1\mathcal{O}_{L_1} = I_{\mathcal{M}} \cap q_1(\det(B)i + q_2)^{-1} I_{\mathcal{M}}$. We run the experiment with 128 trials for each prime $30 < p < 200$.

Results. All experiments finished correctly with the runtime shown in Fig. 4. Computing the intersection is the most expensive step, requiring a large Hermite Normal Form computation. All other steps were negligible in comparison.

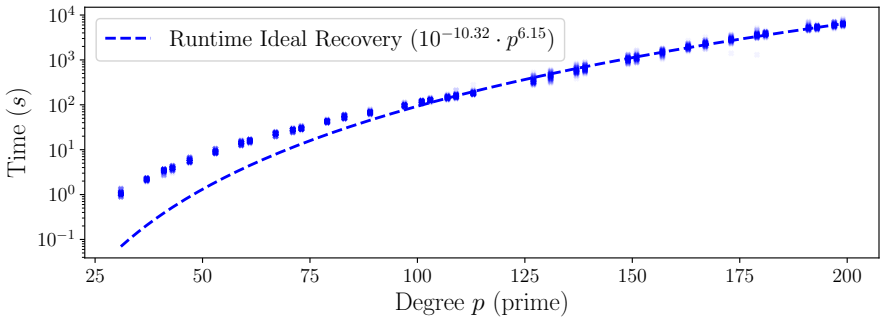

Fig. 4: Runtime for recovery of ideal $(a + bi)\mathcal{O}_{L_1}$ using Corollary 1 in `PARI/GP`. 128 trials per prime on an AMD Zen2 EPYC 7452 @ 2.35 GHz CPU. The fitting was done over the primes $100 < p < 200$ and the log of their runtime.

5 Recovery of z

In this final section we show how to recover $z_1 = a + bi \in \mathcal{O}_{L_1}$ up to a root of unity of L_1. Since the roots of unity of L_1 can be computed in polynomial time and that there are at most poly(n) such elements (see preliminaries), we can then enumerate all possible z_1's, and hence recover the first column of the secret basis B. Doing the same thing for the second column allows to recover the full secret basis.[9] Since we focus here on z_1, let us drop the index and write $z := z_1 = a + bi$. In order to recover z, we will use the knowledge of $a^2 + b^2$ from Section 3, as well as the knowledge of the ideal $z\mathcal{O}_{L_1}$ from Section 4, and the

[9] One can actually save a little bit of time by remembering that in the previous section we showed that $z_1 z_2^{-1}$ was efficiently computable, so once z_1 has been recovered, z_2 can be computed by a single multiplication in L_1.

knowledge of $\overline{\boldsymbol{\sigma}(B)}^{\top}\boldsymbol{\sigma}(B)$ (and in particular its top-left coefficient $\overline{\boldsymbol{\sigma}(a)}\boldsymbol{\sigma}(a) + \overline{\boldsymbol{\sigma}(b)}\boldsymbol{\sigma}(b) \in \boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$) which was the input of the problem.

Using the polynomial-time quantum algorithms from Theorems 1 and 2, we can recover a principal generator g of $z\mathcal{O}_{L_1}$ and a basis of the Log-unit lattice $\mathbf{Log}(\mathcal{O}_{L_1}^{\times})$. We show in Section 5.1 that the knowledge of $a^2 + b^2$ allows us to compute some coefficients of $\mathbf{Log}(z)$ corresponding to the pairs of complex embeddings of L_1 above the real embeddings of K_1. The generators z and g of $z\mathcal{O}_{L_1}$ must differ by some unit u and thus $\mathbf{Log}(z) - \mathbf{Log}(g) = \mathbf{Log}(u)$ lies in the Log-unit lattice. We thus also know some coefficients of $\mathbf{Log}(u)$, which using similar techniques as in Section 3 allows us in Section 5.2 to heuristically recover u and thus z up to a root of unity.

Furthermore, in the case where Φ is 2-transitive, we show in Section 5.3 that we can recover all coefficients of $\mathbf{Log}(z)$, leading to a provable recovery of z up to a root of unity in Section 5.4. In the 2-transitive case we also show in Section 5.5 and Appendix A an adaptation of the Gentry–Szydlo algorithm which heuristically recovers z without the quantum steps in classical polynomial time.

5.1 Partial recovery of Log(z)

Recall that the $2n$ complex embeddings of L_1 are given by $\sigma_{j,\pm} : a + bi \mapsto \sigma_j(a) \pm i \cdot \sigma_j(b)$ for $j = 1, \ldots, n$ and $\pm \in \{+, -\}$. In particular this means for $z = a + bi$ that $|\sigma_{j,\pm}(z)|^2 = \sigma_j(a^2 + b^2)$ for any $j = 1, \ldots, r_1$ (corresponding to a real embedding of K_1) and any sign $\pm \in \{+, -\}$. So given $a^2 + b^2$ we can compute $2r_1$ coefficients of $\mathbf{Log}(z)$. The remaining values $|\sigma_{j,\pm}(z)|$ for the complex embeddings σ_j with $j = r_1 + 1, \ldots, r_1 + 2r_2$ we can only determine up to a pairwise swap, thanks to Lemma 6.

Lemma 6. *Let $a, b \in K_1$ and $z = a + bi \in L_1$. Fix some embedding σ_j of K_1 and define, $\gamma = |\sigma_j(a)|^2 + |\sigma_j(b)|^2$ and $\delta = \sigma_j(a)^2 + \sigma_j(b)^2$. Then the two real roots of the polynomial*

$$f(t) = t^2 - 2\gamma t + |\delta|^2,$$

are $\{|\sigma_{j,+}(z)|^2, |\sigma_{j,-}(z)|^2\}$.

Proof. Let $\sigma_+ := |\sigma_{j,+}(z)|^2$ and $\sigma_- := |\sigma_{j,-}(z)|^2$, and let us write for simplicity $\alpha = \sigma_j(a)$ and $\beta = \sigma_j(b) \in \mathbb{C}$. Then

$$\sigma_{\pm} = (\alpha \pm i\beta)(\overline{\alpha} \mp i\overline{\beta}) = |\alpha|^2 + |\beta|^2 \mp i(\alpha\overline{\beta} - \beta\overline{\alpha}) \in \mathbb{R}.$$

We verify that

$$\begin{aligned}(t - \sigma_+)(t - \sigma_-) &= t^2 - (\sigma_+ + \sigma_-)t + \sigma_+\sigma_- \\ &= t^2 - 2(|\alpha|^2 + |\beta|^2)t + (|\alpha|^2 + |\beta|^2)^2 + (\alpha\overline{\beta} - \beta\overline{\alpha})^2 \\ &= t^2 - 2\gamma t + |\delta|^2.\end{aligned}$$

□

Note that the real numbers γ and δ from the lemma can be computed with arbitrary precision from the knowledge of $\overline{\boldsymbol{\sigma}(a)}\boldsymbol{\sigma}(a) + \overline{\boldsymbol{\sigma}(b)}\boldsymbol{\sigma}(b)$ and $a^2 + b^2$. So one can compute the polynomial $f(t)$ from the lemma in polynomial time, and then compute its roots with arbitrary precision. Hence, we can obtain the values of $|\sigma_{j,\pm}(z)|$ for all embeddings $\sigma_{j,\pm} : L_1 \to \mathbb{C}$ up to pairwise orderings. For each of the real embeddings σ_j where $j = 1, \ldots, r_1$, the two values $|\sigma_{j,+}(z)|$ and $|\sigma_{j,-}(z)|$ coincide and we obtain the corresponding coefficients of $\mathbf{Log}(z)$ precisely as mentioned before.

5.2 Heuristic recovery of z from partial information

Knowing even one coefficients of $\mathbf{Log}(z)$ up to high precision seems in practice enough to reconstruct z. Proving this seems difficult however as it requires certain geometric properties of the (scaled) Log-unit lattice. In 5.3 we therefore assume 2-transitivity of the Galois group to construct $\mathbf{Log}(z)$ fully, leading to a fully provable algorithm. In this section we shortly discuss the heuristic approach that does not require this extra condition.

The reconstruction proceeds similarly to the reconstruction of an element of $\mathcal{O}_{K_1}$ from a single known embedding in Section 3. Here we try to recover $\mathbf{Log}(z)$ from knowing at least one of its coefficients. One embedding is in principle enough to heuristically recover $\mathbf{Log}(z)$ using a generator g of the ideal $z\mathcal{O}_{L_1}$ and a basis of the Log-unit lattice $\mathbf{Log}\,\mathcal{O}_{L_1}^{\times}$. The knowledge of a principal generator g of the ideal $z\mathcal{O}_{L_1}$ fixes the lattice coset $\mathbf{Log}(z)+\mathbf{Log}\,\mathcal{O}_{L_1}^{\times}$. Now if at least one coefficient of $\mathbf{Log}(z)$ is known up to sufficiently high precision we can use the discreteness of the lattice to recover $\mathbf{Log}(z)$. This requires the following geometric scaling property on the Log-unit lattice.

Heuristic 1 (Scaling heuristic). *Let $\mathcal{L}_{L_1} := \mathbf{Log}(\mathcal{O}_{L_1}^{\times}) \subset \mathbb{R}^{2n}$ be the Log-unit lattice of L_1 of rank $n-1$. Let $\mathcal{L}_{L_1}^{\lambda} = \mathrm{diag}(2^{\lambda}, 1, \ldots, 1) \cdot \mathcal{L}_{L_1}$ be the Log-unit lattice where the first coordinate (corresponding to an embedding $\sigma_{1,\pm}$ of L_1 above a real embedding σ_1 of K_1) is scaled up by 2^{λ}. We say that the* scaling heuristic *holds for the Log-unit lattice if*

$$\lambda_1(\mathcal{L}_{L_1}) \geq \Omega(\mathrm{poly}(n)^{-1} \cdot 2^{\lambda/(n-1)}).$$

for increasing $\lambda \geq 1$.

Note that Heuristic 1 essentially says that the first minimum roughly grows in correspondence with the increase in volume of the scaled lattice. In practice we observe an even stronger property, namely the scaled Log-unit lattice behaves like a random lattice and has its first minimum close to the Gaussian Heuristic ($\mathrm{gh}(\mathcal{L}) \approx \sqrt{k/2\pi e} \cdot \mathrm{vol}(\mathcal{L})^{1/k}$ for a lattice of rank k). See e.g. Fig. 5 where we experimentally show this for the NTRU Prime case. With that heuristic in place we can recover vectors of the Log-unit lattice from a single coefficient.

Heuristic Claim 1 (Unit reconstruction) *Let $C = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{n-1})$ be an LLL-reduced basis of $\mathcal{L}_{L_1}$. If Heuristic 1 holds, there exists an algorithm that, given C*

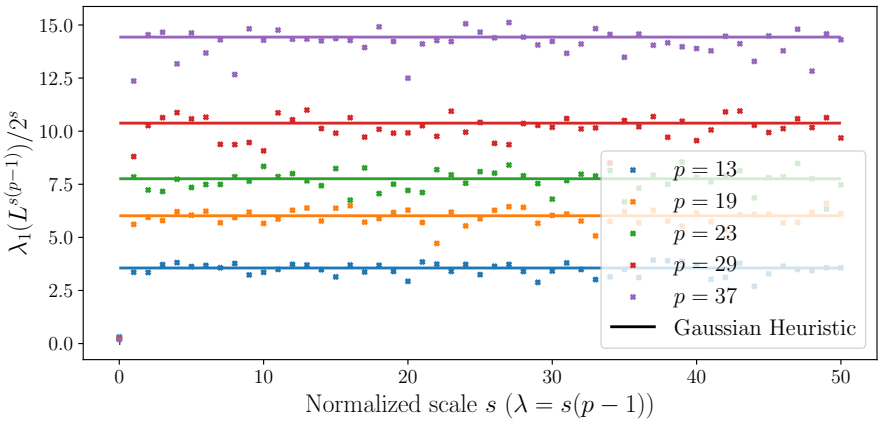

Fig. 5: The normalized first minimum $\lambda_1(\mathcal{L}^\lambda_{L_1})/2^{\lambda/(p-1)}$ of the scaled Log-unit lattice $\mathcal{L}^\lambda_{L_1}$ for the NTRU Prime case. Heuristic 1 states that $\lambda_1(\mathcal{L}^\lambda_{L_1})/2^{\lambda/(p-1)} \geq \Omega(\mathrm{poly}(n)^{-1})$. In the figure we see that in fact a much stronger property $\lambda_1(\mathcal{L}^\lambda_{L_1})/2^{\lambda/(p-1)} \approx \mathrm{gh}(\mathcal{L}_{L_1}) > \Omega(1)$ is satisfied.

and $\tilde{\gamma}_1 \in \mathbb{R}$ *such that* $|\tilde{\gamma}_1 - u_1| \leq 2^{-\lambda} \cdot |u_1|$ *for some* $\boldsymbol{u} = \sum_{j=1}^{n-1} x_j \boldsymbol{c}_j \in \mathbf{Log}(\mathcal{O}^\times_{L_1})$ *and some*

$$\lambda \geq \mathrm{poly}(n, \log\|\boldsymbol{u}\|_2),$$

recovers $\boldsymbol{x} = (x_1, \ldots, x_{n-1}) \in \mathbb{Z}^{n-1}$ *in polynomial time with respect to* λ.

Justification. We consider the scaled lattice $\mathcal{L}^\lambda_{L_1}$. Note that the vector $\boldsymbol{t} := (2^\lambda \tilde{\gamma}_1, 0, \ldots, 0)$ has a close lattice vector $\boldsymbol{u}^\lambda := (2^\lambda u_1, u_2, \ldots, u_{2n}) \in \mathcal{L}^\lambda_{L_1}$ as

$$\|(2^\lambda \tilde{\gamma}_1, 0, \ldots, 0) - (2^\lambda u_1, u_2, \ldots, u_{2n})\|^2 \leq 2^{2\lambda} \cdot |\tilde{\gamma}_1 - u_1|^2 + \|\boldsymbol{u}\|_2^2 \leq 1 + \|\boldsymbol{u}\|_2^2.$$

Under Heuristic 1 we have $\lambda_1(\mathcal{L}^\lambda_{L_1}) \geq 2^{\lambda/(n-1)} \cdot \Omega(\mathrm{poly}(n)^{-1})$, which for some $\lambda = O(n(n + \log\|\boldsymbol{u}\|_2))$ satisfies $\lambda_1(\mathcal{L}^\lambda_{L_1}) > 2^n \cdot \sqrt{(1 + \|\boldsymbol{u}\|_2^2)}$. Lemma 1 then suffices to recover the close vector $\boldsymbol{u}^\lambda = (2^\lambda u_1, u_2, \ldots, u_n)$ from $\boldsymbol{t}$ by an application of LLL and Babai's nearest plane algorithm. Note that here we ignored further rounding errors which can be handled as in the proof of Lemma 4. △

We see from the justification that a precision of $O(n^2)$ is generally sufficient to recover $\mathbf{Log}(z)$ from a single coefficient.

More generally, with a similar heuristic when scaling multiple coefficients, one would require a precision of $O(n^2/k)$ when k coordinates are known. Lemma 9 shows indeed that when all $k = 2n$ coefficients are known a precision of $O(n)$ suffices.

5.3 Full recovery of Log(z) using 2-transitivity

From now on, we assume that the Galois group $\mathrm{Gal}(\Phi)$ acts 2-transitively on the roots of Φ, and we will use this to recover the full vector $\mathbf{Log}(z)$ (or, equivalently, to recover $|\sigma_{j,\pm}(z)|$ for all the embeddings $\sigma_{j,\pm}$ of L_1).

Recall that, due to the 2-transitivity of $\mathrm{Gal}(\Phi)$, $K_2 = \mathbb{Q}(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}})$ has a conjugation automorphism ϕ which swaps α_{r_1+1} and $\overline{\alpha_{r_1+1}}$. This automorphism can be somehow viewed as a complex conjugation over K_2 (even though we do *not* have $\boldsymbol{\sigma}(\phi(x)) = \overline{\boldsymbol{\sigma}(x)}$ for all $x \in K_2$). In particular, we can embed the ring $\boldsymbol{\sigma}(K_1) \cdot \overline{\boldsymbol{\sigma}(K_1)}$ into K_2 via the $\mathbb{Q}$-linear map: $\boldsymbol{\sigma}(b_j)\overline{\boldsymbol{\sigma}(b_k)} \mapsto b_j\phi(b_k)$ (recall that the $(\boldsymbol{\sigma}(b_j)\overline{\boldsymbol{\sigma}(b_k)})_{j,k}$ form a generating set of $\boldsymbol{\sigma}(K_1) \cdot \overline{\boldsymbol{\sigma}(K_1)}$ as a $\mathbb{Q}$-vector space). In particular, from the knowledge of the gram matrix $\overline{\boldsymbol{\sigma}(B)}^T\boldsymbol{\sigma}(B)$, we can compute in polynomial time the element $a\phi(a) + b\phi(b) \in K_2$. Recall also that $L_2 = \mathbb{Q}(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}}, i)$.

Lemma 7. *Let $z_\pm = a \pm ib$ for $a, b \in \mathcal{O}_{K_1}$ and let $\tilde{z}_\pm = \phi(a) \mp i\phi(b) \in \mathcal{O}_{L_2}$. Let $\gamma = a\phi(a) + b\phi(b) \in \mathcal{O}_{K_2}$ and $\delta = a^2 + b^2 \in \mathcal{O}_{K_1}$. Then $z_+\tilde{z}_+, z_-\tilde{z}_- \in L_2$ are the roots of $Q(x) = x^2 - 2\gamma x + \delta\phi(\delta)$ in L_2 and one can recover $\{z_+\tilde{z}_+, z_-\tilde{z}_-\}$ in polynomial time, given γ, δ.*

Proof. To verify that $z_+\tilde{z}_+$ and $z_-\tilde{z}_-$ are the roots of $Q(x)$ note that,

$$\begin{aligned} z_+\tilde{z}_+ \cdot z_-\tilde{z}_- &= (a+ib)(\phi(a) - i\phi(b)) \cdot (a-ib)(\phi(a) + i\phi(b)) \\ &= (a^2+b^2)(\phi(a)^2 + \phi(b)^2) = \delta\phi(\delta), \end{aligned}$$

and

$$\begin{aligned} z_+\tilde{z}_+ + z_-\tilde{z}_- &= (a+ib)(\phi(a) - i\phi(b)) + (a-ib)(\phi(a) + i\phi(b)) \\ &= 2a\phi(a) + 2b\phi(b) = 2\gamma. \end{aligned}$$

Hence, we have that $(x - z_+\tilde{z}_+)(x - z_-\tilde{z}_-) = x^2 - (z_+\tilde{z}_+ + z_-\tilde{z}_-) \cdot x + z_+\tilde{z}_+ \cdot z_-\tilde{z}_- = Q(x)$. We conclude that the roots of Q in L_2 are $\{z_+\tilde{z}_+, z_-\tilde{z}_-\}$, as desired. In order to compute $\{z_+\tilde{z}_+, z_-\tilde{z}_-\}$, it then suffices to construct the polynomial $Q(x)$, which can be done in polynomial time from γ an δ, and then to factor it over L_2, which can also be performed in polynomial time using Lemma 2. □

We do not know which of the roots is $z_+\tilde{z}_+$ but we can simply try both options. For the next Lemma we therefore assume to know $z_+\tilde{z}_+$.

Lemma 8. *Given $z\tilde{z}$ where $z = a+bi$ and $\tilde{z} = \phi(a) - i\phi(b)$ and $q_1 = a^2+b^2$, one can recover all absolute embeddings $|\sigma_{j,\pm}(z)|$ for $j = 1, \ldots, n$ and $\pm \in \{+,-\}$ up to arbitrary precision in polynomial time. In particular one recovers $\mathbf{Log}(z) \in \mathbf{Log}(\mathcal{O}_{L_1})$ up to arbitrary precision.*

Proof. We first consider the case that $[L_2 : K_2] = 2$ and later the special case that $L_2 = K_2(i) = K_2$. Due to this and the 2-transitivity of $\mathrm{Gal}(\Phi)$ we get that $\sigma_{k,l,\pm} : L_2 \to \mathbb{C}$ as defined in Section 2.6 is a complex embedding for all distinct $k, l \in \{1, \ldots, n\}$ and $\pm \in \{+,-\}$. Recall that $\sigma_{k,l,\pm}(\alpha_{r_1+1}) = \alpha_k$,

$\sigma_{k,l,\pm}(\overline{\alpha_{r_1+1}}) = \alpha_l$ and $\sigma_{k,l,\pm}(i) = \pm i$. We consider the case that $\{k,l\} = \{r_1 + j, r_1 + r_2 + j\}$ for $j = 1, \dots, r_2$ such that σ_k and σ_l are conjugate embeddings. Note that

$$\sigma_{k,l,\pm}(z\tilde{z}) = (\sigma_{k,l}(a) \pm i \cdot \sigma_{k,l}(b))(\sigma_{k,l}(\phi(a)) \mp i \cdot \sigma_{k,l}(\phi(b)))$$

Now note that $a \in K_1$ and thus $\sigma_{k,l}(a) = \sigma_k(a)$ (and the same for b). Furthermore since $\phi : \alpha_{r_1+1} \mapsto \overline{\alpha_{r_1+1}}$, it holds that $\sigma_{k,l}(\phi(a)) = \sigma_l(a) = \overline{\sigma_k(a)}$. From this we get that

$$\sigma_{k,l,\pm}(z\tilde{z}) = (\sigma_k(a) \pm i \cdot \sigma_k(b))(\overline{\sigma_k(a)} \mp i \cdot \overline{\sigma_k}(b)) = |\sigma_{k,\pm}(z)|^2.$$

Note that such embeddings can efficiently be computed and constructed. So we can efficiently recover $|\sigma_{k,\pm}(z)|^2$ for $k = r_1 + 1, \dots, r_1 + 2r_2$. For $k = 1, \dots, r_1$ Lemma 6 already suffices.

We now consider the case that $[L_2 : K_2] = 1$. In this case $i \in \mathbb{Q}(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}})$ and thus there exists a rational multivariate polynomial $P \in \mathbb{Q}[X, Y]$ such that $i = P(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}})$. Then, by the 2-transitivity we get that $P(\alpha_k, \alpha_l) = \pm i$ for all roots $k \neq l$. In particular, we can fix the order of the conjugate roots $\alpha_{r_1+j}, \alpha_{r_1+r_2+j}$ such that $P(\alpha_{r_1+j}, \alpha_{r_1+r_2+j}) = i$. Because $L_2 = K_2$ its complex embeddings are given by the $\sigma_{k,l}$ for $k \neq l$. Furthermore, because we fixed the ordering based on the sign of i we know that

$$\sigma_{r_1+j,r_1+r_2+j}(i) = \sigma_{r_1+j,r_1+r_2+j}(P(\alpha_{r_1+1}, \overline{\alpha_{r_1+1}})) = P(\alpha_{r_1+j}, \overline{\alpha_{r_1+j}}) = i,$$

and thus that $\sigma_{r_1+j,r_1+r_2+j} = \sigma_{r_1+j,r_1+r_2+j,+}$. From here we can proceed as before to compute $|\sigma_{k,+}(z)|$. Using that $|\sigma_{k,-}(z)| = |\sigma_k(a^2 + b^2)|/|\sigma_{k,+}(z)|$ and with Lemma 6 we again obtain all values of $|\sigma_{k,\pm}(z)|$. □

5.4 Reconstructing z from $\mathbf{Log}(z)$ and $z\mathcal{O}_{L_1}$

In Section 5.2 we already showed how to heuristically recover z from the ideal $z\mathcal{O}_{L_1}$ and a single coefficient of $\mathbf{Log}(z)$. Then, in the case that $\mathrm{Gal}(\Phi)$ is 2-transitive we have shown in Section 5.3 that we can recover all coefficients of $\mathbf{Log}(z)$. Using this full information we can provable recover z up to a root of unity of L_1, in quantum polynomial time. The reason for that is that we do not have to scale up (one coordinate of) the Log-unit lattice anymore, and therefore we can simply use existing bounds on the first minimum of the Log-unit lattice, dropping the need for Heuristic 1.

Lemma 9. *Let $C = (\boldsymbol{c}_1, \dots, \boldsymbol{c}_{n-1})$ be an LLL-reduced basis of the Log-unit lattice $\mathbf{Log}(\mathcal{O}_{L_1}^\times)$. There exists an algorithm that given C and $\tilde{\boldsymbol{\gamma}} \in \mathbb{R}^{2n}$ such that $|\tilde{\gamma}_j - u_j| \leq 2^{-\lambda} \cdot |u_j|$ for all $1 \leq j \leq 2n$, for some $\boldsymbol{u} = \sum_{j=1}^{n-1} x_j \boldsymbol{c}_j \in \mathbf{Log}(\mathcal{O}_{L_1}^\times)$ and some*

$$\lambda \geq \mathrm{poly}(n, \log \|\boldsymbol{u}\|),$$

recovers $\boldsymbol{x} = (x_1, \dots, x_{n-1}) \in \mathbb{Z}^{n-1}$ in polynomial time with respect to λ.

Proof. The proof follows similarly to the proof of Heuristic Claim 1, but with Lemma 3 instead of Heuristic 1. □

This now allows us to present a full polynomial-time quantum algorithm for rank-2 module-LIP when there is at least one real embedding and if the Galois group of the defining polynomial is 2-transitive.

Theorem 3. *Let $K_1 = \mathbb{Q}[x]/(\Phi)$ be a number field with at least one real embedding and such that* $\mathrm{Gal}(\Phi)$ *acts 2-transitively on the roots of Φ. Then the rank-2 module-LIP problem on $\mathcal{O}_{K_1}$ can be solved in quantum polynomial-time in* $\log|\Delta_{K_1}|$ *and in the input size.*[10]

Proof. Firstly, using Lemma 4 one recovers $B^\top B$. Given $B^\top B$ we recover the ideal $z\mathcal{O}_{L_1}$ where $z = a + bi$ using Corollary 1. Then Lemmas 7 and 8 gives us $\mathbf{Log}(z)$ up to the required precision. The polynomial-time quantum algorithm of Theorem 1 gives us a generator g of $z\mathcal{O}_{L_1}$, and thus there exists a unit $u \in \mathcal{O}_{L_1}^\times$ such that $\mathbf{Log}(u) = \mathbf{Log}(z) - \mathbf{Log}(g)$. The latter can be computed because $\mathbf{Log}(z)$ and g are known. The polynomial-time quantum algorithm of Theorem 2 gives us a basis of the Log-unit lattice $\mathbf{Log}(\mathcal{O}_{L_1}^\times)$. We can then apply Lemma 9 to recover the coefficients of $\mathbf{Log}(u)$ in the known basis of $\mathbf{Log}(\mathcal{O}_{L_1}^\times)$, from a sufficiently good approximation of $\mathbf{Log}(u)$. This, in turn, can be used to recover u up to a root of unity, and therefore $z = ug$ up to a root of unity. The time complexity and required precision at all steps is polynomial in $\log|\Delta_{K_1}|$ and in the size of the module-LIP instance.

5.5 Gentry–Szydlo

Our previous two attacks both involved the Log-unit lattice and used quantum algorithms. Note that without any of the quantum steps we can recover the ideal $z\mathcal{O}_{L_1}$ where $z = a + bi$ by the results in Sections 3 and 4. Additionally, when $\mathrm{Gal}(\Phi)$ is 2-transitive we can also recover $z\tilde{z}$ as defined in Section 5.3. The Gentry–Szydlo algorithm is a classical polynomial-time algorithm that recovers z (up to a root of unity) given such information. The main obstacle for us is that, so far, this algorithm had only been considered in CM-fields, which our number field L_1 is not. In Appendix A we generalize it to any number field that is what we call *Gentry-Szydlo friendly* (or GS-friendly) and Heuristic 2 says that any field is GS-friendly (although we only really need it for L_1). For completeness we informally recall here the definition of a GS-friendly field and the heuristic.

Heuristic 2 (Informal, see Definition 1 and Heuristic 2). *For any number field K of degree n one can efficiently find large primes $p_1, p_2 \in [2^{2n+1}, 2^{2n+2}]$ such that*

$$\gcd\left(o_{\max}((\mathcal{O}_K/p_1\mathcal{O}_K)^\times), o_{\max}((\mathcal{O}_K/p_2\mathcal{O}_K)^\times)\right) = O(\mathrm{poly}(n)).$$

We call such number fields GS-friendly.

[10] Recall that, in this work, our definition of module-LIP is restricted to free modules represented by a basis.

In Fig. 6 we show experimentally that if $K = K_1(i)$ for K_1 an NTRU Prime field, or if K is a cyclotomic fields or a random field (i.e., a field defined by a random irreducible polynomial), then K seems to be GS-friendly. For a more extensive explanation of this heuristic and why it is needed, we refer to Appendix A.3.

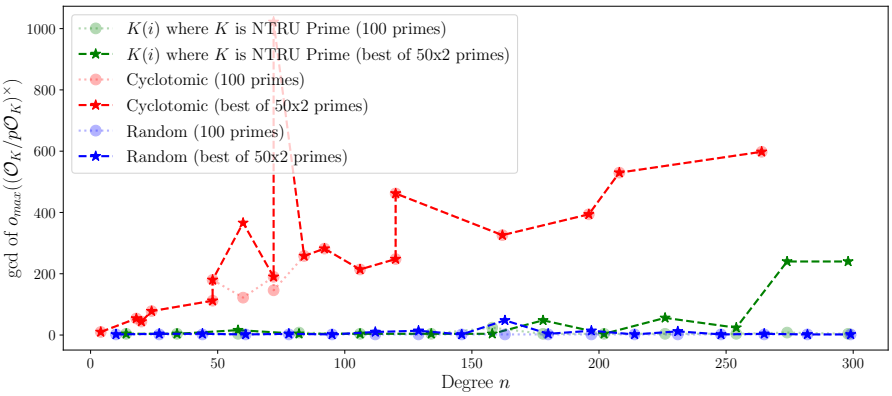

Fig. 6: Plotting the gcd's as a function of the degree for the three families of fields

Under Heuristic 2 we can prove the following generalized version of the Gentry–Szydlo algorithm. We refer to Appendix A for the proof of this heuristic claim, and extensive treaty of this generalization.

Heuristic Claim 2 (Generalized Gentry–Szydlo) *Let K be any number field, and assume that we know an LLL-reduced $\mathbb{Z}$-basis of $\mathcal{O}_K$. Let $g \in \mathcal{O}_K \setminus \{0\}$. Under Heuristic 2, there is a probabilistic polynomial time (in its input size and in $\log|\Delta_K|$) algorithm that takes as input the HNF basis of $g\mathcal{O}_K$ and the element $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)} \in \boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ and outputs $g\varepsilon$ for some $\varepsilon \in \mu(K)$.*

To conclude we can now apply Heuristic Claim 2 to $K = L_1$ using the ideal $z\mathcal{O}_{L_1}$ and the element $\mathbf{Log}(z)$ obtained in Lemma 8, to recover the secret element z up to a root of unity.

Theorem 4. *Let $K_1 = \mathbb{Q}[x]/(\Phi)$ be a number field with at least one real embedding and such that $\mathrm{Gal}(\Phi)$ acts 2-transitively on the roots of Φ. Then there is a heuristic probabilistic polynomial time classical algorithm that solves the rank-2 module-LIP problem*[11] *on $\mathcal{O}_{K_1}$.*

Proof. The algorithm starts by recovering the matrix $B^\top B$ in (classical) polynomial time, using Lemma 4. Then, it recovers the ideal $z\mathcal{O}_{L_1}$ where $z = a + bi$

[11] Recall that our definition of module-LIP is restricted to free modules.

using Corollary 1, as well as the vector $(|\sigma_{j,\pm}(z)|^2)_{1\leq j\leq n,\pm\in\{+,-\}}$ using Lemma 8 (both computation can be performed in classical polynomial time).

Note that the vector $(|\sigma_{j,\pm}(z)|^2)_{1\leq j\leq n,\pm\in\{+,-\}} \in \mathbb{R}^{2n}$ is equal to the vector $\boldsymbol{\sigma}(z)\overline{\boldsymbol{\sigma}(z)} \in \boldsymbol{\sigma}(L_1)\overline{\boldsymbol{\sigma}(L_1)}$. Hence, given enough precision for the real coordinates of $(|\sigma_{j,\pm}(z)|^2)_{1\leq j\leq n,\pm\in\{+,-\}} \in \mathbb{R}^{2n}$, one can recover the element $\boldsymbol{\sigma}(z)\overline{\boldsymbol{\sigma}(z)}$ represented by a vector of rational coordinates in the fixed basis of $\boldsymbol{\sigma}(L_1)\overline{\boldsymbol{\sigma}(L_1)}$. The algorithm then runs the Gentry-Szydlo algorithm on input $z\mathcal{O}_{L_1}$ and $\boldsymbol{\sigma}(z)\overline{\boldsymbol{\sigma}(z)} \in \boldsymbol{\sigma}(L_1)\overline{\boldsymbol{\sigma}(L_1)}$, which outputs in classical polynomial time the element z, up to a root of unity of L_1.

Finally, the algorithm enumerates all the poly(n) candidate z by enumerating all the roots of unity of L_1 (which can be done in polynomial time, see preliminaries). For each choice of $z = z_1 = a' + ib'$, it computes the corresponding $z_2 = c' + id'$ using Lemma 5. It tests whether the matrix $B' = \begin{pmatrix} a' & c' \\ b' & d' \end{pmatrix}$ is a solution to the module-LIP instance, and if it is, it outputs it. We know that at least one of the candidate z will give the matrix B, hence the algorithm must successfully output a solution to the module-LIP instance. □

References

1. Belabas, K., van Hoeij, M., Klüners, J., Steel, A.: Factoring polynomials over global fields. Journal de théorie des nombres de Bordeaux **21**(1), 15–39 (2009)
2. Bennett, H., Ganju, A., Peetathawatchai, P., Stephens-Davidowitz, N.: Just how hard are rotations of $\mathbb{Z}^n$? algorithms and cryptography with the simplest lattice. In: EUROCRYPT 2023: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 252–281. Springer (2023)
3. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU prime: reducing attack surface at low cost. In: Selected Areas in Cryptography–SAC 2017: 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers 24. pp. 235–260. Springer (2018)
4. Bhargava, M., Shankar, A., Taniguchi, T., Thorne, F., Tsimerman, J., Zhao, Y.: Bounds on 2-torsion in class groups of number fields and integral points on elliptic curves. Journal of the American Mathematical Society **33**(4), 1087–1099 (2020)
5. Biasse, J.F., Fieker, C.: Subexponential class group and unit group computation in large degree number fields. LMS Journal of Computation and Mathematics **17**(A), 385–403 (2014)
6. Biasse, J.F., Song, F.: Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In: Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms. pp. 893–902. SIAM (2016)
7. Chevignard, C., Fouque, P.A., Mureau, G., Pellet-Mary, A., Wallet, A.: A reduction from Hawk to the principal ideal problem in a quaternion algebra. Cryptology ePrint Archive (2024)
8. Cohen, H.: A course in computational algebraic number theory, vol. 138. Springer Science & Business Media (2013)
9. Cohen, S.D.: The distribution of galois groups and hilbert's irreducibility theorem. Proceedings of the London Mathematical Society **3**(2), 227–250 (1981)

10. Ducas, L., Gibbons, S.: Hull attacks on the lattice isomorphism problem. In: PKC 2023: IACR International Conference on Public-Key Cryptography. pp. 177–204. Springer (2023)
11. Ducas, L., Postlethwaite, E.W., Pulles, L.N., van Woerden, W.: Hawk: Module LIP makes lattice signatures fast, compact and simple. In: ASIACRYPT 2022: International Conference on the Theory and Application of Cryptology and Information Security. pp. 65–94. Springer (2022)
12. Ducas, L., van Woerden, W.: On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography. In: EUROCRYPT 2022: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 643–673. Springer (2022)
13. Eisenträger, K., Hallgren, S., Kitaev, A., Song, F.: A quantum algorithm for computing the unit group of an arbitrary degree number field. In: Proceedings of the forty-sixth annual ACM symposium on Theory of computing. pp. 293–302 (2014)
14. Espitau, T., Pliatsok, H.: On hermitian decomposition lattices and the module-LIP problem in rank 2. Cryptology ePrint Archive (2024)
15. Fieker, C., Stehlé, D.: Short bases of lattices over number fields. In: International Algorithmic Number Theory Symposium. pp. 157–173. Springer (2010)
16. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32. pp. 1–17. Springer (2013)
17. Gentry, C., Szydlo, M.: Cryptanalysis of the revised NTRU signature scheme. In: Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings 21. pp. 299–320. Springer (2002)
18. Kessler, V.: On the minimum of the unit lattice. Journal de théorie des nombres de Bordeaux **3**(2), 377–380 (1991)
19. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische annalen **261**, 515–534 (1982)
20. Lenstra, H.W., Silverberg, A.: Revisiting the gentry-szydlo algorithm. In: Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34. pp. 280–296. Springer (2014)
21. Lenstra Jr, H.W., Silverberg, A.: Testing isomorphism of lattices over CM-orders. SIAM Journal on Computing **48**(4), 1300–1334 (2019)
22. Ling, C., Liu, J., Mendelsohn, A.: On the spinor genus and the distinguishing lattice isomorphism problem. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 329–358. Springer (2024)
23. Luo, H., Jiang, K., Pan, Y., Wang, A.: Cryptanalysis of rank-2 module-LIP with symplectic automorphisms. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 359–385. Springer (2024)
24. Minkowski, H.: Gesammelte Abhandlungen. Chelsea, New York (1967)
25. Mureau, G., Pellet-Mary, A., Pliatsok, G., Wallet, A.: Cryptanalysis of rank-2 module-LIP in totally real number fields. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 226–255. Springer (2024)
26. Pellet-Mary, A., Stehlé, D.: On the hardness of the NTRU problem. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part I 27. pp. 3–35. Springer (2021)

27. The FPLLL development team: fplll, a lattice reduction library, Version: 5.4.5 (2023), available at `https://github.com/fplll/fplll`

A A Gentry–Szydlo algorithm for any number field

In [17], Gentry and Szydlo described an algorithm that recovers in polynomial time a secret element $g \in \mathcal{O}_K$ for a *cyclotomic field* K, given as input the ideal $g\mathcal{O}_K$ generated by g and the element $g\overline{g} \in K$ (which corresponds the relative norm of g with respect to the maximal totally real subfield of K).[12] This algorithm was then extended by Lenstra and Silverberg, first in [20] to cover a larger class of lattices with symmetries, and later in [21], to cover the case of all *CM number fields* K (as well as the more general case of lattices over a CM order).

In this section, we show that the Gentry–Szydlo algorithm can be extended to *any number field* K, provided that this number field is "Gentry–Szydlo friendly". This condition requires the existence of appropriate primes in the number field, and we argue that all number fields are likely to be Gentry–Szydlo friendly (even though we are not able to formally prove it). We also verify experimentally that cyclotomic fields, $K = K_1(i)$ where K_1 is an NTRU Prime field, and random fields seem Gentry–Szydlo friendly.

The main technicality one faces when extending the Gentry–Szydlo algorithm to all number fields, is that the definition of $\overline{g}$ does not have a natural meaning anymore when K is not CM (CM-fields are by definitions the fields in which a complex-conjugation-like automorphism exists). We suspect that this is the reason why previous works never considered extending the Gentry–Szydlo algorithm to a more general class of fields than CM fields. In an arbitrary number field K, if $g \in K$, there is usually no element $\overline{g} \in K$ satisfying $\boldsymbol{\sigma}(\overline{g}) = \overline{\boldsymbol{\sigma}(g)}$ (where we apply complex-conjugation on every coordinate of the vector on the right hand side of the equality). Yet, such an element exists in $K_{\mathbb{R}}$: for any $\boldsymbol{\sigma}(g) \in \boldsymbol{\sigma}(K) \subset K_{\mathbb{R}}$, one can define $\overline{\boldsymbol{\sigma}(g)} \in K_{\mathbb{R}}$ by taking complex conjugation of every coordinate of the vector. We can then define the set $\overline{\boldsymbol{\sigma}(K)}$ consisting of all the complex conjugates (in $K_{\mathbb{R}}$) of elements of $\boldsymbol{\sigma}(K)$, as well as the set $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$, which is a finite dimensional vector space over $\mathbb{Q}$, and in which the (canonical embedding of the) element $g\overline{g}$ exists. See Section 2.4 for a formal definition of these objects.

With this definition, we can properly define the input and expected output of a generalization of the Gentry–Szydlo algorithm for *any* number field K. We then prove that the original algorithm by Gentry and Szydlo [17] can be easily adapted to this more general context. The proof we provide below follows almost step by step the proof of the Gentry–Szydlo algorithm as described in [16, Section 7.3]. We start by describing the so-called polynomial chain computation in Section A.1. Then we explain how this can be used to recover a small power of the secret g in Section A.2. And we conclude the proof in Section A.4. The main difference between our general case and the cyclotomic case is that we do not have an explicit description of how prime integers split in $\mathcal{O}_K$ (whereas this

[12] The algorithm actually only recovers g up to a root of unity of K, which is the best we can hope for given only $g\mathcal{O}_K$ and $g\overline{g}$. This is usually sufficient for any cryptanalytic application, so we will ignore it in this introductory paragraph.

is well understood for cyclotomic fields). This is why we need to make some assumption on the behavior of such primes: this is the purpose of Section A.3.

A.1 Polynomial chain

The first step of the Gentry–Szydlo algorithm is to compute a so-called polynomial chain. This chain will be used to compute a high power of the secret element g, modulo some prime numbers. The objective of this chain is to maintain a small size of the objects, thanks to successive calls to the LLL algorithm, which avoid explosion of the coefficients. Before computing the polynomial chain, we need the following result, called implicit lattice reduction (see, e.g., [16, Lemma 6]). It allows us to recover a somewhat small multiple of g, given $g\mathcal{O}_K$ and $g\overline{g}$.

Lemma 10. *Let K of degree n and $g \in K \setminus \{0\}$. Given the HNF basis of $g\mathcal{O}_K$ and the element $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)} \in \boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$, we can compute in polynomial time in $\log |\Delta_K|$ and the size of g and element $z \in g\mathcal{O}_K$ such that $z = g \cdot a$ with $a \in \mathcal{O}_K$ satisfying $\|\boldsymbol{\sigma}(a)\| \leq 2^n \cdot \sqrt{n}$.*

Proof. From the knowledge of the HNF basis of $g\mathcal{O}_K$, one can compute in polynomial time $(c_1, \ldots, c_n) \in K^n$ that forms a $\mathbb{Z}$-basis of $g\mathcal{O}_K$. This implies that $(\boldsymbol{\sigma}(c_1), \ldots, \boldsymbol{\sigma}(c_n))$ forms a basis of the lattice $\boldsymbol{\sigma}(g\mathcal{O}_K)$. We will not call the LLL algorithm directly in $\boldsymbol{\sigma}(g\mathcal{O}_K)$ but distort it first (to ensure that we recover a small *multiple* of g in $g\mathcal{O}_K$, and not simply a small element of $g\mathcal{O}_K$).

Let us call $|\boldsymbol{\sigma}(g)| = (|\sigma_j(g)|)_{1 \leq j \leq n} \in K_{\mathbb{R}}$, and define $I = |\boldsymbol{\sigma}(g)|^{-1} \cdot \boldsymbol{\sigma}(g\mathcal{O}_K) \subset K_{\mathbb{R}}$ (which is a lattice of rank n). We want to call the LLL algorithm in I, hence we need to compute a basis of I. Such a basis is provided by the set $(|\boldsymbol{\sigma}(g)|^{-1} \cdot \boldsymbol{\sigma}(c_j))_{1 \leq j \leq n}$, where as usual multiplication is performed coordinate-wise. We can compute in polynomial time a floating point approximation of these basis vectors, since we have seen that we know the c_j's, and we can compute (a floating point of approximation of) $|\boldsymbol{\sigma}(g)|$ using the knowledge of $\boldsymbol{\sigma}(g)\overline{\boldsymbol{\sigma}(g)}$.

Once we have computed a basis of I (with sufficient precision), we call the LLL algorithm to recover (in polynomial time) a vector $\boldsymbol{s} \in I$ such that $\|\boldsymbol{s}\| \leq 2^n \lambda_1(I)$ [19]. We know that the vector $|\boldsymbol{\sigma}(g)|^{-1} \cdot \boldsymbol{\sigma}(g)$ belongs to I by definition, and this vectors has euclidean norm $\sqrt{n}$ (since all its coordinates have absolute value 1). Hence, we have $\lambda_1(I) \leq \sqrt{n}$ and so $\|\boldsymbol{s}\| \leq 2^n \sqrt{n}$.

Since $\boldsymbol{s} \in I$, there must exists $a \in \mathcal{O}_K$ such that $\boldsymbol{s} = |\boldsymbol{\sigma}(g)|^{-1} \cdot \boldsymbol{\sigma}(ga)$. Writing $\boldsymbol{s}$ as an integer linear combination of the basis vectors of I allows to recover the element $z = ga \in g\mathcal{O}_K$ in polynomial time. To conclude that z is as desired, it only remains to see that $\boldsymbol{\sigma}(a) = \boldsymbol{s} \cdot |\boldsymbol{\sigma}(g)| \cdot \boldsymbol{\sigma}(g)^{-1}$, so $\|\boldsymbol{\sigma}(a)\| = \|\boldsymbol{s}\| \leq 2^n \lambda_1(I)$, where the inequality comes from the LLL-guarantee on the size of $\boldsymbol{s}$ that we proved above. □

With this lemma at hand, we can now compute the so-called polynomial chain. In the original Gentry–Szydlo algorithm (see, e.g., [16, Lemma 7]), the polynomial chain elements were of the form $g^{k_{r-j}} a_j \overline{a_{j-1}}^2$, where the conjugation of $\overline{a_{j-1}}$ somehow plays the role of inversion, but allows to keep everything

integral. In our case, conjugation is more costly than inversion (because it requires leaving the field K), so we will instead compute chain elements of the form $g^{k_{r-j}} a_j a_{j-1}^{-2}$.

Lemma 11. *Let $g \in K \setminus \{0\}$, $p \in \mathbb{Z}$ be a prime integer, and let $k = \sum_{j=0}^{r} k_j 2^j$ be a positive integer, with $r = \lfloor \log_2(k) \rfloor$. There is a polynomial time algorithm (in the size of g, $\log |\Delta_K|$ and $\log k$) that takes as input the HNF basis of $g\mathcal{O}_K$, the element $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)} \in \boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$, k, and p, and returns a list of non-zero elements $z_0, \ldots, z_r \in K$ such that there exists $a_0, \ldots, a_r \in \mathcal{O}_K$ coprime with p, and $a_{-1} = 1$ satisfying*

- $z_j = g^{k_{r-j}} \cdot a_j \cdot a_{j-1}^{-2}$ *for* $0 \le j \le r$
- $\|\boldsymbol{\sigma}(a_j)\| \le 2^n \cdot \sqrt{n}$ *for* $-1 \le j \le r$.

Proof. For $0 \le j \le r$, let us write $h_j = g^{k_{r-j}} \cdot a_{j-1}^{-2}$. We will prove by induction on j that for all $j \in \{0, \ldots, r\}$, we can compute in polynomial time the ideal $h_j\mathcal{O}_K$, the element $\boldsymbol{\sigma}(h_j) \cdot \overline{\boldsymbol{\sigma}(h_j)}$, and the element z_j. This will prove in particular that we can compute $z_0, \ldots, z_r$ in polynomial time, as desired.
Initialization. Let $j = 0$, then $h_0 = g^{k_r} a_{-1}^{-2} = g$ (because k_r is the most significant bit of k so it has to be 1). Computing $h_0\mathcal{O}_K$ and $\boldsymbol{\sigma}(h_0) \cdot \overline{\boldsymbol{\sigma}(h_0)}$ can then be done in polynomial time from the input $g\mathcal{O}_K$ and $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)}$. In order to compute z_0, we call the algorithm from Lemma 10 on input $g\mathcal{O}_K$ and $\boldsymbol{\sigma}(g)\overline{\boldsymbol{\sigma}(g)}$, and we obtain some z_0 as output. By Lemma 10,this element satisfies $z_0 = g \cdot a_0 = g^{k_r} \cdot a_0 \cdot a_{-1}^{-2}$ and $\|\boldsymbol{\sigma}(a_0)\| \le 2^n \sqrt{n}$ as desired.
Induction step. Let $j \ge 1$, and assume by induction hypothesis that $h_{j-1}\mathcal{O}_K$, $\boldsymbol{\sigma}(h_{j-1}) \cdot \overline{\boldsymbol{\sigma}(h_{j-1})}$, z_{j-1} have already been computed. Recall also that $g\mathcal{O}_K$ and $\boldsymbol{\sigma}(g)\overline{\boldsymbol{\sigma}(g)}$ are assumed to be given as input to the algorithm (hence they are known).

Note that by definition of the h_j's and z_j's, it holds that $h_j = g^{k_{r-j}} \cdot a_{j-1}^{-2} = g^{k_{r-j}} \cdot h_{j-1}^2 \cdot z_{j-1}^{-2}$. Hence, from the knowledge of $g\mathcal{O}_K$, $h_{j-1}\mathcal{O}_K$, and z_{j-1}, one can compute $h_j\mathcal{O}_K = (g\mathcal{O}_K)^{k_{r-j}} \cdot (h_{j-1}\mathcal{O}_K)^2 \cdot (z_{j-1}\mathcal{O}_K)^{-2}$ in polynomial time (by performing basic operations on fractional ideals of K). Similarly, from the knowledge of $\boldsymbol{\sigma}(g)\overline{\boldsymbol{\sigma}(g)}$, $\boldsymbol{\sigma}(h_{j-1}) \cdot \overline{\boldsymbol{\sigma}(h_{j-1})}$, and z_{j-1}, one can compute $\boldsymbol{\sigma}(h_j) \cdot \overline{\boldsymbol{\sigma}(h_j)} = (\boldsymbol{\sigma}(g)\overline{\boldsymbol{\sigma}(g)})^{k_{r-j}} \cdot (\boldsymbol{\sigma}(h_{j-1}) \cdot \overline{\boldsymbol{\sigma}(h_{j-1})})^2 \cdot (\boldsymbol{\sigma}(z_{j-1}) \cdot \overline{\boldsymbol{\sigma}(z_{j-1})})^{-2}$ in polynomial time. Here, we use the fact that multiplication in $\boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ and inversion of an elementary product (such as $\boldsymbol{\sigma}(z_{j-1}) \cdot \overline{\boldsymbol{\sigma}(z_{j-1})}$) can be performed in polynomial time.

It remains to compute z_j. Observe that h_j is non-zero since g and the a_j's are non-zero. Moreover, we have just seen that $h_j\mathcal{O}_K$ and $\boldsymbol{\sigma}(h_j) \cdot \overline{\boldsymbol{\sigma}(h_j)}$ can be computed in polynomial time. Using Lemma 10, we can compute in polynomial time $z_j \in h_j\mathcal{O}_K$ such that $z_j = h_j \cdot a_j$ with $\|\boldsymbol{\sigma}(a_j)\| \le 2^n \cdot \sqrt{n}$ (and $a_j \in \mathcal{O}_K$). Hence z_j and a_j satisfy the two conditions of the lemma.[13] This concludes the induction step.

[13] If we are unlucky and a_j is not coprime with p, we can resample it until it is the case.

Conclusion. The above induction shows that computing the quantities at step $j+1$ can be done in polynomial time from the quantities at step j. Since the number of steps is $r = O(\log(k))$, the overall complexity remains polynomial. We note also that the size of the objects manipulated during the successive steps of the computation does not increase, since we always have $\|\boldsymbol{\sigma}(a_j)\| \leq 2^n \cdot \sqrt{n}$, which provides an absolute upper bound on the size of the a_j's (independent on the index j of the iteration step). □

A.2 Computing a small power of the secret

In this section, we will use the polynomial chain from the previous section to compute a (somewhat) small power of the secret element g. Before proving this, we will need the following lemma, which tells us that if we know some element $a \in \mathcal{O}_K$ modulo $m\mathcal{O}_K$ for some $m \in \mathbb{Z}$ large enough, then we can recover a exactly. In the original Gentry–Szydlo algorithm over power-of-two cyclotomic fields, the analogous result was easily proved because the basis $(1, X, \ldots, X^{n-1})$ formed an orthogonal basis of $\boldsymbol{\sigma}(\mathcal{O}_K)$. In the general case, we use our known LLL-reduced basis of $\mathcal{O}_K$ and the bound is slightly less favorable (but this will not appear in the final statement).

Lemma 12. *Let K be a number field of degree n. Let $a \in \mathcal{O}_K$ and $m \geq 0$ be an integer such that $m \geq 2^{n+1}/\sqrt{n} \cdot \|\boldsymbol{\sigma}(a)\|$. Assume that an LLL-reduced basis $(b_1, \ldots, b_n)$ of $\mathcal{O}_K$ is known (LLL-reduced for the norm induced by the canonical embedding). Given $a \bmod m \in \mathcal{O}_K/m\mathcal{O}_K$, one can recover $a \in \mathcal{O}_K$ exactly in time polynomial in $\log|\Delta_K|$ and $\log m$.*

Proof. Let $t = a + mx$ with $a, x \in \mathcal{O}_K$ be any representative of $a \mod m\mathcal{O}_K$. We aim to recover the element $a \in \mathcal{O}_K$ given the target $t \in a + m\mathcal{O}_K$ using Babai's nearest plane algorithm. By Lemma 1 (applied to the lattice $\boldsymbol{\sigma}(m\mathcal{O}_K)$) a sufficient condition for this is that $\|\boldsymbol{\sigma}(a)\| < 2^{-n} \cdot \lambda_1(\boldsymbol{\sigma}(m\mathcal{O}_K))$. In the preliminaries we have seen that $\lambda_1(\boldsymbol{\sigma}(\mathcal{O}_K)) \geq \sqrt{n}$. The condition on m then implies that $2^{-n} \cdot \lambda_1(\boldsymbol{\sigma}(m\mathcal{O}_K)) \geq 2^{-n} m\sqrt{n} \geq 2\|\boldsymbol{\sigma}(a)\| > \|\boldsymbol{\sigma}(a)\|$ as desired. So Babai's nearest plane algorithm efficiently recovers $\boldsymbol{\sigma}(a)$. Given enough precision in those floating point computation, one can then recover exactly the integer coefficients of $a \in \mathcal{O}_K$ in the basis $(\boldsymbol{\sigma}(b_1), \ldots, \boldsymbol{\sigma}(b_n))$. □

In the following lemmas, we will compute powers of the secret element g, where the exponent depends on the quantity $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ for some prime integer $p \in \mathbb{Z}$. Recall that, for a finite multiplicative group G, the integer $o_{\max}(G)$ is the least common multiple of the orders of all the elements of G. For $p \in \mathbb{Z}$ a prime number not dividing $|\Delta_K|$,[14] we then want to prove that one can compute $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ in polynomial time. To do so, we first factor the ideal $p\mathcal{O}_K = \prod_{j=1}^{r} \mathfrak{p}_j^{e_j}$ (with the $\mathfrak{p}_j$'s distinct prime ideals). This factorization can be computed in (classical) polynomial time [8, Section 6.2.5]. Moreover, since p does not divide Δ_K, we know that each e_j is equal to 1. Hence, by Chinese

[14] Equivalently, p does not ramify in K.

remainder theorem, it holds that $(\mathcal{O}_K/p\mathcal{O}_K)^\times = \prod_{j=1}^r (\mathcal{O}_K/\mathfrak{p}_j)^\times$. Each of the $\mathcal{O}_K/\mathfrak{p}_j$ is a finite field of cardinality $\mathcal{N}(\mathfrak{p}_j)$, which implies that $(\mathcal{O}_K/\mathfrak{p}_j\mathcal{O}_K)^\times$ is a cyclic group of cardinality $\mathcal{N}(\mathfrak{p}_j) - 1$. Hence we obtain $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times) = \mathrm{lcm}_j(\mathcal{N}(\mathfrak{p}_j) - 1)$, which can be computed in polynomial time from the $\mathfrak{p}_j$'s.

The lemma below proves that one can compute a large power of the secret element g modulo (almost) any prime q. The fact that the computation is performed modulo q here is important if we want a polynomial time algorithm, because the power of g we compute is so large that computing it exactly would be too costly.

Lemma 13. *Let $p \in \mathbb{Z}$ be a prime number satisfying $p \geq 2^{2n+1}$ and $g \in \mathcal{O}_K$ be invertible modulo p. Given the prime p, another prime integer q, the HNF basis of $g\mathcal{O}_K$ and the element $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)}$, we can compute (for almost all choice of q) the element*

$$g^{o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)} \bmod q\mathcal{O}_K$$

in polynomial time in $\log|\Delta_K|$, the size of g, $\log(p)$ and $\log(q)$. The computation succeeds, except for polynomially (in $\log(p)$ and n) many values of q.

Proof. Let $k = o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$. Recall from preliminaries that this quantity can be computed in polynomial time given p and a basis of $\mathcal{O}_K$. Since g lies in $(\mathcal{O}_K/p\mathcal{O}_K)^\times$, then we know that $g^k = 1 \bmod p$, which will be useful later on in the proof.

Let us write $r = \lfloor \log_2(k) \rfloor$ and $k = \sum_{j=0}^r k_j 2^j$ as in Lemma 11. Using Lemma 11, we can compute in polynomial time a list of non-zero elements $z_0, \ldots, z_r \in K$ of the form $z_j = g^{k_{r-j}} \cdot a_j \cdot a_{j-1}^{-2}$, for some $a_j \in \mathcal{O}_K$ coprime with p satisfying $a_{-1} = 1$ and $\|\boldsymbol{\sigma}(a_j)\| \leq 2^n \cdot \sqrt{n}$ for all j. Since both the a_j's and g are coprime with p, then one can reduce the z_j's modulo p and obtain an element $z_j \bmod p \in (\mathcal{O}_K/p\mathcal{O}_K)^\times$. One can then compute $\prod_{j=0}^r z_j^{2^{r-j}} \bmod p$ in polynomial time, by repeated squaring. Expending the product, we have

$$\begin{aligned}
\prod_{j=0}^r z_j^{2^{r-j}} \bmod p &= \prod_{j=0}^r (g^{k_{r-j}} a_j \cdot a_{j-1}^{-2})^{2^{r-j}} \bmod p \\
&= a_r \cdot a_{-1}^{-2^{r+1}} \cdot \prod_{j=0}^r g^{2^{r-j} k_{r-j}} \bmod p \\
&= a_r \cdot g^k \bmod p = a_r \bmod p.
\end{aligned}$$

For the last equality, we used the fact that $g^k = 1 \bmod p$, and in the previous equality we used the fact that $a_{-1} = 1$. Summing up, we can compute $a_r \bmod p$ in polynomial time from the z_j's. Now, observe that from the condition in the lemma statement and using the fact that $\|\boldsymbol{\sigma}(a_r)\| \leq 2^n\sqrt{n}$, it holds that $p \geq 2^{2n+1} \geq 2^{n+1}/\sqrt{n} \cdot \|\boldsymbol{\sigma}(a_r)\|$. Using Lemma 12, we conclude that we can recover $a_r \in \mathcal{O}_K$ in polynomial time from $a_r \bmod p$ (and the LLL-reduced basis of $\mathcal{O}_K$ that we assumed we know in all this section).

From the knowledge of a_r, let us now explain how to compute $g^k \bmod q$ in polynomial time. In order to do so, we want to ensure that all the a_j's are

invertible modulo q, i.e., that q does not divide the algebraic norm $\mathcal{N}(a_j)$ of any of the a_j's. Note that for each j, it holds that $|\mathcal{N}(a_j)| \leq \|\boldsymbol{\sigma}(a_j)\|^n \leq (2^n \cdot \sqrt{n})^n$. Hence, $\mathcal{N}(a_j)$ is divisible by at most $\log_2(\mathcal{N}(a_j)) = \text{poly}(n)$ prime factors. Since we have $r + 1 = \text{poly}(\log k)$ such a_j's, we conclude that the number of q that are not allowed (because they are not coprime with at least one of the a_j's) is at most polynomial in $\log k$ and n. Finally, observe that $k = o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ is upper bounded by $|\mathcal{O}_K/p\mathcal{O}_K| = p^n$, so $\log k = \text{poly}(n, \log p)$. This means that we exclude at most $\text{poly}(n, \log p)$ choices of q, as stated in the lemma.

From now on, we then assume that q is such that all the a_j's are invertible modulo q. Using the same reasoning and algorithm as above, we can compute in polynomial time the element $\prod_{j=0}^{r} z_j^{2^{r-j}} \bmod q = a_r \cdot g^k \bmod q \in \mathcal{O}_K/q\mathcal{O}_K$, by repeated squaring and multiply. Since a_r has been computed before and it is invertible modulo q, we can divide by a_r and obtain $g^k \bmod q$ as desired. □

We are now ready to prove the main lemma of this section, which states that one can compute a (somewhat) small power of the secret element g in polynomial time. The idea of the algorithm is to compute two large powers of g modulo the same q using Lemma 13 above, then combine these powers to obtain a smaller power of g modulo q. If this power is sufficiently small and q is large enough, we can then use Lemma 12 to recover the power of g in $\mathcal{O}_K$.

Lemma 14. *Let $p_1, p_2 \in \mathbb{Z}$ be prime numbers satisfying $2^{2n+1} \leq p_1, p_2 \leq 2^{2n+2}$ and $g \in \mathcal{O}_K$ be invertible modulo p_1 and p_2. Let $r = \gcd\Big(o_{\max}((\mathcal{O}_K/p_1\mathcal{O}_K)^\times), o_{\max}((\mathcal{O}_K/p_2\mathcal{O}_K)^\times)\Big)$. Given p_1, p_2, the HNF basis of $g\mathcal{O}_K$ and the element $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)}$, we can compute the element $g^r \in \mathcal{O}_K$ in time polynomial in r, $\log|\Delta_K|$ and the size of g.*

Proof. The algorithm first uses the knowledge of $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)}$ to compute (a floating point approximation of) $\|\boldsymbol{\sigma}(g)\| \in \mathbb{R}$. Then, it sets $B = 2^{n+1} \cdot \lceil \|\boldsymbol{\sigma}(g)\|^r \rceil$, and it samples $q \in \{B, \ldots, 2B\}$ until q is prime and until q is a valid prime for applying Lemma 13 with p_1 and with p_2. We know that there are only $\text{poly}(n)$ many bad primes for applying Lemma 13 with p_1, and similarly for p_2. So the probability to find a good q when sampling it uniformly in $\{B, \ldots, 2B\}$ is $\Omega(1/\log(B)) - \text{poly}(n)/B = \Omega(1/\log(B))$. This means that the expected number of iteration needed to find a suitable q is polynomial in $\log(B)$, which is itself polynomial in r, $\log|\Delta_K|$ and the size of g.

Once we have found a suitable prime q, we call the algorithm from Lemma 13 twice: once on input p_1, q, $g\mathcal{O}_K$ and $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)}$, and once on input p_1, q, $g\mathcal{O}_K$ and $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)}$. The algorithm can be run in polynomial time according to Lemma 13 (using the fact that $\log(p_1), \log(p_2)$ and $\log(q)$ are all $\text{poly}(\log|\Delta_K|, r, \text{size}(g))$). The algorithm outputs the two elements $h_1 = g^{k_1} \bmod q$ and $h_2 = g^{k_2} \bmod q$, where $k_j = o_{\max}((\mathcal{O}_K/p_j\mathcal{O}_K)^\times)$.

Using the extended gcd algorithm, we then compute $u, v \in \mathbb{Z}$ such that $uk_1 + vk_2 = \gcd(k_1, k_2) = r$. And we finally compute $h := h_1^u \cdot h_2^v \bmod q = g^r \bmod q$. This computation can be performed in polynomial time in $\log(u)$,

$\log(v)$, $\log(q)$ and $\log|\Delta_K|$, which are all polynomial in r, $\log|\Delta_K|$ and the size of g.

To conclude the proof, we use the fact that

$$\|\boldsymbol{\sigma}(g^r)\| \leq \sqrt{n} \cdot \|\boldsymbol{\sigma}(g^r)\|_\infty \leq \sqrt{n} \cdot \|\boldsymbol{\sigma}(g)\|_\infty^r \leq \sqrt{n} \cdot \|\boldsymbol{\sigma}(g)\|^r.$$

Hence, by choice of B and q, it holds that $q \geq B \geq 2^{n+1}/\sqrt{n} \cdot \|\boldsymbol{\sigma}(g^r)\|$. Using Lemma 12, we can then recover $g^r \in \mathcal{O}_K$ exactly in polynomial time from $g^r \bmod q$ (using the known LLL-reduced basis of $\mathcal{O}_K$). □

A.3 Finding good primes

In order to obtain an efficient algorithm, we would like that the exponent r from Lemma 14 is of polynomial size (and ideally as small as possible). Hence, the main question remaining is whether there exist suitable primes p_1 and p_2 for which $r := \gcd\left(o_{\max}((\mathcal{O}_K/p_1\mathcal{O}_K)^\times), o_{\max}((\mathcal{O}_K/p_2\mathcal{O}_K)^\times)\right)$ is polynomially bounded (let's say polynomial in n), and if yes, how easily computable are these suitable primes p_1 and p_2. Below, we define the notion of Gentry–Szydlo friendly fields, which are fields for which we have a positive answer to the two questions above: it is possible to efficiently find two primes p_1 and p_2 such that r is polynomial. We then argue that all number fields are likely to be Gentry–Szydlo friendly fields.

Definition 1. *Let $\beta, \delta \geq 1$. We say that a number field K of degree n is (β, δ)-Gentry–Szydlo friendly (or GS-friendly for short) if*

$$\Pr_{p_1,p_2 \xleftarrow{\$} \mathbb{P} \cap [2^{2n+1}, 2^{2n+2}]} \left(\gcd\left(o_{\max}((\mathcal{O}_K/p_1\mathcal{O}_K)^\times), o_{\max}((\mathcal{O}_K/p_2\mathcal{O}_K)^\times)\right) \leq \beta \right) \geq 1/\delta,$$

where $\mathbb{P}$ is the set of all prime integers, and p_1, p_2 are sampled independently.

Remark 1. There is no fundamental reason why we consider only two primes p_1 and p_2, and not a constant number of primes (or even a polynomial number of primes). Lemma 14 could be adapted to more primes, resulting only in a polynomial dependency in the running time on the number of primes considered. Considering more primes could be useful if finding two primes with a small gcd happens to be hard: the more prime we use, the smaller their gcd will be. Experiments seem to indicate however that only two primes are usually sufficient to obtain a relatively small gcd, hence we do not pursue this idea of using more than two primes.

We make the following heuristic assumption, which we justify theoretically and experimentally below.

Heuristic 2. *There exists some polynomial functions $\beta(n), \delta(n)$ such that any number field K of degree n is $(\beta(n), \delta(n))$-Gentry–Szydlo friendly.*

Justification. First, we argue that the gcd over all sufficiently large primes p of $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ is likely to be equal to $|\mu(K)|$, the number of roots of unity in K. Indeed, if p is large enough such that all the elements of $\mu(K)$ are distinct modulo p, then the image of $\mu(K)$ modulo p provides a cyclic group of cardinality $|\mu(K)|$ in $(\mathcal{O}_K/p\mathcal{O}_K)^\times$, and so it holds that $|\mu(K)|$ divides $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$. Reciprocally, assume that $d > 0$ divides all $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ (at for p large enough). Then it means that the group $(\mathcal{O}_K/p\mathcal{O}_K)^\times$ contains an element of order n. In other words, for any p (sufficiently large), the polynomial $X^n - 1$ has a root in $\mathcal{O}_K/p\mathcal{O}_K$. Then, it seems reasonable to assume that if this polynomial has a root locally modulo each prime p, then it also has a global root in $\mathcal{O}_K$. If this is the case, then this root it a primitive n-th root of unity in K, and so it implies that n divides $|\mu(K)|$.

Combining this argument and the fact that $|\mu(K)| \leq 2n^2$ justifies the heuristic claim for $\beta(n) = 2n^2$. The fact that taking only two random primes is sufficient to obtain this gcd (or a gcd not much bigger) with non-negligible probability is justified by the experiments.

We provide experimental data in Fig. 6 and Figs. 7 to 9 for three families of fields: $K = K_1(i)$ where K_1 is an NTRU Prime field, for cyclotomic fields, and number fields defined by a random irreducible polynomial of a given degree (we call the latter ones "random fields" from now on). For NTRU Prime fields K_1, the number of roots of unity in $K = K_1(i)$ is always 4. For random number fields we expect only two roots of unity. For cyclotomic fields, the number of roots of unity is larger than 2, and can be efficiently computed. For these three families of fields, we computed the gcd over 100 random primes p (between 2^{2n+1} and 2^{2n+2}) of the quantity $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$. We observe that this gcd over 100 primes almost always matches the number of roots of unity in K for all families of field considered (and when it does not match it, it is off by a multiplicative factor 2 or 3). We also computed $\gcd\left(o_{\max}((\mathcal{O}_K/p_1\mathcal{O}_K)^\times), o_{\max}((\mathcal{O}_K/p_2\mathcal{O}_K)^\times)\right)$ for only two random primes (between 2^{2n+1} and 2^{2n+2}). We sampled 50 pairs of random primes and kept the smallest gcd among all the ones we obtained. When doing so, we obtained a gcd which is, most of the time, not much larger than the number of roots of unity of K. More importantly, the ratio between the gcd we obtain and the number of roots of unity of K does not seem to increase significantly with the degree of K. △

A.4 Concluding the proof

We are now ready to state and prove a generalization of the Gentry–Szydlo algorithm to all Gentry–Szydlo friendly fields.

Theorem 5. *Let $\beta, \delta \geq 1$ be integers. Let K be a (β, δ)-Gentry–Szydlo friendly number field, and assume that we know an LLL-reduced $\mathbb{Z}$-basis of $\mathcal{O}_K$ (for the canonical embedding). Let $g \in \mathcal{O}_K \setminus \{0\}$. Given the HNF basis of $g\mathcal{O}_K$ and the element $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)} \in \boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ (represented as in Section 2.4), we can*

recover g up to multiplication by a root of unity of K (i.e., we recover $g\varepsilon$ for some $\varepsilon \in \mu(K)$) in time polynomial in the size of g, in $\log|\Delta_K|$, in β, and in δ.

Under Heuristic 2, all number fields are (β, δ)-Gentry–Szydlo friendly for some polynomial values of β and δ. Hence, the above algorithm can be applied to all number fields and we obtain the following heuristic claim.

Heuristic Claim 2 (Generalized Gentry–Szydlo) *Let K be any number field, and assume that we know an LLL-reduced $\mathbb{Z}$-basis of $\mathcal{O}_K$. Let $g \in \mathcal{O}_K \setminus \{0\}$. Under Heuristic 2, there is a probabilistic polynomial time (in its input size and in $\log|\Delta_K|$) algorithm that takes as input the HNF basis of $g\mathcal{O}_K$ and the element $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)} \in \boldsymbol{\sigma}(K) \cdot \overline{\boldsymbol{\sigma}(K)}$ and outputs $g\varepsilon$ for some $\varepsilon \in \mu(K)$.*

Proof (Proof of Theorem 5). The algorithm sets $B = 2^{2n+1}$ and samples randomly and independently p_1 and p_2 in $\{B, \ldots, 2B\}$ until they are both prime, both coprime to the ideal $g\mathcal{O}_K$, and until we have $\gcd\big(o_{\max}((\mathcal{O}_K/p_1\mathcal{O}_K)^\times), o_{\max}((\mathcal{O}_K/p_2\mathcal{O}_K)^\times)\big) \leq \beta$. By definition of (β, δ)-Gentry–Szydlo friendly fields, we know that these conditions will be met with probability $\Omega(1/(\delta \cdot (\log B)^2))$,[15] so the expectation of the number of iterations needed to find suitable primes p_1 and p_2 satisfying the above condition is polynomial in $\log(B) = \operatorname{poly}|\Delta_K|$ and in δ.

We then call the algorithm from Lemma 14 on input p_1, p_2, $g\mathcal{O}_K$ and $\boldsymbol{\sigma}(g) \cdot \overline{\boldsymbol{\sigma}(g)}$, and we recover $g^r \in \mathcal{O}_K$ in time $\operatorname{poly}(r, \log|\Delta_K|, \operatorname{size}(g))$, where $r := \gcd\big(o_{\max}((\mathcal{O}_K/p_1\mathcal{O}_K)^\times), o_{\max}((\mathcal{O}_K/p_2\mathcal{O}_K)^\times)\big)$ satisfies $r \leq \beta$.

Last, we use the algorithm from Lemma 2 to compute the roots of the polynomial $X^r - g^r$ in K and return any of these roots. We know that g has to be among the set of roots, so the set is non-empty. We also know that any other root α satisfies $\alpha = g \cdot \varepsilon$ for some $\varepsilon \in K$ such that $\varepsilon^r = 1$. In particular, ε is a root of unity in K, and so α has the desired shape. Our algorithm is correct, and all its steps can be performed in polynomial time in the size of g, in $\log|\Delta_K|$, in β, and in δ. □

[15] Note that there are at most $\log_2|\mathcal{N}(g)|$ prime integers not coprime with the ideal $g\mathcal{O}_K$.

degree of the field	gcd of $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ over 100 random primes	gcd of $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ over 2 random primes (best among 50 trials)
14	4	4
34	4	4
58	4	16
82	8	4
106	4	4
134	4	4
158	24	4
178	4	48
202	4	4
226	4	56
254	4	24
274	8	240
298	4	240

Fig. 7: Experimental results validating Heuristic 2 for $K_1(i)$ where K_1 is an NTRU Prime field.

degree of the field	number of roots of unity in K	gcd of $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ over 100 random primes	gcd of $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ over 2 random primes (best among 50 trials)
4	10	10	10
18	54	54	54
20	44	44	44
24	78	78	78
48	112	112	112
48	180	180	180
60	122	122	366
72	190	190	190
72	146	146	1022
84	258	258	258
92	282	282	282
106	214	214	214
120	248	248	248
120	462	462	462
162	326	326	326
196	394	394	394
208	530	530	530
264	598	598	598

Fig. 8: Experimental results validating Heuristic 2 for Cyclotomic fields.

degree of the field	gcd of $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ over 100 random primes	gcd of $o_{\max}((\mathcal{O}_K/p\mathcal{O}_K)^\times)$ over 2 random primes (best among 50 trials)
10	2	2
27	2	4
44	2	4
61	2	2
78	2	4
95	2	2
112	2	10
129	2	14
146	2	2
163	2	48
180	2	4
197	2	14
214	2	2
231	2	12
248	2	2
265	2	4
282	2	2
299	2	2

Fig. 9: Experimental results validating Heuristic 2 for random fields (i.e., fields whose defining polynomial is sampled randomly).