FuLeakage: Breaking FuLeeca by Learning Attacks

Felicitas Hörmann[1,2] and Wessel van Woerden[3]

[1] Institute of Communications and Navigation, German Aerospace Center (DLR),
Oberpfaffenhofen–Wessling, Germany
felicitas.hoermann@dlr.de
[2] School of Computer Science, University of St. Gallen, St. Gallen, Switzerland
[3] Univ. Bordeaux, CNRS, Inria, Bordeaux INP, IMB, Talence, France
wessel.van-woerden@math.u-bordeaux.fr

Abstract. FuLeeca is a signature scheme submitted to the recent NIST call for additional signatures. It is an efficient hash-and-sign scheme based on quasi-cyclic codes in the Lee metric and resembles the lattice-based signature Falcon. FuLeeca proposes a so-called concentration step within the signing procedure to avoid leakage of secret-key information from the signatures. However, FuLeeca is still vulnerable to learning attacks, which were first observed for lattice-based schemes. We present three full key-recovery attacks by exploiting the proximity of the *code*-based FuLeeca scheme to *lattice*-based primitives.

More precisely, we use a few signatures to extract an $n/2$-dimensional circulant sublattice from the given length-n code, that still contains the exceptionally short secret-key vector. This significantly reduces the classical attack cost and, in addition, leads to a full key recovery in quantum-polynomial time. Furthermore, we exploit a bias in the concentration procedure to classically recover the full key for any security level with at most 175,000 signatures in less than an hour.

1 Introduction

Most of today's asymmetric cryptosystems are based on discrete-logarithm problems or integer factorization and will not withstand powerful quantum computers. Thus, the U.S. National Institute of Standards and Technology (NIST) currently aims to standardize cryptographic schemes that resist conventional as well as quantum attacks. The project started with the first call for post-quantum algorithms in 2016 and now arrived at the fourth and last round [25]. Since many of the remaining signature schemes share similar underlying security primitives, NIST opened up a call for additional signatures in 2023 [26] to foster diversity.

One of the new proposals is FuLeeca [32,33], which is the first scheme relying on coding-theoretical problems in the Lee metric. FuLeeca adopts a hash-and-sign approach and thus generates signatures as codewords that are close to the hashed message. Then, *any* basis of the code allows the recipient to verify the signature, whereas the signer needs access to the secret key, i.e., a *good* basis for decoding the signature from the target hash.

The hash-and-sign paradigm is the basis for code-based schemes such as CFS [11] and Wave [5], but it is also well-known in lattice-based cryptography. Unfortunately, the first lattice proposals adopting this strategy, namely GGH [19] and NTRUSIGN [20], were fully broken by learning attacks [18, 27]. In particular, already a few hundred NTRUSIGN signatures leak enough information to recover the secret key, as their generation heavily depends on the secret lattice basis. Several heuristic approaches tried to prevent leakage [21, 22, 28, 35] but failed in the end [14, 23, 37]. In contrast, signatures generated with the GPV framework [17] *provably* do not leak any information about the secret key and thus resist learning-type attacks. Instantiations of the GPV framework are e.g. the code-based scheme Wave [5] and the lattice-based signature FALCON [31], and NIST selected the latter for standardization [25].

FuLeeca does *not* adopt the GPV framework and provides *no* proof of non-leakage. Instead, a heuristic *concentration step* is added to the signing algorithm to prevent leakage. Unfortunately, this countermeasure is not enough.

Table 1: Overview of the presented attacks against FuLeeca.

	few signatures ($\ll 100$)	**many signatures** ($\leq$ 175,000)
classical attack	leaked-sublattice attack (reduced security) → section 4	learning attack (full break) → section 6
quantum attack	ideal-structure attack (full break) → section 5	← see this attack

Contributions. We present three types of attacks against the signature scheme FuLeeca, which all profit from the proximity of FuLeeca to known lattice-based constructions. This showcases once more the close connection between code- and lattice-based cryptography. Table 1 classifies the three attacks and provides pointers to the corresponding sections. In the following, we give a short summary of each approach:

Firstly, we present a *leaked-sublattice attack*. The idea of applying basis reduction to the construction-A lattice obtained from the public key was already considered in the FuLeeca specification but did not influence the parameter choices as it is less efficient than other combinatorial attacks. We observe however that all signatures are part of a lower-dimensional sublattice generated by the short secret vectors. Moving to this lower-dimensional sublattice, for which a basis can be computed from a small sample of signatures, gives a drastic speedup. The attack reduces the security level of the FuLeeca-I, FuLeeca-III, and FuLeeca-V parameter sets from 160, 224, and 288 bits to only 111, 155, and 199 bits, respectively.

Secondly, we perform an *ideal-structure attack* by realizing that the first half of the secret vector is an unusually short element in a circulant lattice. Again, a basis for this circulant lattice can be computed from a few signatures. This enables a polynomial-time quantum attack in the FuLeeca setting, thus giving a full quantum key recovery for all three security levels.

Thirdly, we set up a *learning attack* which yields a full key recovery when enough signature vectors are known. The attack exploits a bias in the signing procedure, which lets every signature leak information about the secret key. In practice, we recover the full secret key of FuLeeca-I, FuLeeca-III, and FuLeeca-V instances with only 90,000, 175,000, and 175,000 signatures, respectively. The observed success rate is shown in Figure 1 and is indeed higher for the supposedly more secure FuLeeca-V instances than for the FuLeeca-III samples.

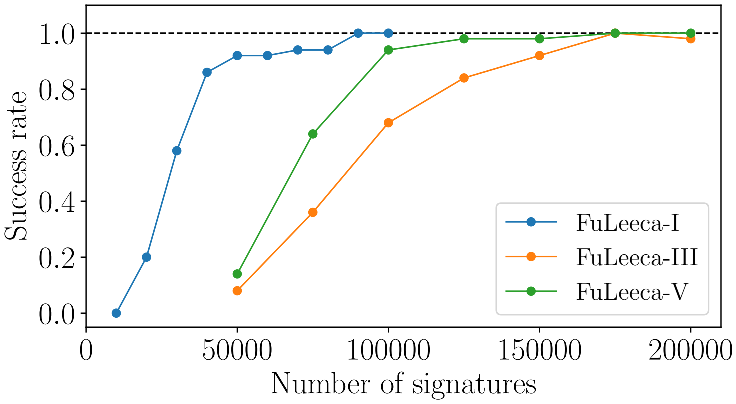


Fig. 1: Success rate of the learning attack with respect to the number of available signatures. Averaged over 50 instantiations for each parameter set.

In summary, we can perform a full key recovery in polynomial time if a reasonable amount of signatures was collected, and in quantum-polynomial time for a small sample of signatures. This is a full break of FuLeeca for all proposed parameter sets, regardless of the availability of quantum computers.

2 Preliminaries

We aim at making this paper accessible for readers from both the lattice community and the coding community. Thus, we give rather detailed preliminaries to provide a mostly self-contained presentation of our results.

Generic notation. Let p be an odd prime and denote the finite field of order p by $\mathbb{F}_p \cong \mathbb{Z}/p\mathbb{Z}$. Note that FuLeeca, and thus this work, fixes $p = 65{,}521$ in all parameter sets. Vectors are considered as *row* vectors and are denoted in lower-case bold, as e.g. $\boldsymbol{a}$, while matrices are denoted in upper-case bold, as e.g. $\boldsymbol{A}$. We identify $\mathbb{F}_p$ with $\left\{-\frac{p-1}{2}, \ldots, \frac{p-1}{2}\right\}$ and implicitly lift vectors and matrices over $\mathbb{F}_p$ to the corresponding objects over $\mathbb{Z}$.

For a real number $x \in \mathbb{R}$, we write $\lfloor x \rfloor$ for the largest integer $n \leq x$, $\lceil x \rceil$ for the smallest integer $n \geq x$, and $\lfloor x \rceil \in \mathbb{Z}$ for the unique integer such that $x - \lfloor x \rceil \in \left[-\frac{1}{2}, \frac{1}{2}\right)$ holds. We use $\operatorname{trunc}(x)$ to denote the rounding of x towards 0, i.e., $\operatorname{trunc}(x) = \lfloor x \rfloor$ if $x \geq 0$, and $\operatorname{trunc}(x) = \lceil x \rceil$ otherwise. The *sign* of x is denoted by $\operatorname{sgn}(x) \in \{-1, 0, 1\}$, and defined as $\operatorname{sgn}(x) = 1$ if $x > 0$, $\operatorname{sgn}(x) = -1$ if $x < 0$, and $\operatorname{sgn}(0) = 0$. For a vector $\boldsymbol{a}$ or a matrix $\boldsymbol{A}$ with coefficients in a set S, we write $f(\boldsymbol{a})$ or $f(\boldsymbol{A})$ to apply a function $f : S \to S$ coefficient-wise.

Circulant matrices. Let $\mathbb{F}$ be any field. We define the *circular shift* of a vector $\boldsymbol{a} = (a_1, \ldots, a_k) \in \mathbb{F}^k$ as $\operatorname{shift}(\boldsymbol{a}) := (a_k, a_1, \ldots, a_{k-1}) \in \mathbb{F}^k$. For $i \in \mathbb{N}$, the i-fold shift of $\boldsymbol{a}$ is obtained by applying the shift-operator i times and denoted by $\operatorname{shift}^i(\boldsymbol{a})$. We further construct a matrix $\operatorname{Shift}(\boldsymbol{a}) \in \mathbb{F}^{k \times k}$ by using all possible shifts of a given vector $\boldsymbol{a} \in \mathbb{F}^k$ as its rows. Namely,

$$\operatorname{Shift}(\boldsymbol{a}) := \begin{pmatrix} \boldsymbol{a} \\ \operatorname{shift}(\boldsymbol{a}) \\ \vdots \\ \operatorname{shift}^{k-1}(\boldsymbol{a}) \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \ldots & a_k \\ a_k & a_1 & \ldots & a_{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_2 & a_3 & \ldots & a_1 \end{pmatrix}.$$

The matrix $\operatorname{Shift}(\boldsymbol{a})$ is called *circulant* and the identity $\operatorname{shift}(\boldsymbol{a}) \cdot \operatorname{Shift}(\boldsymbol{b}) = \operatorname{shift}(\boldsymbol{b}) \cdot \operatorname{Shift}(\boldsymbol{a})$ holds for any $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^k$.

2.1 Codes

A *linear code* $\mathcal{C}$ of length n and dimension k is a k-dimensional subspace of $\mathbb{F}_p^n$. We usually represent it as the row space of a matrix and call any full-rank matrix $\boldsymbol{G} \in \mathbb{F}_p^{k \times n}$ with $\mathcal{C} = \langle \boldsymbol{G} \rangle_{\mathbb{F}_p}$ a *generator matrix* of $\mathcal{C}$. We can apply Gaussian elimination to transform a generator matrix, up to column permutations, into its systematic form $(\boldsymbol{I}_k \mid \boldsymbol{T})$, where $\boldsymbol{T} \in \mathbb{F}_p^{k \times (n-k)}$ and $\boldsymbol{I}_k \in \mathbb{F}_p^{k \times k}$ is the identity matrix. The classical weight that is considered for linear codes is the *Hamming weight*. It counts the number of nonzero entries of a vector, i.e.,

$$\operatorname{wt}_H(\boldsymbol{x}) := |\{i \in \{1, \ldots, n\} : x_i \neq 0\}| \qquad \text{for any } \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_p^n.$$

Quasi-cyclic codes. A k-dimensional *quasi-cyclic code* of length $n = 2k$ has a generator matrix $\boldsymbol{G} = (\boldsymbol{A} \mid \boldsymbol{B}) \in \mathbb{F}_p^{k \times n}$ consisting of two circulant blocks $\boldsymbol{A} = \operatorname{Shift}(\boldsymbol{a}) \in \mathbb{F}_p^{k \times k}$ and $\boldsymbol{B} = \operatorname{Shift}(\boldsymbol{b}) \in \mathbb{F}_p^{k \times k}$. If $\boldsymbol{A}$ is invertible over $\mathbb{F}_p$, the matrix $\boldsymbol{T} = \boldsymbol{A}^{-1}\boldsymbol{B}$ in the systematic form $(\mathbf{I}_k \mid \boldsymbol{T})$ of $\boldsymbol{G}$ is also circulant.

Lee metric. The *Lee weight* of an element $x \in \mathbb{F}_p$ is defined as $\operatorname{wt}_L(x) := |x|$, where we use the symmetric field representation $\mathbb{F}_p = \left\{-\frac{p-1}{2}, \ldots, \frac{p-1}{2}\right\}$. The Lee weight can be extended to vectors additively, i.e.,

$$\operatorname{wt}_L(\boldsymbol{x}) = \sum\nolimits_{i=1}^{n} \operatorname{wt}_L(x_i) \qquad \text{for any } \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_p^n.$$

The metric induced by this weight, that is, $d_L(\boldsymbol{x}, \boldsymbol{y}) := \operatorname{wt}_L(\boldsymbol{x} - \boldsymbol{y})$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_p^n$, is called the *Lee metric* on $\mathbb{F}_p^n$.

Typical Lee vectors. Suppose one wants to sample a vector $\boldsymbol{x} \in \mathbb{F}_p^n$ uniformly at random from the Lee sphere

$$\mathcal{S}_L(t, n) := \{\boldsymbol{v} \in \mathbb{F}_p^n : \mathrm{wt}_L(\boldsymbol{v}) = t\}$$

of vectors having a fixed Lee weight t. This is not as straightforward as e.g. for the Hamming metric, where sampling t uniform nonzero values and a random permutation suffices. A vector of Lee weight t can be obtained by first choosing a weight partition $t = t_1 + \ldots + t_n$ with integers $0 \leq t_i \leq \frac{p-1}{2}$ and then returning a signed permutation of the vector $\boldsymbol{t} = (t_1, \ldots, t_n)$. Uniform sampling over $\mathcal{S}_L(t, n)$ is possible by properly randomizing the chosen partition and the signed permutation [6].

However, randomizing the partition correctly is costly and therefore FuLeeca fixes a precomputed *typical partition* or *typical Lee vector* $\boldsymbol{t}$ for the used length n, modulus p, and Lee weight t, and only randomizes by applying a signed permutation to it. In other words, FuLeeca samples uniformly from the set

$$T(\boldsymbol{t}) := \{\pi(\boldsymbol{t}) : \pi \text{ is a signed permutation}\}.$$

2.2 Lattices

A *lattice* $\mathcal{L} \subset \mathbb{R}^n$ is a discrete subgroup of the vector space $\mathbb{R}^n$. A prominent example is the integer lattice $\mathbb{Z}^n \subset \mathbb{R}^n$. More generally, let $\boldsymbol{B} \in \mathbb{R}^{k \times n}$ be the matrix whose rows are the $\mathbb{R}$-linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k \in \mathbb{R}^n$. Then we define the lattice $\mathcal{L}(\boldsymbol{B})$ with *basis* $\boldsymbol{B}$ by

$$\mathcal{L}(\boldsymbol{B}) := \mathbb{Z}^k \cdot \boldsymbol{B} := \left\{ \sum\nolimits_{i=1}^{k} x_i \boldsymbol{b}_i : x_i \in \mathbb{Z} \quad \forall i \right\}.$$

We call k the *rank* or *dimension* of the lattice, and $\mathcal{L}(\boldsymbol{B})$ has full rank if $k = n$.

For lattices $\mathcal{L} \subset \mathbb{R}^n$, we usually consider the standard *Euclidean metric* in $\mathbb{R}^n$. The Euclidean inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ for two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ is given by $\langle \boldsymbol{x}, \boldsymbol{y} \rangle := \sum_{i=1}^n x_i y_i$. It induces the Euclidean (or ℓ_2) norm by

$$\|\boldsymbol{x}\|_2 := \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle} \qquad \text{for any } \boldsymbol{x} \in \mathbb{R}^n.$$

Alternatively, and related to the Lee metric, one can consider the metric induced by the ℓ_1 norm $\|\boldsymbol{x}\|_1 := \sum_{i=1}^n |x_i|$.

A lattice has a *volume* $\mathrm{vol}(\mathcal{L}) := \sqrt{\det(\boldsymbol{B}\boldsymbol{B}^\top)}$, where $\boldsymbol{B}$ is an arbitrary basis of $\mathcal{L}$. Note that the volume is independent of the considered basis despite the given definition because every basis $\widetilde{\boldsymbol{B}}$ has the form $\widetilde{\boldsymbol{B}} = \boldsymbol{U}\boldsymbol{B}$ for a unimodular matrix $\boldsymbol{U} \in \mathrm{GL}_k(\mathbb{Z})$ with determinant $\det(\boldsymbol{U}) = \pm 1$.

The length of any shortest nonzero lattice vector is denoted by

$$\lambda_1(\mathcal{L}) := \min_{\boldsymbol{v} \in \mathcal{L}, \boldsymbol{v} \neq \boldsymbol{0}} \|\boldsymbol{v}\|_2$$

and called the *first minimum* of $\mathcal{L}$. Note that $\lambda_1(\mathcal{L})$ is a *positive* real because of the discrete nature of lattices. For a random lattice of rank k,[4] the first minimum $\lambda_1(\mathcal{L})$ is concentrated around $\mathrm{gh}(\mathcal{L}) := \mathrm{gh}(k) \cdot \mathrm{vol}(\mathcal{L})^{1/k}$ with high probability, where $\mathrm{gh}(k) := \mathrm{vol}(\mathcal{B}_1^k)^{-1/k} \approx \sqrt{k/(2\pi e)}$ is the *Gaussian heuristic* and $\mathcal{B}_1^k$ denotes the unit ball of dimension k. The Gaussian heuristic can be seen as the lattice analog of the Gilbert–Varshamov (GV) bound for codes.

Computing short lattice vectors. The *shortest-vector problem (SVP)* takes a lattice basis $\boldsymbol{B}$ as input and asks to compute a shortest lattice vector, i.e., a $\boldsymbol{v} \in \mathcal{L}(\boldsymbol{B})$ with $\|\boldsymbol{v}\|_2 = \lambda_1(\mathcal{L}(\boldsymbol{B}))$. Finding a shortest vector of a high-dimensional lattice is in general a hard problem. State-of-the-art heuristic algorithms for SVP take both exponential time $2^{0.292k+o(k)}$ and memory $2^{0.208k+o(k)}$ for a k-dimensional lattice. However, finding a *short-enough* vector might suffice to break cryptographic schemes. Lattice-reduction algorithms such as BKZ [34] find such short-enough vectors with a trade-off between the vector length and the computational cost. BKZ requires an oracle that solves SVP *exactly* in β-dimensional lattices for $\beta < k$ and runs in time $2^{0.292\beta+o(k)}$. We say that BKZ runs with *blocksize* β and a higher blocksize heuristically recovers shorter vectors.

Heuristic Claim 1 (BKZ approximation [2]) *For a k-dimensional lattice $\mathcal{L}$, BKZ with blocksize $2 \leq \beta \leq k$ heuristically recovers a lattice vector $\boldsymbol{v} \in \mathcal{L}$ of length*

$$\|\boldsymbol{v}\|_2 \leq \mathrm{gh}(\beta)^{\frac{k-1}{\beta-1}} \cdot \mathrm{vol}(\mathcal{L})^{1/k} \approx \left(\frac{\beta}{2\pi e}\right)^{\frac{k-1}{2(\beta-1)}} \cdot \mathrm{vol}(\mathcal{L})^{1/k}$$

in time $2^{0.292\beta+o(k)}$.

The special cases of the BKZ algorithm are $\beta = 2$, for which BKZ runs in polynomial time but gives an exponentially large approximation radius, and $\beta = k$, for which it runs in exponential time but recovers a shortest lattice vector. If the given lattice is not random, BKZ might be faster. For example, if the lattice contains an unusually short vector, i.e., a vector shorter than what the Gaussian heuristic predicts, BKZ can recover it with a blocksize $\beta \ll k$.

Heuristic Claim 2 (BKZ unusual SVP [2]) *Let $\mathcal{L}$ be a lattice of dimension k and let $\boldsymbol{v} \in \mathcal{L}$ be an unusually short vector with $\|\boldsymbol{v}\|_2 \ll \mathrm{gh}(\mathcal{L})$. Then, BKZ with blocksize β heuristically recovers $\boldsymbol{v}$ if*

$$\sqrt{\frac{\beta}{k}} \cdot \|\boldsymbol{v}\|_2 < \mathrm{gh}(\beta)^{\frac{2\beta-k-1}{\beta-1}} \cdot \mathrm{vol}(\mathcal{L})^{1/k}.$$

For example, if $\|\boldsymbol{v}\|_2 = \mathrm{gh}(\mathcal{L})/\Theta(\sqrt{k})$, then BKZ recovers the unusually short vector with blocksize $\beta = \frac{k}{2} + o(k)$ and thus in time $2^{\frac{0.292k}{2}+o(k)}$ instead of $2^{0.292k+o(k)}$. The above estimate can be refined further, leading to accurate concrete predictions of the precise blocksize β that is needed to recover an unusually short lattice vector [4, 13, 29]. See [3] for a survey on these results.

[4] The meaning of a *random* lattice is a bit more complex than for codes, but can be made rigorous by considering the unique normalized Haar measure on the space of lattices.

2.3 Ideal and log-unit lattices

Recall that an *ideal* $\mathcal{I}$ of a ring R is an additive subgroup of R satisfying $R{\cdot}\mathcal{I} = \mathcal{I}$. Any ideal is already equipped with the additive properties of a lattice, but there is no a-priori notion of a distance. We now discuss two constructions of *ideal lattices*, their relation, and the log-unit lattice.

Coefficient embedding. Firstly, consider polynomial quotient rings of the form $R := \mathbb{Z}[x]/(h)$ for a monic polynomial $h \in \mathbb{Z}[x]$ of degree d. We define the *coefficient embedding* $\mathrm{cf} : R \to \mathbb{Z}^d$, which maps any element $f(x) = \sum_{i=1}^{d} f_i x^{i-1} \in R$ to its coefficient vector $\mathrm{cf}(f) := (f_1, \ldots, f_d) \in \mathbb{Z}^d$. Since $\mathbb{Z}^d \subset \mathbb{R}^d$ is equipped with the Euclidean norm $\|\cdot\|_2$, this mapping induces a metric on R by setting $\|f\| := \|\mathrm{cf}(f)\|_2$ for any $f \in R$. Thus, we call the image $\mathrm{cf}(\mathcal{I}) \subset \mathbb{R}^d$ of any ideal $\mathcal{I} \subset R$ an ideal lattice. We will encounter an example of this construction in the following, where we consider the case $h(x) := x^k - 1$ leading to the quotient ring $R = \mathbb{Z}[x]/(x^k - 1)$. In this setting, the lattice $\mathrm{cf}(\mathcal{I})$ constructed from any principal ideal $\mathcal{I} = (a)$ in R with generator $a = \sum_{i=0}^{k-1} a_i x^i \pmod{x^k - 1}$ has a circulant basis. This derives from the fact that $a, xa, \ldots, x^{k-1}a$ is a $\mathbb{Z}$-basis of $\mathcal{I}$ and multiplication with x^i in R coincides with the i-fold shift of the image vector in $\mathrm{cf}(\mathcal{I})$. More precisely, $\mathrm{cf}(x^i a) = \mathrm{shift}^i(\mathrm{cf}(a))$ holds for $i = 0, \ldots, k-1$ and thus we obtain $\mathrm{cf}(\mathcal{I}) = \mathcal{L}(\mathrm{Shift}(\boldsymbol{a}))$ with $\boldsymbol{a} := \mathrm{cf}(a)$.

Canonical embedding. Secondly, we choose R as the ring of integers $\mathcal{O}_K$ of a number field $K := \mathbb{Q}[x]/(h)$ with $h \in \mathbb{Q}[x]$ being an irreducible monic polynomial of degree d. Note that K is naturally equipped with d field homomorphisms $\sigma_i : K \to \mathbb{C}$ with $i = 1, \ldots, d$ that map x to distinct roots of h in $\mathbb{C}$. We call $\sigma : K \to \mathbb{C}^d$ with $f \mapsto (\sigma_1(f), \ldots, \sigma_d(f))$ for any $f \in K$ the *canonical embedding* of K into $\mathbb{C}^d \simeq \mathbb{R}^{2d}$. We use it to transfer the Euclidean metric from $\mathbb{R}^{2d}$ to K, similar as we did with the coefficient embedding cf in the first construction. Namely, we set $\|f\| := \|\sigma(f)\|_2$ for any $f \in K$, where $\sigma(f)$ on the right-hand side is implicitly understood as an element of $\mathbb{R}^{2d} \simeq \mathbb{C}^d$. Since σ preserves the additive and discrete structure of any nonzero ideal $\mathcal{I} \subset R$, we call the rank-d image $\sigma(\mathcal{I}) \subset \mathbb{C}^d$ an ideal lattice. One particular example of this construction, which we will come across later, is the case of cyclotomic fields. For any odd prime k, we consider the monic *cyclotomic polynomial*

$$\Psi_k(x) := (x^k - 1)/(x - 1) = 1 + x + \ldots + x^{k-1} \in \mathbb{Z}[x]$$

which is irreducible over $\mathbb{Q}$ and has degree $d = k - 1$. We obtain d-dimensional ideal lattices by applying the canonical embedding to nonzero ideals in the ring of integers $\mathcal{O}_k := \mathbb{Z}[x]/(\Psi_k)$ of the *cyclotomic number field* $K := \mathbb{Q}[x]/(\Psi_k)$.

Circulant and cyclotomic ideals. The two examples we considered are in fact closely related. Namely, for a prime k, the quotient ring $\mathbb{Z}[x]/(x^k - 1)$ splits into

$$\mathbb{Z}[x]/(x^k - 1) \cong \mathbb{Z}[x]/(\Psi_k) \times \mathbb{Z}[x]/(x - 1)$$

by means of the isomorphism $f \mapsto (f \pmod{\Psi_k}, f \pmod{x-1})$. As a result, we can map any circulant ideal $\mathcal{I} \subset \mathbb{Z}[x]/(x^k - 1)$ of rank k onto a cyclotomic ideal $\mathcal{I} \pmod{\Psi_k} \subset \mathbb{Z}[x]/(\Psi_k)$ of rank $k-1$, and onto an ideal $\mathcal{I} \pmod{x-1} \subset \mathbb{Z}[x]/(x-1) \cong \mathbb{Z}$ of rank 1. An important property of the first part of the mapping is that the geometry of the ideal lattice $\mathrm{cf}(\mathcal{I})$ via the coefficient embedding is, up to scaling, very close to the ideal lattice $\sigma(\mathcal{I} \pmod{\Psi_k})$ via the canonical embedding [8]. So we can easily move from our circulant lattice to an ideal lattice over a cyclotomic number ring. This will become useful later because some generally hard lattice problems can be solved in quantum-polynomial time for principal ideal lattices over cyclotomic number rings.

Log-unit lattice. We recall one last embedding that is useful for the multiplicative structure of units. Consider again a number ring $\mathcal{O}_K$ of a number field K of degree d with canonical embedding σ given by $\sigma_1, \dots, \sigma_d$. We define the *logarithmic embedding* $\mathrm{Log} : K^\times \to \mathbb{R}^d$ by

$$f \mapsto (\log(|\sigma_1(f)|, \dots, \log(|\sigma_d(f)|)) \in \mathbb{R}^d.$$

Note that multiplication in $K^\times = K \setminus \{0\}$ corresponds to addition after the logarithmic embedding, i.e.,

$$\mathrm{Log}(u \cdot v) = \mathrm{Log}(u) + \mathrm{Log}(v) \quad \text{for all } u, v \in K^\times.$$

Now consider the subgroup of units $\mathcal{O}_K^\times$ of $\mathcal{O}_K$. By Dirichlet's unit theorem the logarithmic embedding $\mathrm{Log}(\mathcal{O}_K^\times)$ is a lattice of a certain rank, named the *log-unit lattice*. Going back to our example, the log-unit lattice $\mathrm{Log}(\mathcal{O}_k^\times)$ of the cyclotomic number field $K := \mathbb{Q}[x]/(\Psi_k)$ has rank $(k-3)/2$ for all odd primes k. Because $|N(u)| := \prod_{i=1}^d |\sigma_i(u)| = 1$ holds for all units $u \in \mathcal{O}_K^\times$, we obtain that the coefficients of $\mathrm{Log}(u)$ sum up to 0. In other words, $\mathrm{Log}(\mathcal{O}_K^\times)$ lies in the subspace orthogonal to $(1, \dots, 1)$. Furthermore, any root of unity $\xi \in K$ satisfies $|\sigma_i(\xi)| = 1$ for all $i = 1, \dots, d$ and thus $\mathrm{Log}(\xi) = \mathbf{0}$, i.e., ξ is contained in the kernel of Log. In particular, Log maps any product $\xi \cdot u$ of a unit $u \in K^\times$ and a root of unity $\xi \in K^\times$ to $\mathrm{Log}(u)$, which means that u can only be recovered from its image $\mathrm{Log}(u)$ up to a root of unity.

3 FuLeeca and lattice-based cryptography

FuLeeca is a hash-and-sign signature scheme based on codes in the Lee metric. The hash-and-sign paradigm is a well-known technique to design digital signatures and has been used for constructions involving both lattices and codes. The main idea is to interpret the hash of the given message as a target point in an ambient space in which a certain selected code or lattice is defined. As the signer knows a suitable *good* basis of the code or the lattice, they can easily find a codeword or a lattice point that is close to the target with respect to the corresponding metric and use it as the signature. Note that even a publicly known *bad* basis, which fully hides the structure of the chosen lattice or code, allows the verifier to efficiently validate the signature by checking if the signature is a lattice point or a codeword and if it is close to the hash of the signed message. The obstacle for an adversary trying to forge a signature is however the hardness of finding a close codeword or lattice point without having access to a good basis. These problems are known as the syndrome-decoding problem (SDP) and the closest-vector problem (CVP), respectively.

Hash-and-sign-based examples from code-based cryptography start with the CFS scheme [11] and include the current NIST proposal Wave [5]. In terms of lattices, the most prominent examples of hash-and-sign signatures are GGH [19], NTRUSign [20], and Falcon [31].

Since the Lee metric is closely related to the Euclidean $\|.\|_2$-norm, not only the overall hash-and-sign structure of FuLeeca coincides with known lattice proposals, but also the underlying notion of distance is similar. We use the remainder of this section to recall the design principles of FuLeeca and describe how it can be interpreted in the context of lattice-based cryptography. This allows us to identify similarities and differences with respect to GGH, NTRUSign, and Falcon. Our observations will enable us to apply lattice techniques to attack FuLeeca later in this paper.

3.1 FuLeeca

The NIST submission of the FuLeeca signature scheme [32] is based on the publication [33]. It comes with three sets of parameters that can be found in Table 2. As the chosen parameter sets correspond to the NIST security levels I, III, and V, we refer to them as FuLeeca-I, FuLeeca-III, and FuLeeca-V, respectively.

Key generation. Both FuLeeca keys are generator matrices of the same quasi-cyclic code $\mathcal{C} \subset \mathbb{F}_p^n$ of dimension k and length $n = 2k$. The secret key $\boldsymbol{G}_{\text{sec}} = (\boldsymbol{A} \mid \boldsymbol{B}) \in \mathbb{F}_p^{k \times n}$ consists of two circulant blocks $\boldsymbol{A} \in \text{GL}(k, \mathbb{F}_p)$ and $\boldsymbol{B} \in \mathbb{F}_p^{k \times k}$, and thus captures the quasi-cyclic structure of $\mathcal{C}$. In contrast, the public key $\boldsymbol{G}_{\text{pub}} = \left(\boldsymbol{I}_k \mid \boldsymbol{A}^{-1}\boldsymbol{B}\right)$ is the systematic form of $\boldsymbol{G}_{\text{sec}}$ and hides the secret key.

As depicted in Algorithm 1, the secret key $\boldsymbol{G}_{\text{sec}}$ is chosen by picking two random signed permutations $\boldsymbol{a}$ and $\boldsymbol{b}$ of a typical Lee vector $\boldsymbol{t} \in \mathbb{F}_p^k$ of Lee weight w_{key} and setting $\boldsymbol{A} := \text{Shift}(\boldsymbol{a})$ and $\boldsymbol{B} := \text{Shift}(\boldsymbol{b})$. The sampling of $\boldsymbol{a}$ is repeated until $\boldsymbol{A}$ is invertible to ensure that the systematic form of $\boldsymbol{G}_{\text{sec}}$, that is, $\boldsymbol{G}_{\text{pub}}$, can be computed.

Algorithm 1: FuLeeca key generation [32].

```
Input  : FuLeeca parameter set, typical Lee vector $\boldsymbol{t}$.
Output: Public key $\boldsymbol{G}_{\text{pub}}$, secret key $\boldsymbol{G}_{\text{sec}}$.

do
    $\boldsymbol{a} \xleftarrow{\$} T(\boldsymbol{t})$
    $\boldsymbol{A} = \text{Shift}(\boldsymbol{a})$
while $\boldsymbol{A} \notin \text{GL}(k, \mathbb{F}_p)$
$\boldsymbol{b} \xleftarrow{\$} T(\boldsymbol{t})$
$\boldsymbol{B} = \text{Shift}(\boldsymbol{b})$
$\boldsymbol{G}_{\text{sec}} = (\boldsymbol{A} \mid \boldsymbol{B})$
$\boldsymbol{G}_{\text{pub}} = (\boldsymbol{I}_k \mid \boldsymbol{A}^{-1}\boldsymbol{B})$
return $\boldsymbol{G}_{\text{pub}}, \boldsymbol{G}_{\text{sec}}$
```

Table 2: Proposed FuLeeca parameter sets [32].

Parameter Set	p	n	w_{key}	w_{sig}/n	s	n_{con}
FuLeeca-I	65,521	1,318	31,102	982.8	3/64	100
FuLeeca-III	65,521	1,982	46,552	982.8	9/256	90
FuLeeca-V	65,521	2,638	61,918	982.8	3/128	178

Signature generation. The signer has access to the secret key $\boldsymbol{G}_{\text{sec}}$ and wishes to sign a message $\boldsymbol{m}$. The signing procedure of FuLeeca is illustrated in Algorithm 2 and starts with applying hash functions and adding randomness to $\boldsymbol{m}$, i.e., with the *hash* part of the hash-and-sign design paradigm. We write $\text{Hash}(\boldsymbol{m}, \mathit{salt})$ to denote the adopted two-stage hashing process, that outputs a vector $\boldsymbol{c} \in \{\pm 1\}^n$ and is described in detail in the FuLeeca specification [32].

Now the *sign* part takes place. It consists of two steps, as is highlighted by the usage of the functions `simpleSign` and `concentrate` in Algorithm 2. These are followed by final checks to decide whether the signature is accepted or the whole process is repeated.

Algorithm 2: FuLeeca signature generation [32].[5]

```
Input  : FuLeeca parameter set, secret key $\boldsymbol{G}_{\text{sec}} \in \mathbb{F}_p^{k \times n}$, message $\boldsymbol{m}$.
Output: Signature $\boldsymbol{v} \in \mathbb{F}_p^n$, salt $\mathit{salt}$.
1  Repeat
2      $\mathit{salt} \xleftarrow{\$} \mathbb{F}_2^{256}$
3      $\boldsymbol{c} = \text{Hash}(\boldsymbol{m}, \mathit{salt}) \in \{\pm 1\}^n$
4      $\boldsymbol{v} = \texttt{simpleSign}(\boldsymbol{c}, \boldsymbol{G}_{\text{sec}})$
5      $\boldsymbol{v} = \texttt{concentrate}(\boldsymbol{c}, \boldsymbol{v}, \boldsymbol{G}_{\text{sec}})$
6      if $w_{sig} - 2w_{key} < \text{wt}_L(\boldsymbol{v}) \leq w_{sig}$
7      and $\text{LMP}(\boldsymbol{v}, \boldsymbol{c}) \geq \lambda + 64$ then
8          return $\mathit{salt}, \boldsymbol{v}$
9      end

10 simpleSign($\boldsymbol{c}$, $\boldsymbol{G}_{\text{sec}}$):
11     for $i = 1, \ldots, k$ do
12         $x_i = \text{trunc}(s \cdot \text{ipm}(\boldsymbol{g}_i, \boldsymbol{c}))$
13     end
14     $\boldsymbol{x} = (x_1, \ldots, x_k)$
15     $\boldsymbol{v} = \boldsymbol{x}\boldsymbol{G}_{\text{sec}}$
16     return $\boldsymbol{v}$

17 concentrate($\boldsymbol{c}$, $\boldsymbol{v}$, $\boldsymbol{G}_{\text{sec}}$):
18     $\mathcal{A} = \{\pm 1, \ldots, \pm k\}$, $\text{lf} = 1$
19     for $j = 1, \ldots, n_{\text{con}}$ do
20         $\boldsymbol{v}' = \boldsymbol{0}$
21         for $i = 1, -1, \ldots, k, -k$ do
22             $\boldsymbol{v}'' = \boldsymbol{v} + \text{sgn}(i) \cdot \boldsymbol{g}_{|i|}$
23             if $|\text{LMP}(\boldsymbol{v}'', \boldsymbol{c}) - (\lambda + 65)| \leq$
                 $|\text{LMP}(\boldsymbol{v}', \boldsymbol{c}) - (\lambda + 65)|$ and
                 $(i \in \mathcal{A}$ or $\text{lf} = 0)$ then
24                 $\boldsymbol{v}' = \boldsymbol{v}''$, $i' = i$
25             end
26         end
27         $\boldsymbol{v} = \boldsymbol{v}'$, $\mathcal{A} = \mathcal{A} \setminus \{-i'\}$
28         if $\text{wt}_L(\boldsymbol{v}) > w_{sig} - w_{key}$ then
29             $\text{lf} = 0$
30         else
31             $\text{lf} = 1$
32         end
33     end
34     return $\boldsymbol{v}$
```

[5] There are significant differences between the specification and the reference implementation of FuLeeca. In particular, the order of the loop on line 21 within the concentration procedure is important for our attack but not properly defined in the specification. Whenever specification and reference implementation differ, we follow the implementation.

Define the number of *sign matches* of two vectors $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{F}_p^n$ and $\boldsymbol{y} = (y_1, \dots, y_n) \in \mathbb{F}_p^n$ as

$$\operatorname{mt}(\boldsymbol{x}, \boldsymbol{y}) := \left|\{i \in \{1, \dots, n\} : \operatorname{sgn}(x_i) = \operatorname{sgn}(y_i) \text{ and } x_i, y_i \neq 0\}\right|.$$

Moreover, the elementwise star product of $\boldsymbol{x}$ and $\boldsymbol{y}$ is $\boldsymbol{x} \star \boldsymbol{y} := (x_1 y_1, \dots, x_n y_n) \in \mathbb{F}_p^n$ and we define

$$\operatorname{ipm}(\boldsymbol{x}, \boldsymbol{y}) := \operatorname{mt}(\boldsymbol{x}, \boldsymbol{y}) - \frac{\operatorname{wt}_H(\boldsymbol{x} \star \boldsymbol{y})}{2}.$$

Note that $2 \cdot \operatorname{ipm}(\boldsymbol{x}, \boldsymbol{y}) = \langle \operatorname{sgn}(\boldsymbol{x}), \operatorname{sgn}(\boldsymbol{y}) \rangle$ applies, where $\langle \cdot, \cdot \rangle$ denotes the Euclidean scalar product. Therefore, we call $\operatorname{ipm}(\boldsymbol{x}, \boldsymbol{y})$ the scaled number of *inner-product matches* of $\boldsymbol{x}$ and $\boldsymbol{y}$.

The first signing step is *simple signing* which maps the hash output $\boldsymbol{c} \in \{\pm 1\}^n$ to a suitable vector $\boldsymbol{x} \in \mathbb{F}_p^k$ depending on the inner-product matches between $\boldsymbol{c}$ and the rows of $\boldsymbol{G}_{\text{sec}}$, i.e., $x_i = \operatorname{trunc}(s \cdot \operatorname{ipm}(\boldsymbol{g}_i, \boldsymbol{c}))$ for some parameter s and $i = 1, \dots, k$. Then, it generates a signature $\boldsymbol{v} = \boldsymbol{x} \boldsymbol{G}_{\text{sec}}$ by interpreting the obtained $\boldsymbol{x}$ as coordinates with respect to the secret basis $\boldsymbol{g}_1, \dots, \boldsymbol{g}_k$ of the code $\mathcal{C} = \langle \boldsymbol{G}_{\text{sec}} \rangle_{\mathbb{F}_p}$. Interestingly, the simple-signing procedure can be expressed as

$$\boldsymbol{v} = \operatorname{trunc}\left(\tfrac{s}{2} \cdot \boldsymbol{c} \cdot \operatorname{sgn}\left(\boldsymbol{G}_{\text{sec}}^{\top}\right)\right) \cdot \boldsymbol{G}_{\text{sec}}.$$

The second signing step is *concentrating* and aims at shifting the Lee weight of the simple signature $\boldsymbol{v}$ and the number of sign matches $\operatorname{mt}(\boldsymbol{c}, \boldsymbol{v})$ to prescribed intervals. This is achieved by a trial-and-error-like process that successively adds or subtracts rows of the secret generator matrix $\boldsymbol{G}_{\text{sec}}$ and then checks if the result was improved. Note that doing this for the i-th row $\boldsymbol{g}_i$ corresponds to adapting the coefficient x_i to $x_i \pm 1 \pmod{p}$ for $i = 1, \dots, k$. The quality of the signature is measured by its Lee weight and the *logarithmic matching probability (LMP)*, which is defined for two fixed vectors $\boldsymbol{v} \in \mathbb{F}_p^n$ and $\boldsymbol{c} \in \{\pm 1\}^n$ as

$$\operatorname{LMP}(\boldsymbol{v}, \boldsymbol{c}) := -\log_2\left(\mathbb{P}\left[\operatorname{mt}(\boldsymbol{v}, \boldsymbol{y}) = \operatorname{mt}(\boldsymbol{v}, \boldsymbol{c})\right]\right).$$

Here, $\boldsymbol{y} \in \{\pm 1\}^n$ is chosen uniformly at random and $\mathbb{P}\left[\operatorname{mt}(\boldsymbol{v}, \boldsymbol{y}) = \operatorname{mt}(\boldsymbol{v}, \boldsymbol{c})\right]$ denotes the probability that $\boldsymbol{v}$ has as many sign matches with $\boldsymbol{y}$ as with $\boldsymbol{c}$. Note that a large value of $\operatorname{LMP}(\boldsymbol{v}, \boldsymbol{c})$ indicates that that number of sign matches $\operatorname{mt}(\boldsymbol{v}, \boldsymbol{c})$ is unusual, i.e., that the number of sign matches is significantly lower or higher than its expectation $\operatorname{wt}_H(\boldsymbol{v} \star \boldsymbol{c})/2$ for random $\boldsymbol{c}$. The behavior of $\operatorname{ipm}(\boldsymbol{c}, \boldsymbol{v})$ is similar, and $\operatorname{ipm}(\boldsymbol{c}, \boldsymbol{v})$ has a large absolute value in this case as well.

The concentration process is repeated n_{con} times, where n_{con} is a preset parameter. Afterward, the obtained signature is accepted if its Lee weight is within the desired interval $(w_{\text{sig}} - 2w_{\text{key}}, w_{\text{sig}}]$ and the LMP between $\boldsymbol{v}$ and $\boldsymbol{c}$ is at least the bit-security level λ plus a security margin of 64 bits.

Signature verification. The verification procedure checks that the signature $\boldsymbol{v}$ is part of the code and has small Lee weight $\operatorname{wt}_L(\boldsymbol{v}) \leq w_{\text{sig}}$. Moreover, the signs of $\boldsymbol{v}$ have to match sufficiently with $\boldsymbol{c}$, i.e., satisfy $\operatorname{LMP}(\boldsymbol{v}, \boldsymbol{c}) \geq \lambda + 64$.

3.2 Interpreting FuLeeca as a lattice-based scheme

We can port the coding-theoretic ideas used in FuLeeca to the lattice setting by applying *construction A* [10, p. 137] to the considered code $\mathcal{C} = \langle \boldsymbol{G}_{\text{sec}} \rangle_{\mathbb{F}_p} \subset \mathbb{F}_p^n$ of length n and dimension k. Namely, we consider the lattice

$$\mathcal{L}_1 := \mathcal{C} + p\mathbb{Z}^n = \{\boldsymbol{v} \in \mathbb{Z}^n : \boldsymbol{v} \pmod{p} \in \mathcal{C}\} \subset \mathbb{R}^n. \tag{1}$$

Recall that we identify $\mathbb{F}_p$ with $\left\{-\frac{p-1}{2}, \dots, \frac{p-1}{2}\right\}$ and that we implicitly lift the entries of vectors and matrices from $\mathbb{F}_p$ to $\mathbb{Z}$. The public generator matrix $\boldsymbol{G}_{\text{pub}} = \left(\mathbf{I}_k \mid \boldsymbol{A}^{-1}\boldsymbol{B}\right)$ of $\mathcal{C}$, whose right side $\boldsymbol{A}^{-1}\boldsymbol{B}$ is computed modulo p, gives a public basis $\boldsymbol{B}_{\text{pub}}$ of $\mathcal{L}_1$. Namely,

$$\boldsymbol{B}_{\text{pub}} = \begin{pmatrix} \mathbf{I}_k & \boldsymbol{A}^{-1}\boldsymbol{B} \\ \mathbf{0} & p\,\mathbf{I}_{n-k} \end{pmatrix}$$

and this shows that $\mathcal{L}_1$ has full rank n and volume $\text{vol}(\mathcal{L}_1) = \det(\boldsymbol{B}_{\text{pub}}) = p^{n-k}$. The normalized volume $\text{vol}(\mathcal{L}_1)^{1/n} = p^{1-r}$ of the lattice is thus determined by the modulus p and the code rate $r := \frac{k}{n}$, whereas the rank of $\mathcal{L}_1$ equals the length of the code $\mathcal{C}$, as $p\mathbb{Z}^n \subset \mathcal{L}_1$ applies by construction. Since we will make use of different lattices throughout the paper, Figure 2 contains an overview of them, visualizes their connections, and indicates how to obtain suitable bases.

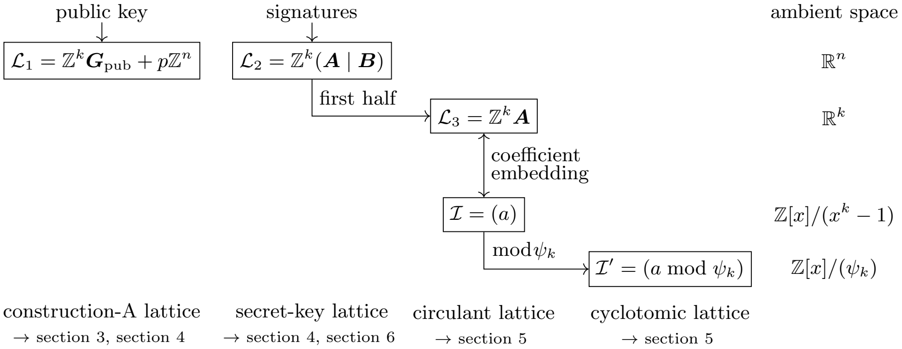


Fig. 2: Schematic overview of the lattices used in this paper and their connections.

Observe that the Lee metric on $\mathcal{C}$ is closely related to the ℓ_1 metric on $\mathcal{L}_1$. In particular, lifting a vector $\boldsymbol{x} \in \mathcal{C}$ with $\text{wt}_L(\boldsymbol{x}) = w$ implicitly to $\boldsymbol{x} \in \mathcal{L}_1$ yields $\|\boldsymbol{x}\|_1 = w$. The ℓ_1 norm is in fact close to the Euclidean ℓ_2 metric, to which most lattice algorithms are tailored. More precisely,

$$\|\boldsymbol{y}\|_2 \leq \|\boldsymbol{y}\|_1 \leq \sqrt{n} \cdot \|\boldsymbol{y}\|_2$$

holds for all $\boldsymbol{y} \in \mathbb{R}^n$ and we expect $\|\boldsymbol{y}\|_1 \sim \|\boldsymbol{y}\|_2$ for very sparse vectors and $\|\boldsymbol{y}\|_1 \sim \sqrt{n} \cdot \|\boldsymbol{y}\|_2$ for dense and balanced vectors. Short or close codewords of $\mathcal{C}$ in the Lee metric are therefore directly related to short or close lattice vectors of $\mathcal{L}_1$ in the Euclidean metric.

In particular, we know that the short rows of the generator matrix $\boldsymbol{G}_{\mathrm{sec}}$ of $\mathcal{C}$ correspond to k short vectors $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k$ in $\mathcal{L}_1$. As each $\boldsymbol{g}_i$ has the form $\left(\mathrm{shift}^i(\boldsymbol{a}) \mid \mathrm{shift}^i(\boldsymbol{b})\right)$ and $\boldsymbol{a}$ and $\boldsymbol{b}$ are signed permutations of a precomputed typical vector $\boldsymbol{t}$, we know that $\|\boldsymbol{g}_i\|_2 = \|\boldsymbol{g}\|_2 = \sqrt{2} \cdot \|\boldsymbol{t}\|_2$ applies for $\boldsymbol{g} := (\boldsymbol{a} \mid \boldsymbol{b})$ and all $i = 1, \ldots, k$. As can be seen in Table 3, this is close to the Gaussian heuristic of $\mathcal{L}_1$, i.e., the expected first minimum of the lattice. The *good* generator matrix $\boldsymbol{G}_{\mathrm{sec}}$ thus gives a *good* basis of the sublattice $\mathcal{L}(\boldsymbol{G}_{\mathrm{sec}}) \subset \mathcal{L}_1$.

FuLeeca uses this good basis to compute a signature $\boldsymbol{v}$ that is short in the ℓ_1 norm and has a large absolute inner-product matching $|\mathrm{ipm}(\boldsymbol{v}, \boldsymbol{c})| = \frac{1}{2}|\langle \mathrm{sgn}(\boldsymbol{v}), \mathrm{sgn}(\boldsymbol{c})\rangle|$ with the hash $\boldsymbol{c} \in \{\pm 1\}^n$. Note that for a balanced vector $\boldsymbol{v} = (\pm v, \ldots, \pm v)$ the inner-product matching correlates perfectly with the actual Euclidean inner product, i.e.,

$$\langle \boldsymbol{v}, \boldsymbol{c} \rangle = v \cdot \langle \mathrm{sgn}(\boldsymbol{v}), \mathrm{sgn}(\boldsymbol{c}) \rangle = 2v \cdot \mathrm{ipm}(\boldsymbol{v}, \boldsymbol{c}).$$

More generally, we expect $\langle \boldsymbol{v}, \boldsymbol{c} \rangle \sim \frac{2\|\boldsymbol{v}\|_2}{\sqrt{n}} \cdot \mathrm{ipm}(\boldsymbol{v}, \boldsymbol{c})$ for reasonably balanced vectors $\boldsymbol{v}$. The large absolute inner-product matching for a FuLeeca signature thus roughly corresponds to a large absolute Euclidean inner product between the signature $\boldsymbol{v}$ and $\boldsymbol{c}$, i.e., $\boldsymbol{v}$ and $\boldsymbol{c}$ have a small angle. Furthermore, this is equivalent to saying that $\boldsymbol{v}$ or $-\boldsymbol{v}$ is a close vector to $\frac{\|\boldsymbol{v}\|_2}{\|\boldsymbol{c}\|_2} \cdot \boldsymbol{c}$. So in terms of the Euclidean metric, a FuLeeca signature $\boldsymbol{v}$ is valid when it is simultaneously small and close to the target obtained from the message.

Table 3: Lengths of the short secret-key vectors relative to the Gaussian heuristic.

Parameter Set	$\mathrm{wt}_L(\boldsymbol{g})$	$\|\boldsymbol{g}\|_2$	$\|\boldsymbol{g}\|_2/\,\mathrm{gh}(\mathcal{L}_1)$	$\|\boldsymbol{g}\|_2/\,\mathrm{gh}(\mathcal{L}_2)$
FuLeeca-I	62,204	2,385.06	1.061	0.184
FuLeeca-III	93,104	2,976.61	1.079	0.150
FuLeeca-V	123,836	3,478.06	1.093	0.130

3.3 Comparing FuLeeca with known lattice-based schemes

We first compare FuLeeca to early lattice-based hash-and-sign signature schemes such as GGH [19] and NTRUSIGN [20]. GGH and NTRUSIGN both sign by hashing the message to a target in the space, and then using a good secret basis to compute a nearby lattice point. Recall from subsection 3.2 that a FuLeeca signature can also be interpreted as a lattice point close to some target hash. For FuLeeca however, the signature vector is also short, similar as in the recent lattice-based signature scheme HAWK [9, 15].

NTRUSIGN relies on structured NTRU lattices over the circulant quotient ring $\mathbb{Z}[x]/(x^k - 1)$, which correspond precisely to the lattice $\mathcal{L}_1$ in FuLeeca up to the choice of parameters. So both schemes use the same structure to achieve efficiency. Where FuLeeca uses a good basis $\boldsymbol{G}_{\mathrm{sec}}$ of the sublattice $\mathcal{L}(\boldsymbol{G}_{\mathrm{sec}}) \subset \mathcal{L}_1$ to compute a nearby lattice point, NTRUSIGN and also FALCON [31] extend $\boldsymbol{G}_{\mathrm{sec}}$ to a full good basis $\boldsymbol{B}_{\mathrm{sec}}$ of the lattice $\mathcal{L}_1$ and use $\boldsymbol{B}_{\mathrm{sec}}$ for signature generation.

Unfortunately, both GGH and NTRUSign were broken by learning attacks, as each signature leaked some information about the secret key. For example, in the NTRUSign scheme the error $\boldsymbol{e}$ between the target and the signature lattice vector is uniform over the parallelepiped $\left[-\frac{1}{2}, \frac{1}{2}\right]^n \cdot \boldsymbol{B}_{\text{sec}}$ of the secret basis $\boldsymbol{B}_{\text{sec}}$. Given enough signatures, it is possible to learn the parallelepiped, i.e., to recover the basis $\boldsymbol{B}_{\text{sec}}$ [27]. For example, the computation $\mathbb{E}\left[\boldsymbol{e}^\top \boldsymbol{e}\right] = \frac{1}{12}\boldsymbol{B}_{\text{sec}}\boldsymbol{B}_{\text{sec}}^\top$ shows that the signatures leak the Gram matrix of the secret basis $\boldsymbol{B}_{\text{sec}}$. The full basis can be recovered by looking at higher moments.

Later variants of NTRUSign tried to mitigate learning attacks in various ways, for example by adding some extra noise to the decoding procedure. Unfortunately, these variants generally fell victim to similar learning attacks [14, 37], as the signatures still somehow depend directly on the secret information. Even recent hash-and-sign schemes, such as the submission Peregrine [35] in the Korean post-quantum-cryptography competition, are vulnerable to learning attacks [23]. In section 6, we show that FuLeeca has similar vulnerabilities.

An important breakthrough in hash-and-sign signature schemes was the GPV framework [17], that cleverly samples the nearby lattice point from a discrete Gaussian distribution around the target, using the secret basis. Because this distribution only depends on the lattice, they *proved* that the signatures do not leak any information about the secret basis, therefore mitigating any learning attack. Falcon [31], the lattice-based signature scheme that will be standardized by NIST [25], combines the hash-and-sign technique of NTRUSign and the GPV framework to prevent learning attacks. Similarly, the code-based hash-and-sign scheme Wave follows the GPV framework with ternary codes in the Hamming metric and uses appropriate rejection sampling to obtain a signature distribution that is independent of the secret code structure. In contrast, FuLeeca does not follow the GPV framework, and generally it remains an open question how to adapt the GPV framework to the Lee metric.

4 A leaked-sublattice attack

Recall that construction A allows to set up the lattice $\mathcal{L}_1 = \mathcal{C} + p\mathbb{Z}^n$ from the code $\mathcal{C}$ spanned by the rows of the public generator matrix $\boldsymbol{G}_{\text{pub}} \in \mathbb{F}_p^{k\times n}$ as described in equation (1). As we explained in subsection 3.2, the rows $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k$ of the secret generator matrix $\boldsymbol{G}_{\text{sec}}$ are short in terms of the $\|.\|_2$-norm in the rank-n lattice $\mathcal{L}_1$, i.e., they are between 1.06 and 1.1 times longer than the expected first minimum of $\mathcal{L}_1$. Following Heuristic Claim 1, the BKZ lattice-reduction algorithm heuristically finds vectors of similar $\|.\|_2$-norm in time $2^{0.292\beta+o(n)}$ for $\beta \geq 0.95n$. Since the FuLeeca parameters are chosen such that the secret vectors $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k$ are *not* unusually short in $\mathcal{L}_1$, we cannot apply Heuristic Claim 2 to speed up the recovery.

The described approach is the only lattice-based attack that was considered in the FuLeeca specification and the security level was mainly determined by the cost of more efficient information-set-decoding (ISD) attacks [32, section 6]. In the following, we show that the transition to a suitable lower-dimensional sublattice of $\mathcal{L}_1$ allows to achieve better attack complexities for lattice-reduction attacks. They outperform ISD attacks and lower the security for all parameter sets significantly.

4.1 Constructing a sublattice with unusually short vectors

The goal of this subsection is to describe a lower-dimensional sublattice $\mathcal{L}_2$ of $\mathcal{L}_1$ that still contains the secret vector $\boldsymbol{g} := (\boldsymbol{a} \mid \boldsymbol{b})$ and its quasi-circular shifts. This will lower the cost of recovering short vectors using BKZ. Moreover, we will see that the Euclidean norms of $\boldsymbol{g}_1, \dots, \boldsymbol{g}_k$ are unusually small for $\mathcal{L}_2$, which makes the vectors even easier to recover.

Recall that each FuLeeca signature is given by a vector $\boldsymbol{v} = \boldsymbol{x}\boldsymbol{G}_{\text{sec}} \pmod p$ of low Lee weight. However, we observe experimentally that during the signature generation the coefficients stay small enough to *not* be reduced modulo p. This means that the equality $\boldsymbol{v} = \boldsymbol{x}\boldsymbol{G}_{\text{sec}}$ actually also holds over $\mathbb{Z}$, that is, if we interpret $\boldsymbol{v}, \boldsymbol{x} \in \left\{-\frac{p-1}{2}, \dots, \frac{p-1}{2}\right\}^n \subset \mathbb{Z}^n$ as integer vectors. Thus, all signature vectors belong to the lattice

$$\mathcal{L}_2 := \left\{ \sum_{i=1}^{k} z_i \boldsymbol{g}_i : z_i \in \mathbb{Z} \quad \forall i \right\} \subset \mathbb{R}^n \tag{2}$$

that is spanned by the rows of $\boldsymbol{G}_{\text{sec}}$.

We generated 5 million signatures per parameter set, which were split evenly over 50 FuLeeca key pairs in every case. All signature vectors belonged to $\mathcal{L}_2$, which experimentally confirms the above observation. Furthermore, the largest observed coefficient size was 7,356, 8,564, and 7,616 for FuLeeca-I, FuLeeca-III, and FuLeeca-V instances, respectively. This is significantly smaller than $\frac{p-1}{2} =$ 32,760. The described behavior of non-wrapping coefficients is explained by the fact that the simple-signing procedure creates an initial signature vector that is a small combination of the vectors $\boldsymbol{g}_1, \dots, \boldsymbol{g}_k$, which have small coefficients. As a result, no reduction modulo p takes place in simple signing. The concentration phase that follows adds vectors $\pm\boldsymbol{g}_i$ for $i = 1, \dots, k$ to the simple signature but furthermore tries to keep its Lee weight and thus its coefficients small, again leading to not wrapping the coefficients modulo p.

It is initially not clear that an attacker has access to a basis of $\mathcal{L}_2$ because $\boldsymbol{G}_{\text{sec}}$ is secret and the knowledge of $\boldsymbol{G}_{\text{pub}}$ only allows to recover a basis of $\mathcal{L}_1$ as described in subsection 3.2. However, a small sample of FuLeeca signatures is enough to recover a basis of $\mathcal{L}_2$, which will be explained in subsection 4.2. Let us for now assume that we know a basis of $\mathcal{L}_2$ and describe how to recover the secret vector $\boldsymbol{g}$ or one of its quasi-circular shifts under this assumption.

The lattice $\mathcal{L}_2$ has only rank $k = n/2$ and still contains the short vectors $\boldsymbol{g}_1, \dots, \boldsymbol{g}_k$. We can thus focus on the rank-k lattice $\mathcal{L}_2$ for the recovery of a secret vector. Compared to the rank-n lattice $\mathcal{L}_1$, finding a shortest vector in $\mathcal{L}_2$ gives a reduced complexity of $2^{0.292k+o(k)}$.

Additionally, the vector $\boldsymbol{g}$ is in fact an unusually short vector in $\mathcal{L}_2$. Namely, $\boldsymbol{g}$ is about a factor $\Theta(\sqrt{n})$ shorter than any vector one would expect in a lattice of this rank and volume. This can be seen from the ratio of $\|\boldsymbol{g}\|_2$ and the Gaussian heuristic $\mathrm{gh}(\mathcal{L}_2)$ of $\mathcal{L}_2$ in Table 3. As a result, a quasi-circular shift of $\boldsymbol{g}$ can be recovered by the BKZ algorithm with $\beta = \frac{k}{2} + o(k) = \frac{n}{4} + o(n)$, leading to a time complexity of about $2^{\frac{0.292n}{4} + o(n)}$. In other words, we obtain a quartic speedup over the lattice attack considered in the FuLeeca specification [32]. A more precise value of β can be computed with concrete estimation scripts,[6] leading to $\beta = 291$, $\beta = 448$, and $\beta = 603$ for the FuLeeca-I, FuLeeca-III, and FuLeeca-V parameters, respectively. This significantly reduces the claimed security levels of the selected parameter sets as shown in Table 4.

Table 4: Estimated security levels of the proposed FuLeeca parameter sets, claims according to [32, Table 1]. The new cost is based on the new blocksize estimate combined with the state-of-the-art cost model of [24] as implemented in the lattice estimator of [4].

Parameter Set	Claimed Security Level (in bits)	New Security Level (in bits)	Blocksize (β)
FuLeeca-I	160	**111**	291
FuLeeca-III	224	**155**	448
FuLeeca-V	288	**199**	603

4.2 Extracting the sublattice from FuLeeca signatures

The previous subsection showed that it is substantially cheaper to attack the sublattice $\mathcal{L}_2$ instead of $\mathcal{L}_1$. But as the attacker does not have access to $\boldsymbol{G}_{\text{sec}}$, it is not trivial to get a basis of $\mathcal{L}_2$. We now explain in more detail how to recover the lattice $\mathcal{L}_2$ from a few signatures.

Assuming signatures are not reduced modulo p, any signature vector $\boldsymbol{v} = (\boldsymbol{w} \mid \boldsymbol{y})$ defines a generating matrix $\boldsymbol{V} := (\mathrm{Shift}(\boldsymbol{w}) \mid \mathrm{Shift}(\boldsymbol{y}))$ of the sublattice $\mathcal{L}_{\boldsymbol{v}} := \mathcal{L}(\boldsymbol{V}) \subset \mathcal{L}_2$. Now, for r signatures $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r$, these generating matrices $\boldsymbol{V}_1, \ldots, \boldsymbol{V}_r$ generate the sublattice

$$\mathcal{L}_{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_r} := \mathcal{L}_{\boldsymbol{v}_1} + \ldots + \mathcal{L}_{\boldsymbol{v}_r} := \left\{ \sum_{i=1}^{r} \boldsymbol{u}_i : \boldsymbol{u}_i \in \mathcal{L}_{\boldsymbol{v}_i} \quad \forall i \right\} \subset \mathcal{L}_2,$$

and for large enough r we can expect that in fact the equality $\mathcal{L}_{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_r} = \mathcal{L}_2$ holds. The Hermite normal form (HNF) is the analog of the echelon form for matrices with integer coefficients and can be used to efficiently extract a basis $\boldsymbol{B}_r$ of $\mathcal{L}_{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_r}$ from all its generating vectors. More precisely, one has $\boldsymbol{B}_r = \mathrm{HNF}((\boldsymbol{V}_1; \ldots; \boldsymbol{V}_r))$, where the zero rows of the HNF are removed and the matrices $\boldsymbol{V}_i$ are stacked vertically. To speed this process up in practice, it can be helpful to compute the HNF incrementally by adding $\boldsymbol{V}_1, \ldots, \boldsymbol{V}_r$ step by step.

[6] See the file `estimates/estimate_reduction.sage` available at `https://github.com/WvanWoerden/FuLeakage/`.

The remaining question is how large r has to be to guarantee the equality $\mathcal{L}_{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_r} = \mathcal{L}_2$ with overwhelming probability. Note that $\mathcal{L}_2$ is a rank-k lattice in a $2k$-dimensional space, and that the rank remains the same if we restrict ourselves to the first k coefficients of each vector. In other words, we do not consider $\mathcal{L}_2$ generated by the quasi-circular shifts of $\boldsymbol{g}$ anymore, but the lattice generated by $\mathrm{Shift}(\boldsymbol{a})$. Recall from the preliminaries that this is in fact an ideal lattice $\mathcal{I} = (a) \subset \mathbb{Z}[x]/(x^k - 1)$ with generator $a = \sum_{i=1}^{k} a_i x^{i-1}$ corresponding to the secret vector $\boldsymbol{a} = (a_1, \ldots, a_k)$. The question then becomes how many random elements we need to sample from $\mathcal{I}$ such that they generate the full ideal.

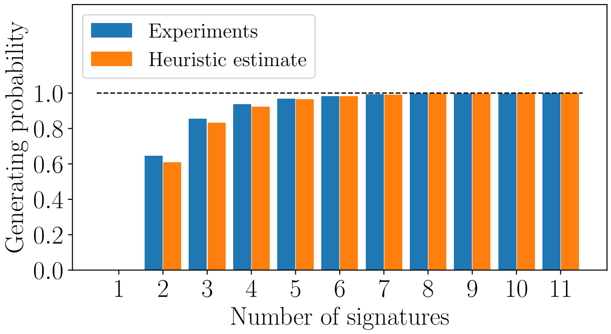


Fig. 3: Number of signatures needed to fully generate the lattice $\mathcal{L}_2$, leaving out signatures whose coefficients sum up to 0 (about 2.5% of the total).

This question has already been explored for an ideal over a number ring $\mathcal{O}_K \subset K$ and decent random distributions for sampling. In this scenario, the probability that r random elements fully generate the ideal is heuristically roughly $\zeta_K(r)^{-1}$, where ζ_K is the Dedekind zeta function of the number field K. See [16, Lemma 2.7] for a provable lower bound of $(4\zeta_K(r))^{-1}$. We have seen that $\mathbb{Z}[x]/(x^k - 1)$ splits into two number rings, a cyclotomic number ring $R_1 := \mathbb{Z}[x]/(\Psi_k)$ of degree $k-1$, and a number ring $R_2 := \mathbb{Z}[x]/(x-1) \cong \mathbb{Z}$ of degree 1. In order to generate an ideal $\mathcal{I} \subset \mathbb{Z}[x]/(x^k-1)$, we need to generate its images $\mathcal{I}_1 := \mathcal{I} \pmod{\Psi_k}$ and $\mathcal{I}_2 := \mathcal{I} \pmod{x-1}$ in R_1 and R_2, respectively. Thus, we expect a probability of roughly $\zeta_{R_1}(r)^{-1} \cdot \zeta_{R_2}(r)^{-1}$. For prime cyclotomic number fields of large degree k, $\zeta_{R_1}(2)$ is close to 1 already and converges quickly for growing r. Concretely, we observe $\zeta_{R_1}(2) \leq 1.0024$ for primes $100 < k < 300$ and we obtain the upper bound $\zeta_{R_1}(r) \leq \exp(O(p^{\frac{3}{2}-r}))$ following the proof of [1, Lemma 1]. Thus, the probability is mostly dominated by $\zeta_{R_2}(r) = \zeta_{\mathbb{Z}}(r)$, which is the Riemann zeta function $\zeta(r)$. Given that $\zeta_{R_2}(r) \leq 1 + 3 \cdot 2^{-r}$ for $r \geq 2$, we can simply pick, say, $r \geq 100$ elements to guarantee that we generate the full ideal with all but negligible probability. We exclude the signatures that are $0 \in R_2$, i.e., those that have coefficients summing up to 0, as they can never help generating the ideal in R_2, and such elements should be ignored for the Dedekind zeta estimate to hold.

In Figure 3, we verify the derived estimate experimentally over 1,000 trials. All ideals were successfully generated by at most 11 signatures.

To conclude, a few dozen signatures are enough to recover a basis for the lattice $\mathcal{L}_2$ containing the unusually short secret vectors $\boldsymbol{g}_1, \dots, \boldsymbol{g}_k$. Then, by the BKZ lattice-reduction algorithm, we can recover any such $\boldsymbol{g}_i$ in time $2^{\frac{0.292k}{2}+o(k)}$, pushing the attack cost significantly below the claimed security levels as shown in Table 4.

This attack could be mitigated by adopting the basis-extension strategy from e.g. NTRUSIGN [20] and FALCON [31]. Namely, their signature generation does not only use the secret good basis $\boldsymbol{G}_{\text{sec}}$ but its extension $\boldsymbol{B}_{\text{sec}}$ that generates $\mathcal{L}_1$. This makes sure that the generated signatures are distributed in the whole lattice $\mathcal{L}_1$ and do not only lie in the sublattice $\mathcal{L}_2$.

5 A polynomial-time quantum attack

In this section, we present a polynomial-time quantum attack on all security levels of FuLeeca, hence breaking the post-quantum claims of the scheme. We show that the leaked lattice $\mathcal{L}_2$ from the previous section can be transformed into a related principal-ideal lattice over the cyclotomics with an unusually short generator linked to the secret key. For this family of ideals, there exists a polynomial-time quantum algorithm to recover the unusually short generator [7, 12]. Thus, we can recover the secret key in quantum-polynomial time.

5.1 Constructing a circulant ideal lattice

Recall that the lattice $\mathcal{L}_2$ defined in equation (2) is generated by the rows of $\boldsymbol{G}_{\text{sec}} = (\boldsymbol{A} \mid \boldsymbol{B})$ and that the blocks $\boldsymbol{A} = \text{Shift}(\boldsymbol{a})$ and $\boldsymbol{B} = \text{Shift}(\boldsymbol{b})$ are circulant matrices. Let us consider the lattice

$$\mathcal{L}_3 := \left\{ \sum_{i=1}^{k} z_i \boldsymbol{a}_i : z_i \in \mathbb{Z} \quad \forall i \right\} \subset \mathbb{R}^k$$

that is spanned by the rows $\boldsymbol{a}_i = \text{shift}^{i-1}(\boldsymbol{a})$ of $\boldsymbol{A}$. We can recover a basis of $\mathcal{L}_3$ from any basis of $\mathcal{L}_2$ simply by considering only the first half of each basis vector. Since this lattice is circulant, it can be represented by an ideal $\mathcal{I}$ in the quotient ring $\mathbb{Z}[x]/(x^k - 1)$ with $k = n/2$. More precisely, we can identify any vector $(f_1, \dots, f_k) \in \mathcal{L}_3$ with the element $f(x) = \sum_{i=1}^{k} f_i x^{i-1} \in \mathbb{Z}[x]/(x^k - 1)$ by means of the coefficient embedding discussed in subsection 2.3. Recall that the circular shift of a lattice vector coincides precisely with the multiplication of its polynomial representation by x modulo $x^k - 1$. Further, $\mathcal{L}_3$ corresponds precisely to the principal ideal $(a) \subset \mathbb{Z}[x]/(x^k - 1)$ generated by $a(x) = \sum_{i=1}^{k} a_i x^{i-1}$, where $\boldsymbol{a} = (a_1, \dots, a_k)$ is the secret vector. As $\mathcal{I}$ is closed under multiplication with x, note that in fact every $x^i a$ for $i = 0, \dots, k-1$ generates $\mathcal{I}$ and shares its short Euclidean norm with a. Thus, it is enough to recover one of these short generators to find a shift of the secret vector $\boldsymbol{a}$.

We will use a quantum algorithm to recover a short generator of a principal ideal in a cyclotomic number ring, but so far we are working over the quotient $\mathbb{Z}[x]/(x^k-1)$. Therefore, we switch to the image of $\mathcal{I}$ in a suitable, slightly smaller, cyclotomic number ring. Namely, choose the cyclotomic polynomial

$$\psi_k(x) := (x^k-1)/(x-1) = 1 + x + \ldots + x^{k-1}$$

and consider the cyclotomic number ring $\mathcal{O}_k := \mathbb{Z}[x]/(\psi_k) \subset \mathbb{Q}[x]/(\psi_k)$, as discussed in subsection 2.3. Any element f of $\mathbb{Z}[x]/(x^k-1)$ can be transferred to $\mathcal{O}_k$ by the ring homomorphism $f \pmod{x^k-1} \mapsto f \pmod{\psi_k}$. This mapping preserves the ideal structure of $\mathcal{I} \subset \mathbb{Z}[x]/(x^k-1)$ and its image in $\mathcal{O}_k$ is the principal ideal $\mathcal{I}' := \mathcal{I} \pmod{\psi_k}$ that is generated by $a \pmod{\psi_k}$, or, more generally, by any $x^i a \pmod{\psi_k}$ for $i = 0, \ldots, k-1$. In the case of cyclotomic rings, this transformation only changes the geometry slightly, i.e., polynomials with small coefficients in $\mathcal{I}$ are generally still small in the ideal lattice $\mathcal{I}'$ under the canonical embedding [8].

5.2 Quantumly recovering the secret key from the ideal lattice

We now exploit the leaked lattice $\mathcal{L}_3$ to fully recover the secret key in quantum-polynomial time. In particular, we make use of a quantum algorithm which can recover a shortest generator of any principal ideal within a cyclotomic number ring that has an exceptionally short generator [7, 12].

As we have seen in the previous subsection, we can identify $\mathcal{L}_3$ with a principal ideal $\mathcal{I}$ in $\mathbb{Z}[x]/(x^k-1)$ and any short generator $x^i a$ of $\mathcal{I}$ corresponds to a shift of the secret vector $\boldsymbol{a}$. Furthermore, we can easily map $\mathcal{I}$ to a principal ideal $\mathcal{I}'$ in the cyclotomic number ring $\mathcal{O}_k = \mathbb{Z}[x]/(\psi_k)$. The strategy to recover a suitable short generator $x^i a$ of $\mathcal{I}$, and thus a shift of the secret vector $\boldsymbol{a}$, is to firstly find a principal generator c of $\mathcal{I}'$, secondly transform it into a *short* generator a' of $\mathcal{I}'$, and thirdly lift it back to a generator of $\mathcal{I}$ in the quotient ring $\mathbb{Z}[x]/(x^k-1)$. Figure 2 visualizes the relations between the involved lattices.

Recovering a principal generator of $\mathcal{I}'$. Since we can obtain a basis of $\mathcal{L}_3$ from any basis of $\mathcal{L}_2$ and the latter can be computed from a small sample of FuLeeca signatures as described in subsection 4.2, we have access to a set of generators $c_1, \ldots, c_s$ of $\mathcal{I}$. Their images $c_1 \pmod{\psi_k}, \ldots, c_s \pmod{\psi_k}$ then span $\mathcal{I}' \subset \mathcal{O}_k$. Recovering a principal generator of $\mathcal{I}'$ from these generators classically takes sub-exponential time, but there is a quantum algorithm achieving that in only polynomial quantum time [7]. We can thus efficiently obtain a generator $c \in \mathcal{I}'$ with $\mathcal{I}' = (c)$.

Shortening the principal generator of $\mathcal{I}'$. Next, we transform c into a short generator a' of $\mathcal{I}'$. Since any two generators of $\mathcal{I}'$ differ by multiplication with a unit in $\mathcal{O}_k$, there is a unit $u \in \mathcal{O}_k^\times$ satisfying $c = u \cdot a'$. This equality reads as

$$\operatorname{Log}(c) = \operatorname{Log}(u) + \operatorname{Log}(a')$$

in terms of the logarithmic embedding, which was discussed in subsection 2.3. Note that $\mathrm{Log}(u)$ lies in the log-unit lattice $\mathrm{Log}(\mathcal{O}_k^\times)$ by definition, while $\mathrm{Log}(c)$ and $\mathrm{Log}(a')$ are contained in the larger set $\mathrm{Log}(K^\times)$ but in general not in $\mathrm{Log}(\mathcal{O}_k^\times)$. In fact, $\mathrm{Log}(c)$ and $\mathrm{Log}(a')$ do not even need to lie in the real span of the log-unit lattice, which is contained in the subspace orthogonal to the all-one vector $\mathbf{1} = (1, \dots, 1) \in \mathbb{R}^{k-1}$. This originates from $\mathrm{Log}(\mathcal{O}_k^\times)$ not being full-rank in $\mathrm{Log}(K^\times)$.[7]

As we want to make use of decoding properties of the log-unit lattice, we consider the orthogonal projection $\pi : \mathbb{R}^{k-1} \to \mathrm{span}\left(\mathrm{Log}(\mathcal{O}_k^\times)\right)$ onto the log-unit lattice and transform the above equality into

$$\pi\left(\mathrm{Log}(c)\right) = \mathrm{Log}(u) + \pi\left(\mathrm{Log}(a')\right). \tag{3}$$

As illustrated in Figure 4, this can now be interpreted as a decoding problem in the log-unit lattice, where $\pi\left(\mathrm{Log}(c)\right)$ is the target, $\mathrm{Log}(u)$ the desired lattice element, and $\pi\left(\mathrm{Log}(a')\right)$ the error. If we solve this decoding problem and recover $\mathrm{Log}(u)$, we can also compute the unit $u \in \mathcal{O}_k$ and thus get a short generator $a' = c/u$ of $\mathcal{I}'$, as desired.

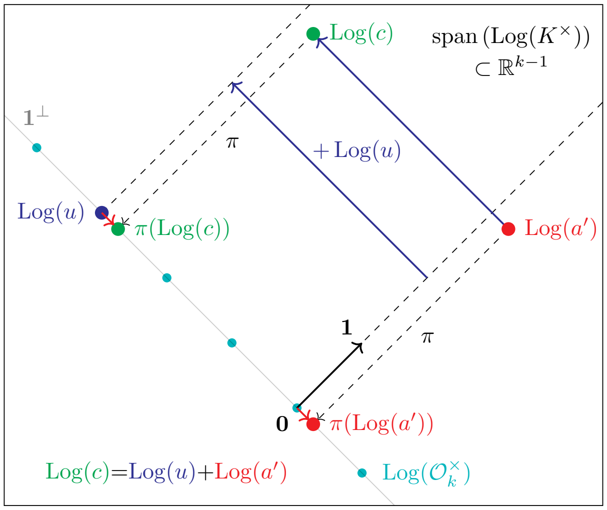


Fig. 4: Visualization of the decoding problem $\pi\left(\mathrm{Log}(c)\right) = \mathrm{Log}(u) + \pi\left(\mathrm{Log}(a')\right)$ after projecting onto the log-unit lattice.

Decoding in the log-unit lattice. Let us now explain why equation (3) indeed gives us a decoding problem in the log-unit lattice $\mathrm{Log}(\mathcal{O}_k^\times)$ that we can

[7] $\mathrm{Log}(K^\times)$ is not full-rank in $\mathbb{R}^{k-1}$ either, as the complex embeddings come in conjugate pairs. However, we are only concerned with the fact that $\mathrm{Log}(\mathcal{O}_k^\times)$ is not full-rank in $\mathrm{Log}(K^\times)$.

decode efficiently to the unique solution $\mathrm{Log}(u)$. We will also provide experimental evidence that this is indeed the case for the FuLeeca setting at the end of the section. For uniqueness, and later for efficient decoding, we need the error $\|\pi\left(\mathrm{Log}(a')\right)\|_2$ to be *small*. Then the target $\pi\left(\mathrm{Log}(c)\right)$ lies exceptionally close to the log-unit lattice vector $\mathrm{Log}(u)$, and far away from the rest of the log-unit lattice.

We argue intuitively why the error $\|\pi\left(\mathrm{Log}(a')\right)\|_2$ is indeed small. Note first that the algebraic norm $N(a') := \prod_{i=1}^{k-1} |\sigma_i(a')|$ coincides for any principal generator of $\mathcal{I}'$, as every unit u satisfies $N(u) = 1$ and the norm is multiplicative. But at the same time, we assume that

$$\|a'\|_2^2 = \sum_{i=1}^{k-1} |\sigma_i(a')|^2$$

is exceptionally small in the canonical embedding because a' is a *short* generator of $\mathcal{I}'$. Thus, the summands $|\sigma_i(a')|^2$ for $i = 1, \ldots, k-1$ need to be roughly balanced in terms of absolute size. But this holds true also for the logarithm of their square roots, and thus the sizes of the coefficients of $\mathrm{Log}(a')$ are approximately equal.

Recall that the log-unit lattice $\mathrm{Log}(\mathcal{O}_k^\times)$ lies in the hyperplane orthogonal to the vector $\mathbf{1} = (1, \ldots, 1) \in \mathbb{R}^{k-1}$. Because we expect the logarithmic embedding $\mathrm{Log}(a')$ to have approximately equal coefficients, the projection $\pi\left(\mathrm{Log}(a')\right)$ is expected to be short. I.e., we expect $\pi\left(\mathrm{Log}(c)\right)$ to lie exceptionally close to the log-unit lattice, and the recovery of $\mathrm{Log}(u)$ from $\pi\left(\mathrm{Log}(c)\right)$ is a unique decoding problem in the log-unit lattice.

For efficient decoding in the log-unit lattice, we make use of a good basis $\boldsymbol{B}$ of $\mathrm{Log}(\mathcal{O}_k)$ and its pseudo-inverse $\boldsymbol{B}^\dagger$. In particular, we choose $\boldsymbol{B}$ as in [12], i.e., consisting of the units $(x^j - 1)/(x - 1)$ for $j = 2, \ldots, (k-1)/2$.[8] We can write $\pi\left(\mathrm{Log}(c)\right) = \boldsymbol{y}\boldsymbol{B}$ for some non-integer vector $\boldsymbol{y} \in \mathbb{R}^{(k-3)/2}$, and thus $\boldsymbol{y} = \pi\left(\mathrm{Log}(c)\right) \cdot \boldsymbol{B}^\dagger$ in terms of the pseudo-inverse $\boldsymbol{B}^\dagger$. Now because $\boldsymbol{B}$ is a good basis and $\pi\left(\mathrm{Log}(c)\right)$ lies exceptionally close to the lattice, we can round the coefficients of $\boldsymbol{y}$ to obtain the nearby lattice point

$$\mathrm{Log}(u) = \lfloor \boldsymbol{y} \rceil \boldsymbol{B} = \left\lfloor \pi\left(\mathrm{Log}(c)\right) \cdot \boldsymbol{B}^\dagger \right\rceil \boldsymbol{B}.$$

The precise condition for the above to be true is that

$$\pi\left(\mathrm{Log}(c)\right) - \mathrm{Log}(u) = \pi\left(\mathrm{Log}(a')\right) \in \left(-\tfrac{1}{2}, \tfrac{1}{2}\right)^{(k-3)/2} \boldsymbol{B},$$

or, equivalently, that all coefficients of $\pi\left(\mathrm{Log}(a')\right) \cdot \boldsymbol{B}^\dagger$ lie in the interval $\left(-\frac{1}{2}, \frac{1}{2}\right)$, which is the case if $\pi\left(\mathrm{Log}(a')\right)$ is small enough.

[8] As discussed in [12], $\boldsymbol{B}$ does *not* necessarily generate the full log-unit lattice but only a sublattice thereof. To still decode efficiently in the full lattice, we only need the index of the sublattice to be small, and that seems to be the case under reasonable heuristics. In particular, the index is less than 11 for all primes $k \leq 241$. If desired, a full basis for the log-unit lattice can efficiently be computed by a quantum algorithm.

The decoding process could be further improved by considering Babai's nearest-plane decoding instead of simple coefficient rounding. However, this is not needed in our case.

Lifting the result and recovering the key. From $\mathrm{Log}(u)$ we obtain the unit u up to some root of unity x^{-i} and therefore a short generator $c/(x^{-i}u) = x^i a'$ of $\mathcal{I}'$. Recall that the quotient ring $R = \mathbb{Z}[x]/(x^k - 1)$ splits into a product of number rings $R_1 = \mathbb{Z}[x]/(\psi_k)$ and $R_2 = \mathbb{Z}[x]/(x-1)$. So to lift $a' = x^i a \pmod{\psi_k} \in R_1$ to $x^i a \in R$ we furthermore need to know the value of $x^i a \pmod{x-1} = a \pmod{x-1} \in R_2$. Note that $a \pmod{x-1} = a(1)$. One can either guess this small value, or use that for any set $c_1, \dots, c_s$ of generators for $\mathcal{I}$ we have $a(1) = \gcd(c_1(1), \dots, c_s(1))$.

To conclude, we obtain $x^i a$, and thus a shift $\mathrm{shift}^i(\boldsymbol{a})$ of the secret vector $\boldsymbol{a}$. Using the public key, or alternatively a basis of $\mathcal{L}_2$, we then receive a shift of $\boldsymbol{b}$, and thus have achieved a full key recovery.

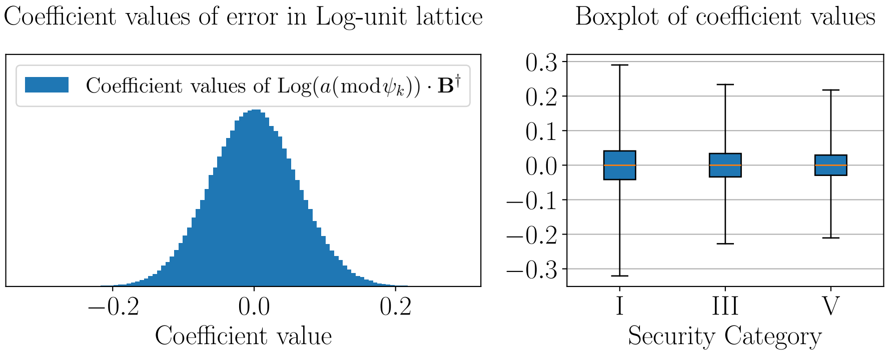


Fig. 5: On the left, distribution of the coefficients of $\mathrm{Log}(a \pmod{\psi_k}) \cdot \boldsymbol{B}^\dagger$ for FuLeeca-I obtained from 1,000 samples. On the right, a boxplot of the same values for all parameter sets showing the standard deviation as well as minimum and maximum values.

Experimental confirmation. We have yet to verify that $\pi(\mathrm{Log}(a'))$ is indeed small enough such that $\mathrm{Log}(u)$ is efficiently recovered by the rounding procedure. To show that this is the case, we generated 1,000 secret-key instances for each security level of FuLeeca. For each instance, we confirmed that the coefficients of $\pi(\mathrm{Log}(a')) \cdot \boldsymbol{B}^\dagger$ are indeed all well within the range of $\left(-\frac{1}{2}, \frac{1}{2}\right)$. In fact, the $\frac{k-3}{2} \cdot 1{,}000$ coefficients of the 1,000 signature samples for the parameter sets FuLeeca-I, FuLeeca-III, and FuLeeca-V are centered around zero with a standard deviation of $0.0617, 0.0503$, and 0.0435, and with a maximal absolute value of at most $0.3206, 0.2335$, and 0.2176, respectively. Figure 5 shows a full histogram of the distribution for FuLeeca-I and boxplots for all security levels.

Remark 1. For certain nice distributions of the unexpectedly short generator a', one can *prove* analytically that the decoding in the log-unit lattice will succeed with high probability [12]. For FuLeeca however, the distribution seems too adhoc to easily prove a similar statement. From the perspective of an attacker, the numerical validation of this step is sufficient.

6 A full learning attack

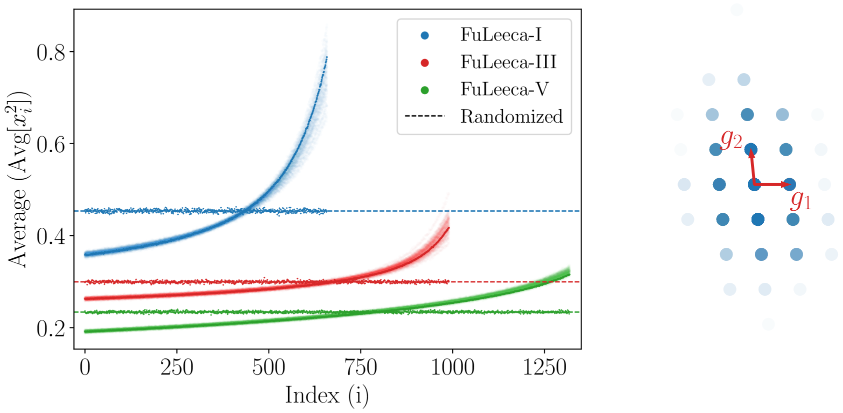
In this section, we present a learning attack on FuLeeca that recovers the full secret key for all parameter sets FuLeeca-I, FuLeeca-III, and FuLeeca-V with only 90,000, 175,000, and 175,000 signatures, respectively. The attack exploits a bias introduced by the concentration procedure within the signing algorithm. Furthermore, we show a simple way to remove the bias and thus prevent this attack.

6.1 Observing and precomputing the bias

Recall that the signing procedure consists of a simple-signing step and a concentration step, which are depicted in Algorithm 2. The output is a signature $\boldsymbol{v}$ that is the product $\boldsymbol{v} = \boldsymbol{x}\boldsymbol{G}_{\text{sec}}$ of a suitable vector $\boldsymbol{x} \in \mathbb{F}_p^k$ and the secret key $\boldsymbol{G}_{\text{sec}} \in \mathbb{F}_p^{k \times n}$. Essentially, the concentration step tries to concentrate the Lee weight of $\boldsymbol{v}$ and the logarithmic matching probability $\text{LMP}(\boldsymbol{c}, \boldsymbol{v})$ around prescribed values, where $\boldsymbol{c} \in \{\pm 1\}^n$ is the message hash. Recall that the value of $\text{LMP}(\boldsymbol{c}, \boldsymbol{v})$ is directly determined by $\text{mt}(\boldsymbol{c}, \boldsymbol{v})$. This number of sign matches $\text{mt}(\boldsymbol{c}, \boldsymbol{v})$ of $\boldsymbol{c}$ and $\boldsymbol{v}$ is concentrated by successively adding or subtracting the rows $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k$ of the secret key $\boldsymbol{G}_{\text{sec}}$. The procedure is similar to the iterative slicing algorithm for finding close lattice vectors [36].

The algorithm proceeds by trying to improve the number of sign matches first by adding or subtracting $\boldsymbol{g}_1$, then $\boldsymbol{g}_2$, all the way up to $\boldsymbol{g}_k$. However, the order in which this is done introduces a significant bias. More precisely, the first vector $\boldsymbol{g}_1$ is considered first every time and thus has a higher probability to be added or subtracted in contrast to, say, the last vector $\boldsymbol{g}_k$. This is visualized in Figure 6b for two dimensions, where the distribution of the signatures rather creates an elliptic shape than a circle and thus clearly deviates from the expected outcome in a non-biased case.

Recall that adding $\pm \boldsymbol{g}_i$ to a signature $\boldsymbol{v} = \boldsymbol{x}\boldsymbol{G}_{\text{sec}}$ corresponds to increasing or decreasing the i-th coordinate x_i of $\boldsymbol{x}$ for $i = 1, \ldots, k$. Indeed, we observe for the signatures generated by the reference implementation of FuLeeca that the first coefficients of $\boldsymbol{x}$ are smaller than the latter ones. More concretely, we observe that the average values $\text{Avg}\left[x_1^2\right], \ldots, \text{Avg}\left[x_k^2\right]$ are increasing, as shown in Figure 6a. We use an approximation of these biased average values for the learning attack. Therefore, we precomputed a single key and 2,500,000 signatures for each parameter setting, and saved the average values as $\widetilde{\boldsymbol{d}} = (\widetilde{d}_1, \dots, \widetilde{d}_k) := \left(\mathrm{Avg}\left[x_1^2\right], \dots, \mathrm{Avg}\left[x_k^2\right]\right)$. Figure 6a shows that not all keys have exactly the same bias, but their individual values turn out to be sufficiently close to the precomputed data to permit our attack.



(a) Experimental observations of $\mathrm{Avg}\left[x_i^2\right]$.

(b) Visualization of signature bias in dimension two.

Fig. 6: Observed bias for FuLeeca signatures $\boldsymbol{v} = \boldsymbol{x}\boldsymbol{G}_{\mathrm{sec}}$. On the left, experimental observations of the average values $\mathrm{Avg}\left[x_i^2\right]$ for $i = 1, \dots, k$ for all parameter sets. The curves depict the results for 2,500,000 signatures from one key, i.e., the precomputed values $\widetilde{d}_1, \dots, \widetilde{d}_k$ used in the learning attack. The fog around the curves corresponds to 100,000 signatures generated for each of 50 random keys. The data points around the dashed lines show the result from one key and 100,000 signatures after properly randomizing the ordering in the concentration procedure. On the right, a two-dimensional visualization of the signature bias that clearly deviates from a circle.

To confirm that the fixed ordering of the concentration procedure is the cause of this bias, we adapted the reference implementation of FuLeeca to properly randomize the order in each iteration of the concentration procedure, and depicted the outcome in Figure 6a. Indeed, the previously biased average values $\mathrm{Avg}\left[x_1^2\right], \dots, \mathrm{Avg}\left[x_k^2\right]$ are roughly constant after this adaptation. This simple solution prevents the learning attack that follows. However, as long as there is no proof of non-leakage we do not expect this solution to hold against other learning attacks. In other schemes like Falcon [31] and Wave [5] this is achieved by following the GPV framework [17]. How to adapt these methods and proofs to the Lee metric is a non-trivial question, but one we think is necessary to answer in order to prevent other learning attacks on FuLeeca.

Remark 2. The learning attack on NTRUSign [27] uses the quasi-cyclic structure to turn a single signature sample into k distinct signatures by considering all quasi-circular shifts. Thereby, the number of needed signature vectors for the attack is severely reduced to a few hundred. This is not possible in our setting because the observed bias depends precisely on the fact that the signature distribution is *not* invariant under quasi-circular shifts.

6.2 Approximating the secret key from the bias

We now exploit the observed bias to recover an approximation of the secret vector $\boldsymbol{g} = (\boldsymbol{a} \mid \boldsymbol{b})$. In fact, we will first retrieve its first half $\pm\boldsymbol{a}$ up to sign and from this recover $\pm\boldsymbol{b}$ as well. Note that $-\boldsymbol{a}$ and $-\boldsymbol{b}$ work just as well for generating signatures and can be considered as an alternative secret key. The attack only uses the first half $\boldsymbol{w} \in \mathbb{Z}^k$ of each signature vector $\boldsymbol{v} = \boldsymbol{x}\boldsymbol{G}_{\text{sec}} \in \mathbb{Z}^n$ and the bias that we obtain therein. We call $\boldsymbol{w}$ a *partial signature* for simplicity and remark that it can be expressed as $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{A}$ because of the block structure of $\boldsymbol{G}_{\text{sec}} = (\boldsymbol{A} \mid \boldsymbol{B})$ with $\boldsymbol{A} = \text{Shift}(\boldsymbol{a})$ and $\boldsymbol{B} = \text{Shift}(\boldsymbol{b})$.

Remark 3. The attack can easily be generalized to use the full signatures, and as such directly recover $\boldsymbol{a}$ *and* $\boldsymbol{b}$. However, this does not seem to significantly improve the performance. In certain cases, it even decreases the success rate, while leading to higher computational costs because of the larger vectors.

Learning from expectations. Assume that N partial signatures $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N$ are available. We can compute the average outer product

$$\text{Avg}\big[\boldsymbol{w}^\top \boldsymbol{w}\big] := \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{w}_i^\top \boldsymbol{w}_i \approx \mathbb{E}\big[\boldsymbol{w}^\top \boldsymbol{w}\big]$$

to approximate the expectation of the outer product $\boldsymbol{w}^\top \boldsymbol{w} \in \mathbb{R}^{k \times k}$. Clearly, the quality of this estimate depends on the number N of given signature vectors. We will use this quantity to recover an approximation of the secret vector $\pm\boldsymbol{a}$.

Formally, we can rewrite the expected value of $\boldsymbol{w}^\top \boldsymbol{w}$ for $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{A}$ in terms of the expected value of $\boldsymbol{x}^\top \boldsymbol{x}$ as

$$\mathbb{E}\big[\boldsymbol{w}^\top \boldsymbol{w}\big] = \mathbb{E}\big[\boldsymbol{A}^\top \boldsymbol{x}^\top \boldsymbol{x}\boldsymbol{A}\big] = \boldsymbol{A}^\top \cdot \mathbb{E}\big[\boldsymbol{x}^\top \boldsymbol{x}\big] \cdot \boldsymbol{A} = \boldsymbol{A}^\top \boldsymbol{D}\boldsymbol{A} + \boldsymbol{A}^\top \boldsymbol{R}\boldsymbol{A},$$

where the last step decomposes the expectation $\mathbb{E}\big[\boldsymbol{x}^\top \boldsymbol{x}\big]$ into its diagonal part $\boldsymbol{D} = \text{diag}\,(d_1, \ldots, d_k)$ with $d_i = \mathbb{E}\big[x_i^2\big]$, and the remainder $\boldsymbol{R} = \mathbb{E}\big[\boldsymbol{x}^\top \boldsymbol{x}\big] - \boldsymbol{D}$ with zero diagonal. We write

$$\text{Avg}\big[\boldsymbol{w}^\top \boldsymbol{w}\big] = \boldsymbol{A}^\top \boldsymbol{D}\boldsymbol{A} + \boldsymbol{E} \qquad \text{with} \qquad \boldsymbol{D} = \text{diag}\,(d_1, \ldots, d_k)\,, \tag{4}$$

where $\boldsymbol{E}$ is an error matrix that includes $\boldsymbol{A}^\top \boldsymbol{R}\boldsymbol{A}$ and the error introduced by the approximation $\text{Avg}\big[\boldsymbol{w}^\top \boldsymbol{w}\big] \approx \mathbb{E}\big[\boldsymbol{w}^\top \boldsymbol{w}\big]$. We discuss in the following how $\pm\boldsymbol{a}$ can be recovered if no error $\boldsymbol{E}$ is present and then generalize the approach to the erroneous scenario.

Recovering $\pm\boldsymbol{a}$ when $\boldsymbol{E} = \boldsymbol{0}$. Let us for simplicity start with the error-free case, that is, with $\boldsymbol{E} = \boldsymbol{0}$ and thus $\operatorname{Avg}\left[\boldsymbol{w}^\top \boldsymbol{w}\right] = \boldsymbol{A}^\top \boldsymbol{D} \boldsymbol{A}$. The following lemma shows that we can fully recover $\pm\boldsymbol{a}$ as soon as we know $\boldsymbol{D}$ or, equivalently, its diagonal entries $d_1, \dots, d_k$.

Lemma 1. *Let $\boldsymbol{d} = (d_1, \dots, d_k) \in \mathbb{R}^k$ be a vector for which the matrix $\operatorname{Shift}(\boldsymbol{d})$ is invertible. Write $\boldsymbol{D} = \operatorname{diag}(\boldsymbol{d})$ and let $\boldsymbol{A} = \operatorname{Shift}(\boldsymbol{a})$ be circulant for a vector $\boldsymbol{a} \in \mathbb{R}^k$. Then, $\pm\boldsymbol{a}$ can be efficiently recovered from $\boldsymbol{d}$ and $\boldsymbol{A}^\top \boldsymbol{D} \boldsymbol{A}$.*

Proof. Note that the i-th row of the matrix $\boldsymbol{D}\boldsymbol{A}$ is given by $d_i \cdot \operatorname{shift}^{i-1}(\boldsymbol{a})$ for $i = 1, \dots, k$. Therefore, we can express $\boldsymbol{A}^\top \boldsymbol{D} \boldsymbol{A}$ as

$$\boldsymbol{A}^\top \boldsymbol{D} \boldsymbol{A} = \sum_{i=1}^{k} d_i \cdot \left(\operatorname{shift}^{i-1}(\boldsymbol{a})\right)^\top \operatorname{shift}^{i-1}(\boldsymbol{a}),$$

i.e., as a weighted sum of outer products. Now fix some $j = 1, \dots, k$ and consider the j-th diagonal of the above matrix equation. Since shifting a vector and taking the j-th diagonal of its outer product commute, we obtain

$$\begin{aligned}
\operatorname{diag}_j\left(\boldsymbol{A}^\top \boldsymbol{D} \boldsymbol{A}\right) &= \sum_{i=1}^{k} d_i \cdot \operatorname{diag}_j\left(\left(\operatorname{shift}^{i-1}(\boldsymbol{a})\right)^\top \operatorname{shift}^{i-1}(\boldsymbol{a})\right) \\
&= \sum_{i=1}^{k} d_i \cdot \operatorname{shift}^{i-1}\left(\operatorname{diag}_j\left(\boldsymbol{a}^\top \boldsymbol{a}\right)\right) \\
&= \boldsymbol{d} \cdot \operatorname{Shift}\left(\operatorname{diag}_j(\boldsymbol{a}^\top \boldsymbol{a})\right).
\end{aligned}$$

Recall that $\boldsymbol{x} \cdot \operatorname{Shift}(\boldsymbol{y}) = \boldsymbol{y} \cdot \operatorname{Shift}(\boldsymbol{x})$ holds for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^k$ to get

$$\operatorname{diag}_j\left(\boldsymbol{A}^\top \boldsymbol{D} \boldsymbol{A}\right) = \boldsymbol{d} \cdot \operatorname{Shift}\left(\operatorname{diag}_j\left(\boldsymbol{a}^\top \boldsymbol{a}\right)\right) = \operatorname{diag}_j\left(\boldsymbol{a}^\top \boldsymbol{a}\right) \cdot \operatorname{Shift}(\boldsymbol{d}).$$

Because $\operatorname{Shift}(\boldsymbol{d})$ is invertible, we can solve the obtained linear system to recover $\operatorname{diag}_j(\boldsymbol{a}^\top \boldsymbol{a})$. After repeating the above for all diagonals $j = 1, \dots, k$, we thus recover the outer product $\boldsymbol{a}^\top \boldsymbol{a}$, from which $\pm\boldsymbol{a}$ is easily computed. □

Note that we can apply Lemma 1 to recover $\pm\boldsymbol{a}$ in the error-free FuLeeca setting, because the observed bias makes the vector $\boldsymbol{d}$ strictly increasing and therefore the matrix $\operatorname{Shift}(\boldsymbol{d})$ invertible. For a constant vector $\boldsymbol{d}$, the matrix $\operatorname{Shift}(\boldsymbol{d})$ would only have rank 1 and the above recovery strategy would fail.

Approximating $\pm\boldsymbol{a}$ when $\boldsymbol{E} \neq \boldsymbol{0}$. Let us now consider the more general case in which $\boldsymbol{d}$ is unknown and $\boldsymbol{E} \neq \boldsymbol{0}$ applies. We will use our precomputed approximation $\widetilde{\boldsymbol{d}}$ of $\boldsymbol{d}$ from subsection 6.1 to obtain an approximation $\widetilde{\boldsymbol{a}}$ of $\pm\boldsymbol{a}$ using the same steps as in Lemma 1.

In the proof of Lemma 1, the entries of $\boldsymbol{a}^\top \boldsymbol{a}$ get computed from a linear system with coefficient matrix $\operatorname{Shift}(\boldsymbol{d})$. The same computation with the approximation $\operatorname{Shift}(\widetilde{\boldsymbol{d}})$ therefore leads to an approximation

$$\boldsymbol{Y} = \boldsymbol{a}^\top \boldsymbol{a} + \boldsymbol{E}_d$$

of $\boldsymbol{a}^\top \boldsymbol{a}$, where $\boldsymbol{E}_d$ is an error matrix that depends on the quality of the estimate $\widetilde{\boldsymbol{d}}$ of $\boldsymbol{d}$. Because our approximation $\widetilde{\boldsymbol{d}}$ of $\boldsymbol{d}$ is good as shown in Figure 6a, we expect $\boldsymbol{E}_d$ to be quite small.

In contrast to the setting of Lemma 1, we only know the erroneous version of $\boldsymbol{A}^\top \boldsymbol{D}\boldsymbol{A}$ displayed in equation (4). Thus, we now focus on the error matrix $\boldsymbol{E}$ that is the sum of $\boldsymbol{A}^\top \boldsymbol{R}\boldsymbol{A}$ and the error introduced by the approximation $\mathrm{Avg}\big[\boldsymbol{w}^\top \boldsymbol{w}\big] \approx \mathbb{E}\big[\boldsymbol{w}^\top \boldsymbol{w}\big]$. As we expect the approximation error to be small for a large enough number of signatures, $\boldsymbol{E}$ is eventually dominated by $\boldsymbol{A}^\top \boldsymbol{R}\boldsymbol{A}$. Recall that for simple signing we have $\boldsymbol{x} = \mathrm{trunc}(s/2 \cdot \boldsymbol{c} \cdot \mathrm{sgn}(\boldsymbol{G}_{\mathrm{sec}}^\top))$, and by the quasi-circulant structure of $\boldsymbol{G}_{\mathrm{sec}}$ the expectations $\mathbb{E}\big[\boldsymbol{x}^\top \boldsymbol{x}\big]$ and $\mathbb{E}\big[\boldsymbol{A}^\top \boldsymbol{x}^\top \boldsymbol{x}\boldsymbol{A}\big]$ are circulant, i.e. they have constant diagonals. After the concentration procedure, experiments show that all diagonals of $\boldsymbol{E}$ are still roughly constant, i.e., that $\boldsymbol{E} \approx \mathrm{Shift}(\boldsymbol{e})$ applies for some error vector $\boldsymbol{e} = (e_1, \ldots, e_k) \in \mathbb{R}^k$. For $j = 1, \ldots, k$, we thus get

$$\begin{aligned} \mathrm{diag}_j\left(\mathrm{Avg}\big[\boldsymbol{w}^\top \boldsymbol{w}\big]\right) &\approx \mathrm{diag}_j\left(\boldsymbol{A}^\top \boldsymbol{D}\boldsymbol{A} + \mathrm{Shift}(\boldsymbol{e})\right) \\ &= \mathrm{diag}_j\left(\boldsymbol{a}^\top \boldsymbol{a}\right) \cdot \mathrm{Shift}(\boldsymbol{d}) + (e_j, \ldots, e_j) \end{aligned}$$

and, since $(1, \ldots, 1)$ is an eigenvector of the circulant matrix $\mathrm{Shift}(\boldsymbol{d})$,

$$\mathrm{diag}_j(\mathrm{Avg}\big[\boldsymbol{w}^\top \boldsymbol{w}\big]) \cdot \mathrm{Shift}(\boldsymbol{d})^{-1} \approx \mathrm{diag}_j(\boldsymbol{a}^\top \boldsymbol{a}) + \tfrac{1}{\lambda} \cdot (e_j, \ldots, e_j) \tag{5}$$

with $\lambda \in \mathbb{R}$ being the corresponding eigenvalue. Because the entries $d_i = \mathbb{E}\big[x_i^2\big]$ of $\boldsymbol{d}$ are positive and thus also $\mathrm{Shift}(\boldsymbol{d})$ only contains positive entries, the eigenvalue $\lambda = \sum_{i=1}^k d_i$ is very large compared to the other eigenvalues. Experimentally, λ is between 9 and 18 times larger than the second biggest eigenvalue, and between 200 and 600 times larger than the average eigenvalue.

As a result, the error $e'_j := e_j/\lambda$ is reduced a lot and becomes relatively small, while in contrast $\mathrm{diag}_j(\boldsymbol{a}^\top \boldsymbol{a})$ does not get scaled down as much because it is expected to have a somewhat random direction. Overall, we thus obtain an approximation $\boldsymbol{Y} \approx \boldsymbol{a}^\top \boldsymbol{a} + \mathrm{Shift}(\boldsymbol{e}')$ of $\boldsymbol{a}^\top \boldsymbol{a}$ from equation (5), where $\boldsymbol{e}' := (e'_1, \ldots, e'_k)$ is relatively small.

To conclude, we note that $\boldsymbol{Y}$ is approximately a sum of a rank-1 matrix $\boldsymbol{a}^\top \boldsymbol{a}$ and a circulant error matrix $\mathrm{Shift}(\boldsymbol{e}')$ with relatively small entries. We can thus hope to recover an approximation $\widetilde{\boldsymbol{a}}$ of $\pm\boldsymbol{a}$ by computing an optimal rank-1 approximation of $\boldsymbol{Y}$. The latter is given by $\widetilde{\boldsymbol{a}} = \sqrt{\sigma_1}\boldsymbol{u}_1$, where σ_1 and $\boldsymbol{u}_1$ are obtained from the singular-value decomposition

$$\boldsymbol{Y} = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i^\top \boldsymbol{u}_i$$

with $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k \in \mathbb{R}^k$ and the singular values $\sigma_1, \ldots, \sigma_k \in \mathbb{R}$. Note that $\sigma_1 \geq \ldots \geq \sigma_k$ holds and that the decomposition can be computed efficiently. For a low number of signatures, the error contribution can still be large and thus one might want to consider $\widetilde{\boldsymbol{a}} = \sqrt{\sigma_i}\boldsymbol{u}_i$ for some small i. In the full attack, we simply

Algorithm 3: Approximate secret-key recovery.

```
Input  : Partial FuLeeca signatures $\boldsymbol{w}_1, \dots, \boldsymbol{w}_N$,
         approximation $\widetilde{\boldsymbol{d}}$ of $\boldsymbol{d} = \left(\mathbb{E}\left[x_1^2\right], \dots, \mathbb{E}\left[x_k^2\right]\right)$.
Output : Approximation $\widetilde{\boldsymbol{a}}$ of $\pm\boldsymbol{a}$.

$\boldsymbol{W} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{w}_i^\top \boldsymbol{w}_i$
$\boldsymbol{Y} = \boldsymbol{0} \in \mathbb{R}^{k \times k}$
for $j = 1, \dots, k$ do
    $\mathrm{diag}_j(\boldsymbol{Y}) = \mathrm{diag}_j(\boldsymbol{W}) \cdot \mathrm{Shift}(\widetilde{\boldsymbol{d}})^{-1}$
end
return $\mathrm{argmin}_{\widetilde{\boldsymbol{a}} \in \mathbb{R}^k} \left(\boldsymbol{Y} - \widetilde{\boldsymbol{a}}^\top \widetilde{\boldsymbol{a}}\right)$        // best rank-1 approximation
```

start with $i = 1$ and continue with larger values until the secret key is recovered or we abort. Algorithm 3 summarizes the resulting procedure.

Figure 7 shows that the distance between $\widetilde{\boldsymbol{a}}$ and $\pm\boldsymbol{a}$ is indeed small and decreases when the number of signatures increases, i.e., when the approximation $\mathrm{Avg}\left[\boldsymbol{w}^\top \boldsymbol{w}\right] \approx \mathbb{E}\left[\boldsymbol{w}^\top \boldsymbol{w}\right]$ improves. For a large number of signatures, the distance stays roughly the same, reflecting the fixed errors $\boldsymbol{E}_d$ and $\mathrm{Shift}(\boldsymbol{e}')$ which are independent of the number of signatures.

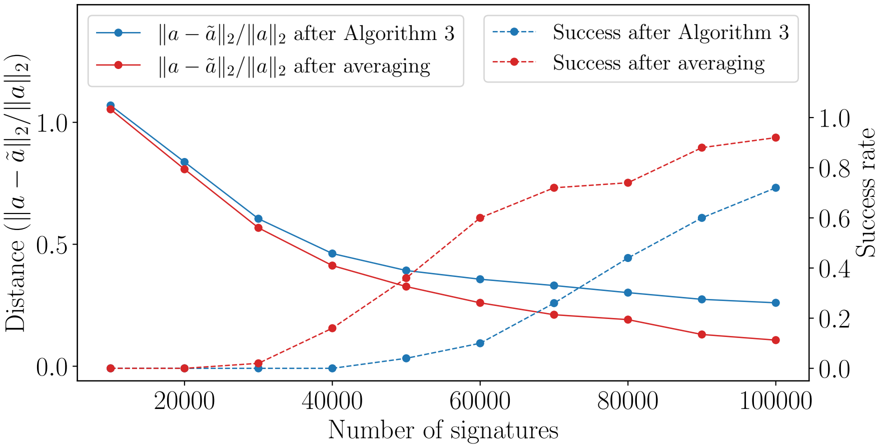


Fig. 7: Relative distance of approximate solution $\widetilde{\boldsymbol{a}}$ to $\pm\boldsymbol{a}$ after Algorithm 3 and after one averaging iteration, respectively. Further, success rates for recovering $\pm\boldsymbol{a}$ by the exact solving procedure from subsection 6.3 with the approximations obtained after no or one averaging step, respectively.

6.3 Recovering the secret key from its approximation

Given an approximation $\widetilde{\boldsymbol{a}}$ of the secret vector $\pm\boldsymbol{a}$, we now try to recover the exact solution. As $\widetilde{\boldsymbol{a}}$ already determines whether we recover $\boldsymbol{a}$ or $-\boldsymbol{a}$ and both options are valid alternative secret keys, we stick to $\boldsymbol{a}$ for brevity. We start with the general idea and then discuss further improvements that generally increase the success rate and therefore require less signatures to succeed.

General idea. The general idea is to use the short partial signature vectors $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{A}$ and the approximation $\widetilde{\boldsymbol{A}} = \mathrm{Shift}(\widetilde{\boldsymbol{a}})$ of $\boldsymbol{A} = \mathrm{Shift}(\boldsymbol{a})$, that was obtained as described in subsection 6.2. Given $\boldsymbol{w}$ and $\widetilde{\boldsymbol{A}}$, we can compute an approximation $\widetilde{\boldsymbol{x}} = \widetilde{\boldsymbol{A}}^{-1} \cdot \boldsymbol{w}$ of $\boldsymbol{x}$. More concretely, if $\widetilde{\boldsymbol{A}}^{-1} = \boldsymbol{A}^{-1} + \boldsymbol{E}'$ holds for some small error matrix $\boldsymbol{E}'$, we obtain

$$\widetilde{\boldsymbol{x}} - \boldsymbol{x} = \left(\boldsymbol{A}^{-1}\boldsymbol{w} + \boldsymbol{E}'\boldsymbol{w}\right) - \boldsymbol{x} = (\boldsymbol{x} - \boldsymbol{x}) + \boldsymbol{E}'\boldsymbol{w} = \boldsymbol{E}'\boldsymbol{w}.$$

Because both $\boldsymbol{E}'$ and $\boldsymbol{w}$ are small, $\widetilde{\boldsymbol{x}}$ is close to $\boldsymbol{x}$. Additionally, we know that $\boldsymbol{x}$ is an integral vector. We can thus recover $\boldsymbol{x} = \lfloor \widetilde{\boldsymbol{x}} \rceil$ by rounding the individual coefficients if $\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|_\infty < \frac{1}{2}$ applies. Once the correct $\boldsymbol{x}$ is recovered, the equation $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{A}$ allows to derive the circulant matrix $\boldsymbol{A}$ and thus the secret vector $\boldsymbol{a}$. To increase the success rate, we can try the above for every available partial signature vector $\boldsymbol{w}$ until we correctly recover $\boldsymbol{a}$.

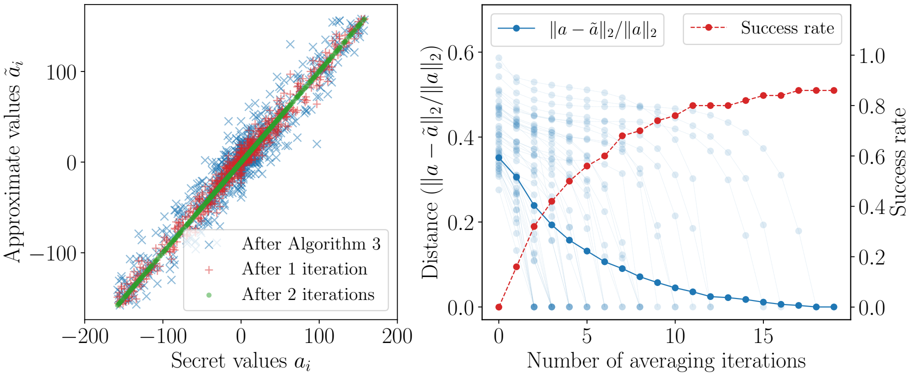
We can detect whether or not we have successfully recovered the secret key $\boldsymbol{a}$ as follows: If our guess of $\boldsymbol{x}$ is correct, then we obtain the correct integer vector $\boldsymbol{a}$ from the equation $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{A}$. If our guess of $\boldsymbol{x}$ was not correct, we would not expect the solution to be close-to-integral and have typical norm $\|\boldsymbol{a}\|_2$, and we use this to efficiently reject the failed trials. In practice, this approach perfectly distinguishes the correct from the incorrect cases.

For the cases that do pass the mentioned checks, we have several options to provably verify their correctness. One option is to compute a basis for the lattice $\mathcal{L}_3 = \mathcal{L}(\boldsymbol{A})$ generated by $\boldsymbol{A}$ from the available signatures as in section 5. We can then check if our recovered vector belongs to this lattice and is short enough in the Euclidean metric. Another option is to compute $\boldsymbol{b} = \boldsymbol{a}\boldsymbol{A}^{-1}\boldsymbol{B} \pmod p$ using the public key $\boldsymbol{G}_{\mathrm{pub}} = (\mathbf{I}_k \mid \boldsymbol{A}^{-1}\boldsymbol{B})$ and to verify that $\boldsymbol{a}$ and $\boldsymbol{b}$ are short enough in the Lee metric.

Making the vector typical. Due to the use of a typical vector in the key generation, we already know $\boldsymbol{a}$ up to a signed permutation. We can thus try to round our approximation $\widetilde{\boldsymbol{a}}$ to the closest typical vector. We do this by sorting the coefficients of $\widetilde{\boldsymbol{a}}$ and of the typical vector by absolute value, respectively, and matching the results up. For example, the largest absolute value x of $\widetilde{\boldsymbol{a}}$ gets transformed into $\mathrm{sgn}(x) \cdot y$, where y is the largest absolute value in the typical vector. The operation is efficient and we have observed that this generally improves the distance $\|\boldsymbol{a} - \widetilde{\boldsymbol{a}}\|_2$ compared to simply rounding $\widetilde{\boldsymbol{a}}$.

Iterative solving and averaging over guesses. Instead of directly recovering the exact secret key $\boldsymbol{a}$, we can try to improve the approximation iteratively until the approximation is good enough to recover the secret key exactly. One approach to achieve this is averaging over the approximated solutions we obtain for $\boldsymbol{a}$, in the hope of balancing out the errors. More precisely, we obtain an approximate solution $\lfloor \widetilde{\boldsymbol{x}} \rceil$ of $\boldsymbol{x}$ for any signature $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{A}$ from the general method presented above, and thus we get an approximation $\boldsymbol{a}'$ of $\boldsymbol{a}$ for every signature. Averaging over all these solutions $\boldsymbol{a}'$ yields a new approximate solution $\widetilde{\boldsymbol{a}}$ which is hopefully better than our initial approximation.

We can iteratively continue the process until the approximation is good enough to directly recover the secret key. The iterative improvements are showcased in Figure 8a. Because the same signature equations are used continuously, we observed that rounding or making $\widetilde{\boldsymbol{a}}$ typical after each iteration is important to not get stuck in a fixed point.



(a) Approximation $\widetilde{\boldsymbol{a}}$ vs secret key $\boldsymbol{a}$. (b) Distance $\|\boldsymbol{a} - \widetilde{\boldsymbol{a}}\|_2$ vs averaging iterations.

Fig. 8: Demonstration of first approximation guess and further improvements by iterative solving and averaging. The data has been collected using 40,000 signatures for each of 50 random FuLeeca-I keys.

Selecting near-integer entries. In the initial method, we try to recover $\boldsymbol{a}$ exactly from a single signature $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{A}$, hoping that all coefficients of $\widetilde{\boldsymbol{x}}$ round correctly to those of $\boldsymbol{x}$. Note however, that coefficients of $\widetilde{\boldsymbol{x}}$ that are already close to being integral have a higher chance of rounding correctly than those close to $\frac{1}{2} + \mathbb{Z}$. Additionally, each correctly rounded coefficient gives a linear equation on $\boldsymbol{a}$. We can thus consider multiple signatures and only select k coefficients that are exceptionally close-to-integral already, and then recover $\boldsymbol{a}$ from the resulting linear system. This method is similar to the threshold rounding in [23, 30]. For simplicity, we did not use this idea for the final attack script, as it only seemed to improve the required number of averaging iterations but not the overall success rate.

6.4 Running the learning attack

To demonstrate the attack, we generated 50 random keys and 100,000, 200,000, and 200,000 signatures for FuLeeca-I, FuLeeca-III, and FuLeeca-V parameters, respectively. We then ran the attack using the first N signatures for different values of N. The overall success rate of the attack versus the number of signatures is demonstrated in Figure 1. All keys get recovered with only 90,000, 175,000, and 175,000 signatures for the different security levels, respectively. More than 80% of the keys get recovered with only 40,000, 125,000, and 100,000 signatures, respectively. Note in particular that FuLeeca-V requires less signatures to break than FuLeeca-III. A possible explanation for this is that the values $\mathrm{Avg}\left[x_i^2\right]$ seem to have less variance between distinct keys for FuLeeca-V and are thus closer to the precomputed values, leading to a smaller error in the computation.

Figure 7 shows that the averaging strategy to improve the approximation $\widetilde{\boldsymbol{a}}$ of $\boldsymbol{a}$ is highly effective and increases the success rate of recovering the exact secret key significantly. In Figure 8b, we consider the regime with a relatively low number of signatures. The multiple averaging iterations combined with making the approximation typical can still recover the full secret even if the initial approximation was not so good.

The average of the best running time for each key, using the optimal number of signatures, is about 2, 12, and 14 minutes on a single core, respectively. The fastest successful attack took only 56 seconds, while the longest successful attack took a total of 98 minutes. All running times were a small fraction of the time needed to generate the signatures. To conclude, we can efficiently break all FuLeeca parameter sets given less than 175,000 signatures.

Running the attack scripts. The source code of our learning attack is available and open-source.[9] The main attack script is `attack/full_attack.py`. We provide pre-generated FuLeeca signatures for one fixed key per FuLeeca parameter set and a convenient shell script to run the three corresponding attack instances. First, execute `attack/1_download_sigs.sh` to download the signature samples. Then, run `attack/2_run_attack.sh` to start the learning attack and see how instances of all FuLeeca parameter sets are broken in real time.

Acknowledgments. Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine. W. van Woerden was supported by the CHARM ANR-NSF grant (ANR-21-CE94-0003).

[9] See `https://github.com/WvanWoerden/FuLeakage/` for the code and instructions.

References

1. Albrecht, M., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions: Cryptanalysis of some FHE and graded encoding schemes. Cryptology ePrint Archive, Report 2016/127 (2016), https://eprint.iacr.org/2016/127
2. Albrecht, M., Ducas, L.: Lattice attacks on ntru and lwe: a history of refinements. Cryptology ePrint Archive (2021)
3. Albrecht, M., Ducas, L.: Lattice attacks on NTRU and LWE: A history of refinements. Cryptology ePrint Archive, Report 2021/799 (2021), https://eprint.iacr.org/2021/799
4. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015). https://doi.org/10.1515/jmc-2015-0016
5. Banegas, G., Carrier, K., Chailloux, A., Couvreur, A., Debris-Alazard, T., Gaborit, P., Karpman, P., Loyer, J., Niederhagen, R., Sendrier, N., Smith, B., Tillich, J.P.: Wave. Tech. rep., National Institute of Standards and Technology (2023), https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/wave-spec-web.pdf
6. Bariffi, J., Bartz, H., Liva, G., Rosenthal, J.: On the properties of error patterns in the constant Lee weight channel. In: International Zurich Seminar on Information and Communication (IZS 2022). Proceedings. pp. 44–48. ETH Zurich (2022). https://doi.org/10.3929/ETHZ-B-000535277
7. Biasse, J.F., Song, F.: Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In: Krauthgamer, R. (ed.) 27th SODA. pp. 893–902. ACM-SIAM, Arlington, VA, USA (Jan 10–12, 2016). https://doi.org/10.1137/1.9781611974331.ch64
8. Blanco-Chacón, I.: On the RLWE/PLWE equivalence for cyclotomic number fields. Applicable Algebra in Engineering, Communication and Computing **33**(1), 53–71 (2022). https://doi.org/10.1007/S00200-020-00433-Z
9. Bos, J.W., Bronchain, O., Ducas, L., Fehr, S., Huang, Y., Pornin, T., Postlethwaite, E.W., Prest, T., Pulles, L.N., van Woerden, W.: HAWK. Tech. rep., National Institute of Standards and Technology (2023), https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/hawk-spec-web.pdf
10. Conway, J.H., Sloane, N.J.A.: Sphere packings, lattices and groups, vol. 290. Springer Science & Business Media (2013)
11. Courtois, N., Finiasz, M., Sendrier, N.: How to achieve a McEliece-based digital signature scheme. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 157–174. Springer, Heidelberg, Germany, Gold Coast, Australia (Dec 9–13, 2001). https://doi.org/10.1007/3-540-45682-1_10
12. Cramer, R., Ducas, L., Peikert, C., Regev, O.: Recovering short generators of principal ideals in cyclotomic rings. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 559–585. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49896-5_20
13. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 329–358. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56880-1_12

14. Ducas, L., Nguyen, P.Q.: Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 433–450. Springer, Heidelberg, Germany, Beijing, China (Dec 2–6, 2012). https://doi.org/10.1007/978-3-642-34961-4_27
15. Ducas, L., Postlethwaite, E.W., Pulles, L.N., van Woerden, W.P.J.: Hawk: Module LIP makes lattice signatures fast, compact and simple. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part IV. LNCS, vol. 13794, pp. 65–94. Springer, Heidelberg, Germany, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22972-5_3
16. Felderhoff, J., Pellet-Mary, A., Stehlé, D.: On module unique-SVP and NTRU. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part III. LNCS, vol. 13793, pp. 709–740. Springer, Heidelberg, Germany, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22969-5_24
17. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press, Victoria, BC, Canada (May 17–20, 2008). https://doi.org/10.1145/1374376.1374407
18. Gentry, C., Szydlo, M.: Cryptanalysis of the revised NTRU signature scheme. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 299–320. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Apr 28 – May 2, 2002). https://doi.org/10.1007/3-540-46035-7_20
19. Goldreich, O., Goldwasser, S., Halevi, S.: Public-key cryptosystems from lattice reduction problems. In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 112–131. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 1997). https://doi.org/10.1007/BFb0052231
20. Hoffstein, J., Howgrave-Graham, N., Pipher, J., Silverman, J.H., Whyte, W.: NTRUSign: Digital signatures using the NTRU lattice. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 122–140. Springer, Heidelberg, Germany, San Francisco, CA, USA (Apr 13–17, 2003). https://doi.org/10.1007/3-540-36563-X_9
21. Hoffstein, J., Howgrave-Graham, N., Pipher, J., Whyte, W.: Practical lattice-based cryptography: NTRUEncrypt and NTRUSign. pp. 349–390. ISC, Springer, Heidelberg, Germany (2010). https://doi.org/10.1007/978-3-642-02295-1
22. Hu, Y., Wang, B., He, W.: NTRUSign with a new perturbation. IEEE Transactions on Information Theory **54**(7), 3216–3221 (2008). https://doi.org/10.1109/TIT.2008.924662
23. Lin, X., Suzuki, M., Zhang, S., Espitau, T., Yu, Y., Tibouchi, M., Abe, M.: Cryptanalysis of the Peregrine lattice-based signature scheme. IACR Cryptology ePrint Archive p. 1628 (2023), https://eprint.iacr.org/2023/1628
24. MATZOV: Report on the security of LWE: Improved dual lattice attack (2022), https://doi.org/10.5281/zenodo.6412487
25. National Institute of Standards and Technology: NIST post-quantum cryptography standardization process (2016), https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions
26. National Institute of Standards and Technology: NIST post-quantum cryptography standardization process: Additional signatures (2023), https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures
27. Nguyen, P.Q., Regev, O.: Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 271–288. Springer, Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_17

28. Plantard, T., Sipasseuth, A., Dumondelle, C., Susilo, W.: DRS. Tech. rep., National Institute of Standards and Technology (2017), available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions
29. Postlethwaite, E.W., Virdia, F.: On the success probability of solving unique SVP via BKZ. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 68–98. Springer, Heidelberg, Germany, Virtual Event (May 10–13, 2021). https://doi.org/10.1007/978-3-030-75245-3_4
30. Prest, T.: A key-recovery attack against Mitaka in the *t*-probing model. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 205–220. Springer, Heidelberg, Germany, Atlanta, GA, USA (May 7–10, 2023). https://doi.org/10.1007/978-3-031-31368-4_8
31. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
32. Ritterhoff, S., Maringer, G., Bitzer, S., Weger, V., Karl, P., Schamberger, T., Schupp, J., Wachter-Zeh, A.: FuLeeca. Tech. rep., National Institute of Standards and Technology (2023), https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/FuLeeca-spec-web.pdf
33. Ritterhoff, S., Maringer, G., Bitzer, S., Weger, V., Karl, P., Schamberger, T., Schupp, J., Wachter-Zeh, A.: FuLeeca: A Lee-based signature scheme. In: Esser, A., Santini, P. (eds.) Code-Based Cryptography - 11th International Workshop, CBCrypto 2023, Lyon, France, April 22-23, 2023, Revised Selected Papers. Lecture Notes in Computer Science, vol. 14311, pp. 56–83. Springer (2023). https://doi.org/10.1007/978-3-031-46495-9_4
34. Schnorr, C., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming **66**, 181–199 (1994). https://doi.org/10.1007/BF01581144
35. Seo, E.Y., Kim, Y.S., Lee, J.W., No, J.S.: Peregrine: Toward fastest FALCON based on GPV framework. Cryptology ePrint Archive, Report 2022/1495 (2022), https://eprint.iacr.org/2022/1495
36. Sommer, N., Feder, M., Shalvi, O.: Finding the closest lattice point by iterative slicing. SIAM Journal on Discrete Mathematics **23**(2), 715–731 (2009). https://doi.org/10.1137/060676362
37. Yu, Y., Ducas, L.: Learning strikes again: The case of the DRS signature scheme. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 525–543. Springer, Heidelberg, Germany, Brisbane, Queensland, Australia (Dec 2–6, 2018). https://doi.org/10.1007/978-3-030-03329-3_18