FuLeeca

Submission to the NIST Post-Quantum Cryptography Standardization Process

Algorithm Specifications and Supporting Documentation

Stefan Ritterhoff, Sebastian Bitzer, Patrick Karl,
Georg Maringer, Thomas Schamberger, Jonas Schupp,
Georg Sigl, Antonia Wachter-Zeh, Violetta Weger

Technical University of Munich, Germany

May 31, 2023

Submitters: The team listed above is the principal submitter. There are no auxiliary submitters.

Inventors/Developers: Same as the principal submitter.

Implementation Owners: The submitters.

Email Address (preferred): fuleeca.cod@xcit.tum.de

Postal Address and Telephone:
Stefan Ritterhoff, Georg Maringer
Technical University of Munich
Institute for Communications Engineering
Theresienstraße 90
80333 Munich
Germany
Tel: +498928929051

Backup contact telephone and address:
Jonas Schupp
Technical University of Munich
Chair of Security in Information Technology
Theresienstraße 90
80333 Munich
Germany
Tel: +498928928190

Signature: ×
(See “Statement by Each Submitter” or “Cover Sheet”)

Contents

1 Introduction

We propose a code-based signature scheme in the Lee metric.

The main features of the submission are:

Alternative Metric The already standardized signature schemes are either based on structured lattices or on hash functions. While classical code-based cryptography considers vector spaces endowed with the Hamming metric, other metrics have attracted attention in the context of cryptography, e.g., the rank metric. This work marks the first Lee-metric-based cryptographic primitive.

Small Signatures FuLeeca achieves small communication costs, i.e., small signature size plus public key size. This is an important quantity for certificate chains. When comparing with the to be standardized schemes, the combined size of signature and public key of FuLeeca is slightly larger than the one of Falcon [32] but smaller than the ones of Dilithium [25] and SPHINCS$^+$[5]. For NIST security level I, we achieve a public key size of 1318 bytes and a signature size of 1100 bytes. The public key size is basically the same is in Dilithium (for level II), but larger than the one for the hash-based scheme SPHINCS$^+$. However, SPHINCS$^+$'s signature size is significantly larger than the one of FuLeeca. In fact, the signature size of FuLeeca is only 14% of the signature size of SPHINCS$^+$ and about 50% smaller than for Dilithium.

1.1 Historical Background

In general, there are two main methods to construct code-based signature schemes: the first one applies the Fiat-Shamir transform to a code-based zero-knowledge identification scheme and the second one is called Hash-and-Sign approach. The former approach usually suffers from huge signature sizes, due to large cheating probabilities within the identification scheme, and the latter one features small signature sizes at the cost of larger public key sizes.

The signature scheme we propose is based on the Hash-and-Sign approach. The first code-based scheme following this approach was introduced in 2001 by Courtois, Finiasz and Sendrier [21] (following the idea of [12]) and is often called the CFS scheme. This classical Hash-and-Sign signature scheme is a direct adaption of the McEliece public-key encryption scheme. In fact, the rationale is to start with an algebraically structured secret code that comes with an efficient decoding algorithm. The public key is a disguised version of the secret code. One then hashes the message (appended by a counter), interprets the digest as a syndrome and repeats this as necessary with an incremented nonce until the secret decoding algorithm produces an error-vector of sufficiently low Hamming weight. This approach has some potential drawbacks that have been exploited for attacks in the past: on one hand, the public code might be distinguishable from a random code and thus leak information about the secret code[1]. On the other hand, the event that the hash of a message is a syndrome of a low-weight codeword is highly unlikely and therefore this process has to be repeated many times. This causes the signing time of CFS to be impractically high. Additionally, as the public key is a disguised version of an algebraically structured code, the public key size of CFS might be rather large.

The CFS scheme was the starting point for several Hash-and-Sign signature schemes, such as [6, 35, 20], which have not survived cryptanalysis [42, 45]. The code-based scheme WAVE [23] follows the same blueprint but is based on a new problem: decoding large weight errors. This scheme managed to prevent all aforementioned attacks and so far no successful cryptanalysis has been mounted. However, this comes

[1] See for example [28], where the CFS scheme using high rate Goppa codes has been attacked.

at the cost of impractically large public key sizes.

Code-based signature schemes based on quasi-cyclic structures with low Hamming density codes, e.g., [6, 44], are vulnerable to statistical key attacks [48, 24]. All mentioned attacks have in common that they make use of the small support of the secret key. An attacker can recover the sparse secret key by observing the distribution of many signatures and comparing it to a random distribution. The use of the Lee metric thwarts such attacks, as even though the Lee weight of the secret basis is low, the number of non-zero entries can be very high for large enough field sizes.

FuLeeca is therefore based on quasi-cyclic Lee-metric codes. We set the Lee weight of the secret generators on the Lee-metric Gilbert-Varshamov (GV) bound. This allows us to treat the secret code like a random linear quasi-cyclic Lee-metric code, which attain this bound with high probability.

1.2 Overview of the Basic Idea

In a nutshell, the signature scheme works as follows: the secret key is a quasi-cyclic generator matrix, where the generators have Lee weight according to the Lee-metric GV bound and the public key is its systematic form. Note that recovering the original basis is as hard as the problem of finding codewords of given Lee weight, which is proven to be NP-hard [53]. The binary hash output of the message $\boldsymbol{m}$ is mapped onto $\{\pm 1\}$ and is considered as the target vector $\boldsymbol{c}$ for the main step of the scheme: the signer uses the secret basis to find a codeword, which will be the signature for $\boldsymbol{m}$, with two properties: firstly, the Lee weight should be close to a fixed target Lee weight and secondly, the signum of the codeword should have many 1, respectively -1 in the same places as the target vector $\boldsymbol{c}$, we will call this the matching of the signs. This second property is used to bind the message to the signature, while the first property is essential to make the scheme secure.

A multiple-use signature scheme should possess an existential unforgeability under adaptive chosen message attacks (EUF-CMA) security proof. For code-based signatures constructed from a zero-knowledge identification scheme this property is assured through the number of rounds and the cheating probability. However, the EUF-CMA security proof is notoriously difficult for Hash-and-Sign approaches. To the bests of our knowledge, WAVE [23] is the only known code-based Hash-and-Sign signature scheme that provides such a proof. Unfortunately, the achieved public-key size of more than 2 megabytes for 128 bit classical security is very large compared to Falcon's 897 bytes. Even though we do not provide a full security proof for FuLeeca, we consider attacks exploiting the leakage via published hash/signature-pairs and design our scheme integrating countermeasures for those attacks. Heuristically, we observe that our scheme does not leak any information via the standard attack vectors.

2 Preliminaries

2.1 Notation

Throughout this work, we denote by $\mathbb{F}_p$ the finite field of order p, where p is a prime. We often choose to represent this prime field as $\{-\frac{p-1}{2}, \ldots, 0, \ldots, \frac{p-1}{2}\}$, which we call the symmetric representation. We denote vectors in bold lowercase and matrices in bold uppercase letters. We refer to the i-th element of the vector $\boldsymbol{v}$ by v_i and similarly, to the j-th row of a matrix $\boldsymbol{A}$ by $\boldsymbol{a}_j$ and we denote the element in the j-th row and k-th column by $a_{j,k}$. The identity matrix of size n is denoted by $\boldsymbol{I}_n$. We denote by uppercase letters sets and for a set $S \subset \{1, \ldots, n\}$, we denote by $|S|$ the cardinality and by $S^C = \{1, \ldots, n\} \setminus S$ the complement. For a set $S \subset \{1, \ldots, n\}$ of size s and matrix $\boldsymbol{A} \in \mathbb{F}_p^{k \times n}$, we denote by $\boldsymbol{A}_S$ the $k \times s$ matrix formed by the columns of $\boldsymbol{A}$ indexed by S, similarly for a vector $\boldsymbol{x} \in \mathbb{F}_p^n$, we denote by $\boldsymbol{x}_S$ the vector of length s formed by the entries of $\boldsymbol{x}$ indexed by S.

The sampling of an element a from the uniform distribution over a set $\mathcal{K}$ is denoted by $a \overset{\$}{\leftarrow} \mathcal{K}$. While the sampling of an element a according to a distribution χ is given by $a \overset{\$}{\leftarrow} \chi$ and by a slight abuse of notation we denote sampling of a vector $\boldsymbol{v}$ independently and identically distributed (i.i.d.) from χ by $\boldsymbol{v} \overset{\$}{\leftarrow} \chi$.

The binary entropy function with parameter p is defined as $h_2(p) \coloneqq -p\log_2(p) - (1-p)\log_2(1-p)$.

2.2 Cryptographic Notation

We denote the security parameter by λ. We use standard definitions of probabilistic polynomial time algorithms. We denote by "Hash" a Hash function in the perfect random oracle model. For more details we refer to Section 7.

In a digital signature scheme we have two parties, *the signer* and the *verifier*, and three efficiently computable algorithms: the key generation, the signature generation and the signature verification. In the key generation, the signer randomly samples a secret key sk and computes and publishes the connected public key pk. For the signature generation, given a message $\boldsymbol{m}$, the signer then uses the secret key sk to compute a signature $\boldsymbol{v}$. The signer then sends $(\boldsymbol{m}, \boldsymbol{v})$ to the verifier. The verifier checks the validity of the signature $\boldsymbol{v}$ for the message $\boldsymbol{m}$ under the constraints imposed by the scheme using the public key in the signature verification step. An adversary might try to construct a valid signature, either using just the knowledge of the public key, or after having observed several signatures corresponding to different messages. The adversary is only allowed to succeed with negligible probability, e.g., $< 2^{-\lambda}$.

2.3 Coding Theory Notation

An $[n, k]$ *linear code* $\mathcal{C}$ is a k-dimensional linear subspace of $\mathbb{F}_p^n$ and can be compactly represented either through a *generator matrix* $\boldsymbol{G} \in \mathbb{F}_p^{k \times n}$, which has the code as its image or through a *parity-check matrix* $\boldsymbol{H} \in \mathbb{F}_p^{(n-k) \times n}$ having the code as its kernel. The elements of a code are called *codewords* and for any $\boldsymbol{x} \in \mathbb{F}_p^n$, we call $\boldsymbol{s} = \boldsymbol{x}\boldsymbol{H}^\top$ the *syndrome* of $\boldsymbol{x}$. The *rate* of an $[n, k]$ code is $R = \frac{k}{n}$.

For an $[n, k]$ linear code $\mathcal{C}$ and a set $I \subset \{1, \dots, n\}$, we denote by $\mathcal{C}_I$ the set of restrictions on codewords restricted to the coordinates specified in I. We say that $I \subset \{1, \dots, n\}$ of size k is an *information set*, if $|\mathcal{C}_I| = |\mathcal{C}|$. As a consequence, we have that for a generator matrix $\boldsymbol{G}$, respectively a parity-check matrix $\boldsymbol{H}$ of the code, $\boldsymbol{G}_I$ and $\boldsymbol{H}_{I^C}$ are invertible. We say that a generator matrix $\boldsymbol{G}$, respectively a parity-check matrix $\boldsymbol{H}$, is in systematic form (with respect to I), if $\boldsymbol{G}_I = \boldsymbol{I}_k$, respectively $\boldsymbol{H}_{I^c} = \boldsymbol{I}_{n-k}$.

Classically, we endow the vector space $\mathbb{F}_p^n$ with the Hamming metric, where the *Hamming weight* of a vector $\boldsymbol{v}$, denoted by $\mathrm{wt}_H(\boldsymbol{v})$, is given by the number of non-zero entries of $\boldsymbol{v}$. However, for this scheme, we are interested in a different metric, called the Lee metric.

The *Lee weight* of an element $a \in \mathbb{F}_p$ is defined as

$$\mathrm{wt}_L(a) := \min\{a, p-a\}, \tag{1}$$

where the representation of a is chosen to be in $\{0, \dots, p-1\}$. In fact, one can think of the Lee weight as the L_1-norm modulo p. Clearly, the Lee weight of an element can be at most $(p-1)/2$, thus we will denote this value by M. For a vector $\boldsymbol{v} \in \mathbb{F}_p^n$ its Lee weight is defined as the sum of the Lee weights of its elements, i.e.,

$$\mathrm{wt}_L(\boldsymbol{v}) := \sum_{i=1}^{n} \mathrm{wt}_L(v_i). \tag{2}$$

Note that, $\mathrm{wt}_H(\boldsymbol{v}) \leq \mathrm{wt}_L(\boldsymbol{v}) \leq M\mathrm{wt}_H(\boldsymbol{v})$ and the average Lee weight of the vectors in $\mathbb{F}_p^n$ is given by $(M/2)n$.

The Lee weight induces the *Lee distance*, which we define by $d_L(\boldsymbol{x}, \boldsymbol{y}) := \mathrm{wt}_L(\boldsymbol{x} - \boldsymbol{y})$, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_p^n$. For a linear code $\mathcal{C}$ we define the *minimum Lee distance* as

$$d_L(\mathcal{C}) = \min\{\mathrm{wt}_L(\boldsymbol{c}) \mid \boldsymbol{c} \in \mathcal{C}, \boldsymbol{c} \neq 0\}.$$

We denote by δ the relative minimum Lee distance, that is $\delta = \frac{d_L(\mathcal{C})}{nM}$. Let us denote by $V_L(p, n, r)$ the *Lee sphere* of radius t

$$V_L(p, n, t) := \{\boldsymbol{x} \in \mathbb{F}_p^n \mid \mathrm{wt}_L(\boldsymbol{x}) = t\},$$

and by

$$F_L(p, T) = \lim_{n \to \infty} \frac{1}{n} \log_p(|V_L(p, n, TnM)|)$$

its asymptotic size. The exact formulas for the size of $V_L(p, n, t)$ and $F_L(p, T)$ can be found in [53, 33].

Let us denote by $A(n, \delta)$ the maximal size of a code in $\mathbb{F}_p^n$ of minimum Lee distance δMn and by

$$R(\delta) = \limsup_{n \to \infty} \frac{1}{n} \log_p(A(n, \delta)).$$

The Gilbert-Varshamov (GV) bound in the Lee-metric [4] then states:

$$R(\delta) \geq 1 - F_L(p, \delta).$$

In [18] it was shown that random Lee-metric codes attain with high probability the Lee-metric GV bound, i.e., a random code has with high probability a relative minimum Lee distance δ such that $R(\delta) = 1 - F_L(p, \delta)$. For the considered quasi-cyclic code of rate 1/2, the corresponding minimum Lee distance δ of codes on the GV bound will only depend on p and is thus denoted by δ_p^{GV}.

If $\mathcal{C} \in \mathbb{F}_p^n$ is a random code of dimension k, we can also compute the expected number of codewords of a given Lee weight w as

$$|V_L(p, n, w)|p^{k-n}.$$

2.4 Functions

For our scheme we represent the elements of $\mathbb{F}_p$ as

$$\left\{-\frac{p-1}{2}, \ldots, 0, \ldots, \frac{p-1}{2}\right\}$$

for $p > 3$ prime and $n \in \mathbb{N}$ even. As usual, we write M for the maximal Lee weight in $\mathbb{F}_p$, that is $M = \frac{p-1}{2}$. We define a function $\mathrm{sgn}(x)$, that gives us the sign of an element in $\mathbb{F}_p$.

Definition 1 (Signum)**.** For $x \in \mathbb{F}_p = \left\{-\frac{p-1}{2}, \dots, 0, \dots, \frac{p-1}{2}\right\}$ let

$$\operatorname{sgn}(x) = \begin{cases} 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0, \\ -1 & \text{if } x < 0. \end{cases}$$

For the symmetric representation of $\mathbb{F}_p$ this corresponds to the common signum function.

Furthermore, we define a matching function $\operatorname{mt}(\boldsymbol{x}, \boldsymbol{y})$ that compares $\boldsymbol{x}$ and $\boldsymbol{y}$ and counts the number of symbols that hold the same sign.

Definition 2 (Sign Matches)**.** Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_p^n$ and consider the number of matches in their sign such that

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

We are interested in upper bounding the probability of an attacker being able to reuse any of the previously published signatures. For that, we introduce a function calculating the probability that a vector and a uniformly random hash digest (in $\{\pm 1\}^n$) have μ sign matches. When talking about the security of the signature scheme, we will usually consider the negative $\log_2$ of this probability.

Definition 3 (Logarithmic Matching Probability (LMP))**.** For a fixed $\boldsymbol{v} \in \mathbb{F}_p^n$ and $\boldsymbol{y} \overset{\$}{\leftarrow} \{\pm 1\}^n$, the probability of $\boldsymbol{y}$ to have $\mu := \operatorname{mt}(\boldsymbol{y}, \boldsymbol{v})$ sign matches with $\boldsymbol{v}$ is

$$B(\mu, \operatorname{wt}_H(\boldsymbol{v}), 1/2),$$

where $B(k, n, q)$ is the binomial distribution defined as

$$B(k, n, q) = \binom{n}{k} q^k (1-q)^{n-k} \quad .$$

To ease notation, we write $\operatorname{LMP}(\boldsymbol{v}, \boldsymbol{y}) = -\log_2(B(\mu, \operatorname{wt}_H(\boldsymbol{v}), 1/2))$.

Note that this function can be efficiently approximated via additions and subtractions of precomputed values of $\log_2(x!)$, i.e. using a look-up table.

In [9], the authors computed the marginal distribution of entries where vectors are uniformly distributed in $V_L(p, n, w)$. Let E denote a random variable corresponding to the realization of an entry of $\boldsymbol{x} \in \mathbb{F}_p^n$. As n tends to infinity we have the following result on the distribution of the elements in $\boldsymbol{x} \in \mathbb{F}_p^n$.

Lemma 4 ([9, Lemma 1])**.** For any $x \in \mathbb{F}_p$, the probability that one entry of $\boldsymbol{x}$ is equal to x is given by

$$p_w(x) = \frac{1}{Z(\beta)} \exp(-\beta \operatorname{wt}_L(x)),$$

where Z denotes the normalization constant and β is the unique solution to $w = \sum_{i=0}^{p-1} \operatorname{wt}_L(i) p_w(x)$.

Definition 5 (Typical Lee Set)**.** For a fixed weight w, let $p_w(x)$ be the probability from Lemma 4 of the element $x \in \mathbb{F}_p$. Then, we define the typical Lee set as

$$T(p, n, w) = \left\{\boldsymbol{x} \in \mathbb{F}_p^n \mid \boldsymbol{x}_i = x \text{ for } f(p_w(x)n) \text{ coordinates } i \in \{1, \dots, n\}\right\},$$

for a rounding function f. That is the set of vectors, for which the element x occurs $f(p_w(x)n)$ times.

In principle, f could be simply chosen as the rounding function. This would, however, mean that the elements of $T(p, n, w)$ do in general not have Lee weight w. This effect is particularly evident when moderate values w are picked, for which number occurrences would be rounded to zero for many field elements.

Therefore, to obtain a closer approximation of the target weight, we design f as follows: if the expected number of occurrences for a symbol $x \in \mathbb{F}_p$ according to $p_w(x)n$ is at least 1, we always round down. If, however, the element x is expected to occur at most once, we round up or down according to a threshold τ. This τ allows us fine control over the Lee weight of the vector $\boldsymbol{x} \in T(p, n, w) \subset \mathbb{F}_p^n$. We choose this value such that the vector used to generate the secret key has Lee weight as close to the GV bound as possible.

3 FuLeeca Signatures

In this section, we describe how FuLeeca works.

3.1 Key Generation

The key generation of our signature scheme is presented in Algorithm 1. The basic idea to generate the secret key $\boldsymbol{G}_{sec}$ is to sample two cyclic matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{F}_p^{n/2 \times n/2}$ of Lee weight $w_{key} = \delta_p^{\mathrm{GV}} n$, where $\boldsymbol{A}$ has to fulfill the extra property of being an invertible matrix. Note that this property is satisfied for random matrices with large probability. The public key is obtained by computing the row reduced Echelon form of $\boldsymbol{G}_{sec}$, referred to as $\boldsymbol{G}_{sys}$. The public key is then formed by the non-trivial part of $\boldsymbol{G}_{sys}$, which we denote by $\boldsymbol{T}$.

Algorithm 1: Key Generation

```
  Input: Prime $p$, code length $n$, security level $\lambda$, Lee weight $w_{\mathrm{key}}$
1 Choose $\boldsymbol{a}, \boldsymbol{b} \xleftarrow{\$} T(p, n/2, w_{key})$.
2 Construct cyclic matrix $\boldsymbol{A} \in \mathbb{F}_p^{n/2 \times n/2}$ from all shifts of $\boldsymbol{a}$. $\boldsymbol{A}$ needs to be invertible. If this is
    not the case, resample $\boldsymbol{a}$ according to Line 1.
3 Construct cyclic matrix $\boldsymbol{B} \in \mathbb{F}_p^{n/2 \times n/2}$ from all shifts of $\boldsymbol{b}$.
4 Generate the secret key $\boldsymbol{G}_{\mathrm{sec}} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \end{pmatrix} \in \mathbb{F}_p^{n/2 \times n}$.
5 Calculate the systematic form $\boldsymbol{G}_{\mathrm{sys}} = \begin{pmatrix} \boldsymbol{I}_{n/2} & \boldsymbol{T} \end{pmatrix}$ of $\boldsymbol{G}_{\mathrm{sec}}$ with $\boldsymbol{T} = \boldsymbol{A}^{-1}\boldsymbol{B}$.
  Output: public key $\boldsymbol{T}$, private key $\boldsymbol{G}_{\mathrm{sec}}$
```

Note that $|T(p, n/2, w_{key})|^2$ corresponds to the cardinality of our key space. In order to prevent brute force attacks this cardinality needs to be larger than 2^λ.

3.2 Signature Generation

Note that most of the Hash-and-Sign schemes require the Hash of a message to be a syndrome for a public parity-check matrix. In this Hash-and-Sign algorithm we proceed differently. We use the generator matrix to generate signatures which are codewords of Lee weight within a fixed range. The connection to the Hash of the message vector is established through the number of sign matches.

The signature generation takes as its input the message $\boldsymbol{m}$ to be signed and makes use of the private key $\boldsymbol{G}_{sec}$ and outputs the signature $\boldsymbol{y}$. To do so the algorithm utilizes the secret generators matrix of the code, namely the rows of $\boldsymbol{G}_{sec}$, to find a codeword $\boldsymbol{v} = [\boldsymbol{y}, \boldsymbol{yT}]$ of Lee weight in $[w_{sig} - \varepsilon_s, w_{sig}]$ with sign matches achieving a desired LMP between the hash of the message and the signature codeword. Without having access to a secret basis (the private key), it is already computationally hard to find codewords in the desired Lee weight range (even ignoring the LMP). Therefore, this property suffices to ensure that it is hard to generate fresh codewords that can function as signatures even for arbitrary hashes.

Loosely speaking, a high LMP value ensures that enough signs of the codeword $\boldsymbol{v}$ and challenge $\boldsymbol{c}$ match. This establishes the connection between the signature and the message and prevents reusing codewords contained in previously published signatures to sign freshly generated hashes. Sampling a fresh salt if a signing attempt does not work, guarantees that any message can be signed successfully.

Algorithm 2: Signing

```
Input: Secret key $\boldsymbol{a}, \boldsymbol{b}$, message $\boldsymbol{m}$, threshold $\varepsilon$, signature weight $w_{sig}$, key weight $w_{key}$, scaling
       factor $s \in \mathbb{R}$, security level $\lambda$, number of concentrating iterations $n_{\mathrm{con}}$.
Output: salt, signature $\boldsymbol{y}$.
1  $\boldsymbol{G}_{sec} \leftarrow (\boldsymbol{A}, \boldsymbol{B})$, $\boldsymbol{G} = \begin{pmatrix} \boldsymbol{G}_{sec} \\ -\boldsymbol{G}_{sec} \end{pmatrix}$ with rows $\boldsymbol{g}'_i$
2  $\boldsymbol{m}' \leftarrow \mathrm{Hash}(\boldsymbol{m})$
3  repeat
4      salt $\xleftarrow{\$} \{0,1\}^{256}$                                // Simple signing starts
5      $\boldsymbol{c} \leftarrow \mathrm{CSPRNG}(\boldsymbol{m}' \mid\mid \mathrm{salt})$
6      $c_i \leftarrow (-1)^{c_i} \quad \forall i$
7      $\boldsymbol{x} \leftarrow (0, \ldots, 0)$
8      for $i \leftarrow 1$ to $n/2$ do
9          $x_{mt} = \mathrm{mt}(\boldsymbol{g}_i, \boldsymbol{c}) - \frac{\mathrm{wt}_H(\boldsymbol{g}_i)}{2}$
10         $\boldsymbol{x}_i = \lfloor x_{mt} s \rfloor$                   // Simple signing ends
       end
11     $\mathcal{A} \leftarrow \{1, \ldots, n\}$                          // Allowed row index set
12     $\boldsymbol{\nu} \leftarrow \boldsymbol{x}\boldsymbol{G}_{sec}$                        // Concentrating starts
13     $\boldsymbol{\nu}' \leftarrow (0, ..., 0)$, $i' = 0$
14     $lf \leftarrow 1$
15     for $j \leftarrow 1$ to $n_{con}$ do
16         for $i \in \{1, \ldots, n\}$ do
17             $\boldsymbol{\nu}'' \leftarrow \boldsymbol{\nu} + \boldsymbol{g}'_i$
18             if $|\mathrm{LMP}(\boldsymbol{\nu}'', \boldsymbol{c}) - (\lambda + 64 + \varepsilon)| \leq |\mathrm{LMP}(\boldsymbol{\nu}', \boldsymbol{c}) - (\lambda + 64 + \varepsilon)|$ then
19                 if $i \in \mathcal{A} \mid\mid lf = 0$ then
20                     $\boldsymbol{\nu}' \leftarrow \boldsymbol{\nu}''$, $i' \leftarrow i$
           end
21         $w' \leftarrow \mathrm{wt}_L(\boldsymbol{\nu}')$
22         if $w' > w_{sig} - w_{key}$ then
               $lf \leftarrow 0$
23         if $w' \leq w_{sig}$ then
24             $\boldsymbol{\nu} \leftarrow \boldsymbol{\nu}'$
25         if $i' \leq \frac{n}{2}$ then
26             $\mathcal{A} \leftarrow \mathcal{A} \setminus \{i' + n/2\}$
27         else
28             $\mathcal{A} \leftarrow \mathcal{A} \setminus \{i' - n/2\}$
       end
29     if $\mathrm{wt}_L(\boldsymbol{\nu}) \leq w_{sig}$ && $\mathrm{wt}_L(\boldsymbol{\nu}) > w_{sig} - 2w_{key}$ && $\mathrm{LMP}(\boldsymbol{\nu}, \boldsymbol{c}) \geq \lambda + 64$ then
30         $[\boldsymbol{y}, \boldsymbol{yT}] \leftarrow \boldsymbol{\nu}$
31         return salt, ENCODE($\boldsymbol{y}$)
32     else
33         go to Line 3                                               // Concentrating ends
   end
```

In line 1, one takes the secret key $\boldsymbol{G}_{sec}$ from the Key Generation 1, and stacks it with its negative $-\boldsymbol{G}_{sec}$.

In line 2, we hash the input message and get $\boldsymbol{m}'$, which will be fed together with a salt to CSPRNG in line 5 to get the target vector $\boldsymbol{c}$ for the number of sign matches, i.e., the $\mathrm{LMP}(\boldsymbol{v}, \boldsymbol{c})$, where $\boldsymbol{v}$ denotes the information vector of the signature $\boldsymbol{y}$. Line 6 assures that $\boldsymbol{c}$ is in $\{\pm 1\}^n$ making its signs comparable with the signs of vectors in $\mathbb{F}_p^n$. In line 9, we are checking how many matches the row $\boldsymbol{g}_i$ has with the target vector $\boldsymbol{c}$. We take into account how many of the signs of $\boldsymbol{c}$ and $\boldsymbol{g}_i$ are matching in line 10, where $\lfloor \cdot \rfloor$ denotes truncation. We do this by setting the magnitude in the corresponding position of the information vector according to the number of matches and the scaling factor s. Thus, if the row has many matches with the target $\boldsymbol{c}$, we add this row multiple times. This results in the information vector $\boldsymbol{x}$ and in line 12 produces the preliminary codeword $\boldsymbol{v}$.

Lines 11-33, which we refer to as the *Concentrating* procedure, are necessary to ensure that the signatures vary as little as possible in Lee weight and sign matches.

$\mathcal{A}$ keeps track of which rows have already been added or subtracted from the codeword $\boldsymbol{v}$ and is updated respectively in line 26, 28. In line 14, we initiate the condition lf with 1, which keeps track whether the conditions of the signature (that is LMP and Lee weight) are satisfied, in which case lf will be set to 0. To ensure a constant time signature generation, the lines 16-28 will only run up to n_{con} times.

To have signatures with much lower Lee weight than other signatures is undesirable, as this might leak information on the secret key. Thus, the iterative approach in lines 16-20 is used to add or subtract the generator row minimizing the absolute difference to the desired LMP. For this we first add the row $\boldsymbol{g}_i'$ to $\boldsymbol{v}$ in line 17 and then check in line 18 if the difference of the LMP to the target is minimized by adding this row. Line 19 checks whether the row $\boldsymbol{g}_i'$ is within the set of allowed rows, i.e., in $\mathcal{A}$ or if the signature conditions are satisfied, i.e., $lf = 0$. This results in a codeword $\boldsymbol{v}'$ which is close enough to the target LMP.

Lines 21-24 aim at creating signatures of almost constant Lee weight. For this we compute in line 21 the Lee weight w' of $\boldsymbol{v}'$ and check in line 22 if it is close enough to the target Lee weight w_{sig}, i.e., at most has a w_{key} difference. In this case, we update the signature condition lf with 0. If the Lee weight w' is larger than the target, we reset $\boldsymbol{v}'$ with the initial $\boldsymbol{v}$ in lines 23, 24. The lines 25-28 update the set of rows which are allowed to be added. In fact, if $i' \leq n/2$, we added a row of $\boldsymbol{G}_{sec}$ and exclude the same row to be extracted again by excluding $i' + n/2$ from the allowed set $\mathcal{A}$. If $i' > n/2$, the added row was from $-\boldsymbol{G}_{sec}$ and we exclude $i' - n/2$ from $\mathcal{A}$ to avoid subtracting the same row again.

After all iterations have been completed, lines 29-33 check whether the resulting codeword is within the desired LMP/Lee weight range. If this is the case, we extract the information vector $\boldsymbol{y}$ from $\boldsymbol{v}$ in line 30 and publish the signature $(\text{salt}, \textsc{encode}(\boldsymbol{y}))$. The encoding procedure $\textsc{encode}(\cdot)$ is described in Section 7.6. Otherwise another salt is sampled and the signing procedure restarts.

The scaling parameter s used in line 10 is experimentally determined with the goal of minimizing the running time of the Signing algorithm. Its value is a trade-off between the probability of creating a valid signature for a specific hash value and the amount of iterations within the *Concentrating* procedure.

3.3 Signature Verification

The verification process is quite simple. In a first step, the received signature $\boldsymbol{y}'$ is decoded as explained in Section 7.6 to obtain the uncompressed vector $\boldsymbol{y}$. The verifier computes in line 3 and 4 $\boldsymbol{c}$ as CSPRNG from the hash of the message and salt. Then, the verifier checks that $\boldsymbol{v}$ is indeed a codeword of the public code; this is ensured by computing $\boldsymbol{v}$ as $[\boldsymbol{y}\ \boldsymbol{y}\boldsymbol{T}]$ in line 5.

Then, the verifier checks in line 6 that the codeword $\boldsymbol{v}$ has Lee weight of at most w_{sig}. Finally, one checks whether a sufficient amount of the signs of the signature $\boldsymbol{v}$ match the output $\boldsymbol{c}$ of the CSPRNG(Hash($\boldsymbol{m}$)||salt), i.e., LMP($\boldsymbol{v}, \boldsymbol{c}$) $\geq \lambda$+64. This verification process is given in Algorithm 3.

```
Algorithm 3: Verification
  Input: signature (salt, $\boldsymbol{y}'$) message $\mathbf{m}$, public key $\boldsymbol{T}$, Lee weight $w_{\mathrm{sig}}$.
1 $\boldsymbol{y} \leftarrow \textsc{decode}(\boldsymbol{y}')$
2 $\boldsymbol{m}' \leftarrow \mathrm{Hash}(\boldsymbol{m})$
3 $\boldsymbol{c} \leftarrow \mathrm{CSPRNG}(\boldsymbol{m}' \mid\mid \mathrm{salt})$
4 $c_i \leftarrow (-1)^{c_i} \quad \forall i$
5 $\boldsymbol{v} = [\boldsymbol{y}\ \boldsymbol{yT}]$.
6 Accept if the following two conditions are satisfied:
    (a) $\mathrm{wt}_L(\boldsymbol{v}) \leq w_{\mathrm{sig}}$,
    (b) $\mathrm{LMP}(\boldsymbol{v}, \boldsymbol{c}) \geq \lambda + 64$.
  Otherwise, Reject.
  Output: Accept or Reject
```

4 Design Rationale

This section explains the design rationale behind FuLeeca. We address each choice we made for the signature scheme.

Code-Based There are several principles used for quantum-secure cryptography, e.g., code-based, lattice-based, isogeny-based, hash-based and multivariate cryptography. The signature schemes which have so far been decided to be standardized, CRYSTALS-Dilithium[25], Falcon[32], and SPHINCS+[5], are hash-based and lattice-based schemes. It is desirable to have at one's disposal also signature schemes based on other hard problems, in case that a major breakthrough in solving lattice-based problems arises.

Quasi-Cyclic Codes The underlying Lee-metric code is chosen with a quasi-cyclic structure, as this allows for very compact public keys. This additional structure is common in code-based cryptography, see e.g., BIKE [2], and has been studied in [50].

Lee Metric The Lee metric has only recently been introduced to code-based cryptography and can be seen as an intermediate metric between the Hamming metric and the Euclidean metric used in lattice-based cryptography. The problem of finding a codeword of given Lee weight has been proven to be NP-complete [53]. The use of the Lee metric thwarts all the known statistical attacks, as the secret basis has a large Hamming weight. For more details see Section 6.2.

Secret Generators on GV By setting the Lee weight of the secret generators on the GV bound, we assume that the code behaves like a random code.

Sign Matches The signature has to have many sign matches with the target vector, which is the hash of the message. This is necessary to bind the message to the signature. The threshold on the logarithmic matching probability is chosen such that this condition is only met with negligible probability if the hash and the codeword are unrelated, i.e., either one is (pseudo)-random. The seemingly "ad hoc" introduced measure of dependence between codewords and hashes by comparing signs of elements is chosen for two reasons: first, conceptual simplicity since we need to be able to upper bound the probability of an attacker being able to reuse a previously published signature by finding a suitable hash. In the random oracle model we are therefore interested in the distribution of "dependence values" conditioned on a (uniformly) random hash. Note that for the chosen measure of dependence this is just the binomial distribution since the probability that any of the non-zero symbols in the codeword matches to a uniformly (pseudo-)random sign in the hash is independent of all other symbols and always $\frac{1}{2}$. To be precise, we aimed at finding a measure of dependence using only elementary arithmetic operations. The second reason is the fact that it is "orthogonal" in some sense to the absolute values / the Lee metric (at least for non-zero elements). This makes the analysis of the signature distribution in the "log probability" / Lee weight space useful since it illustrates how leakage of secret information may occur and how this may be counteracted. More specifically, this perspective leads to the addition of the *Concentrating* procedure.

Concentrating In order to thwart statistical attacks, we want all signatures to be concentrated around the target Lee weight and LMP values. Without this step, we might obtain signatures of much lower Lee weight than other signatures and thus might leak information on the secret key.

Hash-and-Sign Following the second request by NIST for compact signatures, we adopt the Hash-and-Sign strategy. Unlike most other schemes which effectively mask an existing algorithm with additional noise, we carefully refine the signature distribution iteratively to avoid leakage. Because this approach is not nearly as comprehensively studied as the other two, we try to keep the rest of the design as simple as possible.

5 Parameters

Due to the quasi-cyclic structure of the private matrix $\boldsymbol{G}_{sec}$ it is sufficient to store only one of its rows. Therefore, the size of the private key is in the order $\mathcal{O}_p(n)$, where the constant depends on the parameter p.

We take a conservative choice for the NIST security levels [38], as shown in Table 1.

Table 1: Conservative NIST Categories

NIST Security Level	Classical Cost	Quantum Cost
I	160	80
III	224	112
V	288	144

Table 2: Parameters for the proposed signature scheme FuLeeca. All sizes are given in Bytes.

p	n	ω_{sig}	ω_{key}	NIST cat.	secret key size	public key size	sign. size
65 521	1318	0.03	0.001 437	I	2636	1318	1100
65 521	1982	0.03	0.001 437	III	3964	1982	1620
65 521	2638	0.03	0.001 437	V	5276	2638	2130

The chosen parameters and associated data sizes for the NIST categories I, III and V are given in Table 2. We also give the relative Lee weights $\omega_{sig} = w_{sig}/(nM)$ and $\omega_{key} = w_{key}/(nM)$, where we recall that $M = \lfloor \frac{p-1}{2} \rfloor$ is the maximal Lee weight in $\mathbb{F}_p$.

The signature sizes are averaged over 1k generated compressed signatures and include the size of the salt. For compression, we have adapted the mechanisms as used in the Falcon signature scheme. Although the signature size is not constant, it can be padded to obtain a fix size. As proposed in [27], it is possible to compress the signatures resulting from Algorithm 2 even further.

Table 3: Comparison of post-quantum signature schemes for NIST level I (except for Dilithium which achieves NIST level II). All sizes are given in kB.

scheme	public key size	signature size	total size	variant
Falcon [32]	0.9	0.6	1.5	-
FuLeeca [This work]	1.3	1.1	2.4	-
Dilitihium [25]	1.3	2.4	3.7	-
R-BG [7]	0.1 0.1	7.7 7.2	7.8 7.3	Fast Short
Rank SDP Fen[29]	0.9 0.9	7.4 5.9	8.3 6.8	Fast Short
Ideal Rank BG[16]	0.5 0.5	8.4 6.1	8.9 6.6	Fast Short
PKP BG [16]	0.1 0.1	9.8 8.8	9.9 8.9	Fast Short
SDItH [31]	0.1 0.1	11.5 8.3	11.6 8.4	Fast Short
Ret. of SDitH [1]	0.1 0.1	12.1 5.7	12.1 5.8	Fast, V3 Shortest, V3
SPHINCS$^+$ [5]	<0.1 <0.1	16.7 7.7	16.7 7.7	Fast Short
Beu [15]	0.1 0.1	18.4 12.1	18.5 12.2	Fast Short
Durandal [3]	15.2	4.1	19.3	-
FJR [30]	0.1 0.1	22.6 16.0	22.7 16.1	Fast Short
GPS [36]	0.1 0.1	24.0 19.8	24.1 19.9	Fast Short
MinRank Fen [29]	18.2 18.2	9.3 7.1	27.5 25.3	Fast Short
LESS-FM [8]	10.4 205.7	11.6 5.3	23.0 211.0	Balanced Short sign
WAVE [23]	3200	2.1	3202	-

In Table 3, we show a comparison with SPHINCS$^+$ [5], Dilithium [25], Falcon [32] and many other post-quantum signature schemes for NIST level I (except for Dilithium which starts with level II). The total size (public key + signature size) of FuLeeca is 2.4kB, and shows that our scheme provides parameters that outperform the NIST selected schemes Dilithium [25] and SPHINCS$^+$ [5] in terms of total bandwidth and the state-of-the-art for code-based signature schemes.

In Table 4, we compare FuLeeca with the 3 standardized signature schemes, Dilithium, Falcon and SPHINCS$^+$ for all provided security levels. Dilithium [25] has similar the public key sizes as FuLeeca, but larger signature sizes. In fact, we can observe that the signature sizes of FuLeeca are roughly 50% smaller. The total size of FuLeeca is less than 75 % of the total size of Dilithium.

We note that Falcon achieves both smaller public keys and signature sizes, which in the total size gives roughly a reduction of 35% compared to FuLeeca.

Finally, we clearly outperform SPHINCS$^+$ in terms of signature size and total size, with a reduction of roughly 70% in total size.

Table 4: Comparison between FuLeeca and Dilithium, Falcon, SPHINCS$^+$ for provided security levels. All sizes are given in bytes.

level	scheme	public key size	signature size	total size
I	SPHINCS$^+$	32	7856	7888
	Falcon	897	666	1563
	FuLeeca	1318	1100	2418
II	Dilithium	1312	2420	3732
III	SPHINCS$^+$	48	16224	16256
	Dilithium	1952	3293	5245
	FuLeeca	1982	1620	3602
V	SPHINCS$^+$	72	29792	29864
	Dilithium	2592	4595	7187
	Falcon	1793	1280	3073
	FuLeeca	2638	2130	4768

6 Security Analysis

In this section, we assess the security of FuLeeca. The analysis consists of three parts. We begin by considering the generic solvers for finding codewords of given Lee weight. The second part describes known attacks and our countermeasures. The third part discusses the applicability of lattice reduction algorithms to solve the hard computational problems underlying this system. Taking all mentioned attacks into account we determine the presented parameters to achieve the security levels required by NIST.

6.1 Hardness of Underlying Problem and Generic Solvers

The adversary can attempt to recover the secret key from the public key, which is known as a key recovery attack. For FuLeeca, this is equivalent to finding any of the the rows of the secret generator matrix, which are of weight w_{key}. Alternatively, the attacker can try to forge a signature directly, without knowledge of the secret key. Forging a signature of FuLeeca is, therefore, equivalent to finding a low Lee weight codeword that satisfies both the number of required matches and the weight restriction.

Hence, both attacks require solving instances of the finding a codeword of given Lee weight problem, which is formally defined as follows.

Problem 6 (Finding Codeword of Given Lee Weight)**.** Given $\boldsymbol{H} \in \mathbb{F}_p^{(n-k)\times n}$ and $w \in \mathbb{N}$ find a $\boldsymbol{c} \in \mathbb{F}_p^n$ such that $\boldsymbol{c}\boldsymbol{H}^\top = \boldsymbol{0}$ and $\mathrm{wt}_L(\boldsymbol{c}) = w$.

This problem has first been studied in [37]. Problem 6, i.e., finding codewords of given weight is equivalent to the decoding problem. The decisional version of this problem has been proven to be NP-complete in [53].

Several algorithms have been proposed to solve this problem, they all belong to the family of Information Set Decoding (ISD) algorithms.

Remark 7. *Note that ISD algorithms can be formulated such that they solve the syndrome decoding problem, that is: given a parity-check matrix $\boldsymbol{H} \in \mathbb{F}_p^{(n-k)\times n}$, a syndrome $\boldsymbol{s} \in \mathbb{F}_p^{n-k}$ and a target weight*

t, they find an error vector $\boldsymbol{e} \in \mathbb{F}_p^n$, such that $\boldsymbol{H}\boldsymbol{e}^\top = \boldsymbol{s}^\top$ and $\mathrm{wt}(\boldsymbol{e}) = t$. Thus, by setting $\boldsymbol{s} = \boldsymbol{0}$, we can use such solvers to find codewords of weight t. However, note that Prange's algorithm [46] searches for a transformed syndrome $\boldsymbol{s}' = \boldsymbol{s}\boldsymbol{U}$, for some invertible $\boldsymbol{U}$ and wants the transformed syndrome to have weight t. As this is never satisfied for $\boldsymbol{s} = \boldsymbol{0}$, Prange cannot be used to find codewords of given weight. However, all improvements upon Prange, such as Stern/Dumer [52, 26], MMT [41], BJMM [11] try to first enumerate the error vector in the information set and then check whether the remaining vector has the remaining weight. This can also be applied to $\boldsymbol{s} = \boldsymbol{0}$.

ISD algorithms make use of an information set of the code, where one assumes a small weight and thus constructs lists of these partial solutions.

Let us quickly recall the main steps of an ISD algorithm. Given $\boldsymbol{H} \in \mathbb{F}_p^{(n-k)\times n}$, choose an information set I and bring $\boldsymbol{H}$ into a partial systematic form. For this, let J be a set of size $k + \ell$, which contains the information set I and transform $\boldsymbol{H}$ as

$$\boldsymbol{U}\boldsymbol{H}\boldsymbol{P} = \tilde{\boldsymbol{H}} = \begin{pmatrix} \boldsymbol{I}_{n-k-\ell} & \boldsymbol{H}_1 \\ 0 & \boldsymbol{H}_2 \end{pmatrix},$$

where $\boldsymbol{U} \in \mathbb{F}_p^{(n-k)\times(n-k)}$ is an invertible matrix and $\boldsymbol{P} \in \mathbb{F}_p^{n\times n}$ is a permutation matrix. Thus, we also split the unknown solution $\boldsymbol{c}$ into the indices J and J^C, i.e., $\boldsymbol{c}\boldsymbol{P}^\top = (\boldsymbol{c}_1, \boldsymbol{c}_2)$. Assuming that $\boldsymbol{c}_2$ has Lee weight v, we get the following two equations:

$$\begin{aligned} \boldsymbol{c}_1 + \boldsymbol{c}_2\boldsymbol{H}_1^\top &= 0 \\ \boldsymbol{c}_2\boldsymbol{H}_2^\top &= 0. \end{aligned}$$

Thus, we can first solve the second equation, $\boldsymbol{c}_2\boldsymbol{H}_2^\top = 0$ with $\mathrm{wt}_L(\boldsymbol{c}_2) = v$ as we then can easily check if the missing part $\boldsymbol{c}_1$ has the remaining Lee weight, by $\mathrm{wt}_L(\boldsymbol{c}_2\boldsymbol{H}_1^\top) = w - v$.

In [53], several algorithms have been presented to solve the smaller instance, namely using Wagner's approach of a set partitioning and using representation technique. In [19], the authors presented the amortized Wagner's approach.

Finally, in [10] the authors presented an adaption of these algorithms, taking into account that a random low Lee weight codeword has the exponential weight distribution observed in [9]. In these papers, it has been observed, that the amortized BJMM approach attains the lowest computational cost, and thus we consider this algorithm to compute the security level of the proposed parameters.

For the details of the algorithm, we refer to [10]. Mathematica programs to compute the computational costs of BJMM are publicly available[2] or for Wagner's cost here[3].

We adapted the program which computes the classical asymptotic cost c in the form $2^{c\cdot n}$, by considering the cost $c/2$ on a capable quantum computer (see [13, 19]).

Since we sample the secret basis for the generator matrix using the typical Lee sets, i.e., any $x \in \mathbb{F}_p$ occurs in the sought-after error vector $\boldsymbol{e}$ in $f(p_{w_{key}}(x)\cdot n)$ number of times, it makes sense to use this information in an ISD algorithm. However, as shown in [10], the amortized BJMM algorithm outperforms even the attempts to use restricted balls in case, where we are beyond the unique decoding radius. Thus, we build

[2] https://git.math.uzh.ch/isd/lee-isd/lee-isd-algorithm-complexities/-/blob/master/Lee-ISD-restricted.nb

[3] https://github.com/setinski/Information-Set-Decoding-Analysis

our security analysis on this fastest known algorithm, taking into account also polynomial speedups due to the quasi-cyclic structure [50].

6.2 Analysis of the Algorithm with Respect to Known Attacks

As required by NIST, we assume that an attacker has access to up to 2^{64} signatures for chosen messages. Such multi-use scenarios require an existential unforgeability under chosen message attack (EUF-CMA) security proof. For Hash-and-Sign approaches, EUF-CMA security proofs are notoriously difficult. Unfortunately, we cannot provide one at the moment.

We prevented possible leakages and vulnerabilities via the *Concentrating* procedure. These considerations are described in more detail below. Note that the *Concentrating* procedure at the moment does not involve a threshold on how close the valid signatures have to be to the target LMP. This flexibility might be of use in a future EUF-CMA security proof. Additionally, the scheme does not involve rejection sampling, which might be helpful to strengthen security, as soon as new attack vectors are known.

Exploiting additional knowledge given to the attacker in form of signatures is perhaps the most common way to attack Hash-and-Sign based signature schemes. In fact information leaked by the signatures has repeatedly been used to retrieve the private key. To give an example, successful attacks on the schemes [6, 44] have been presented in [48, 24]. Specifically, these attacks exploit the fact that for the proposed schemes in the Hamming metric a basis vector as well as the signatures have low weight, i.e., a small support. The main problem in the design of these attacked schemes was that the supports of the published signatures correlate with the private key.

We consider attacks exploiting leakage via published hash/signature pairs.

Support-based attacks such as those mentioned cannot be applied to FuLeeca as in the Lee metric vectors of low Lee weight do not necessarily have a small Hamming support. In fact, by putting the weight of the secret generators on the GV bound, we may even treat the resulting code as a random code.

This thwarts Hamming-metric attacks as the secret generators and the signatures have close to full Hamming weight.

Setting a sufficiently high threshold for the number of required sign matches prevents that a previously published signature can be directly used to sign another message. An obvious generalization of this reuse attack is creating linear combinations of existing signatures to forge new signatures. Note, however, that with overwhelming probability the Lee weight of the resulting vector will be too large to be accepted by the verifier. Hence, such an attack, which is similar to performing a sieving algorithm known from lattice-based cryptography, requires complexity which is exponential in the code parameters.

Notably the works [34, 40] show that finding a codeword of lower Lee weight in a quasi-cyclic code is significantly easier in case the code dimension $n/2$ is a composite number. In fact the security reduces to the codeword finding problem in a quasi-cyclic code with dimension equal to the smallest factor of $n/2$. Therefore, for all considered parameter sets in this work, we choose $n/2$ to be prime.

To avoid leakage via published hash/signature pairs we integrated a specific procedure into the signing algorithm, which we refer to as the *Concentrating* procedure. In the following, we first examine the signing algorithm without applying the specified *Concentrating* procedure. We randomly draw $k = 500$ salts and messages and observe the corresponding outputs of the hash-function $h_1, \dots, h_k$, i.e., $h_\ell = \text{Hash}(\text{salt}||m_\ell)$. For two different private keys we compare the Lee weights and sign matches of the corresponding signatures after just applying “Simple Signing”.

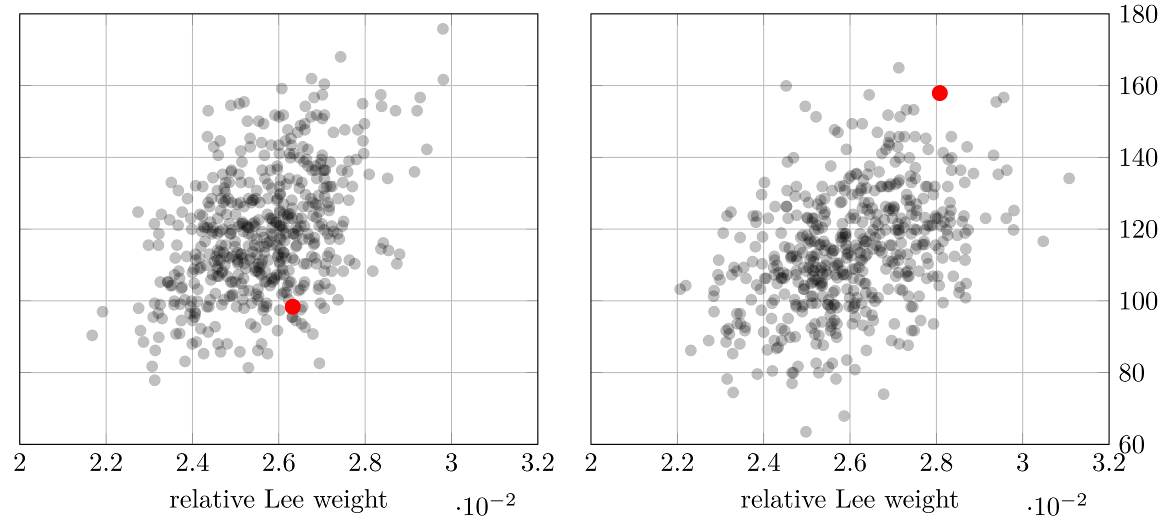

Figure 1: Evaluation of 500 signatures for simulated hashes (i.i.d uniform) using *two different keys* (left and right) after application of “Simple Signing”.

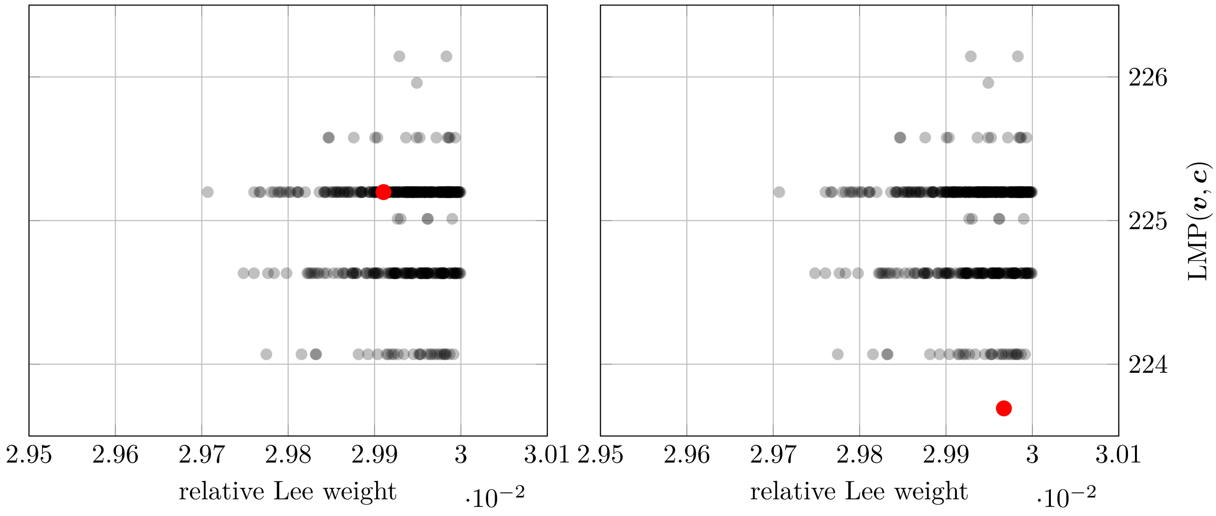

Figure 2: Evaluation of 500 signatures for simulated hashes (i.i.d uniform) for *two different keys* after application of both “Simple Signing” and “Concentrating”.

Figure 1 shows the relation between the relative Lee weights and the LMP between the codeword and the target vector, which is the hash of the message. Since the signature algorithm effectively correlates the secret key and the hashes it appears to be possible to learn at least some information about the secret key based on the distribution of resulting codewords in this Lee weight / LMP space.

The distribution of signatures for both private keys of Figure 1 show that the LMP between hash and codeword as well as the resulting Lee weights vary significantly and depend on the secret key. Since we are using two different private keys, we obtain two different signatures for each of the hashes. To exemplify this, we marked the resulting signatures before the *Concentrating* procedure for the same hash (the red dots) but using different private keys in Figure 1. Even though we do not provide a specific attack exploiting this behavior, the results suggest that some information about the private key is leaked and can potentially be exploited to help in the process of recovering the secret key.

Figure 2 shows the distribution of LMP values and relative Lee weights for the same hashes as in Figure 1

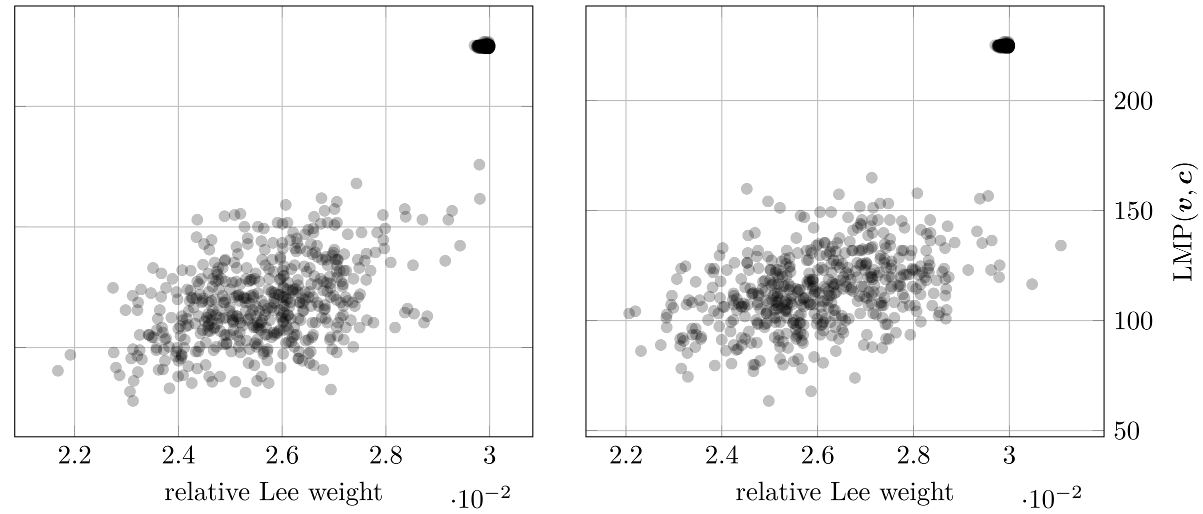

Figure 3: Evaluation of 500 signatures for simulated hashes (i.i.d uniform) using *two different keys* both after application of the "Simple Signing" part of Algorithm 2 and as well as applying the "Concentrating" procedure (dense clusters in the upper right).

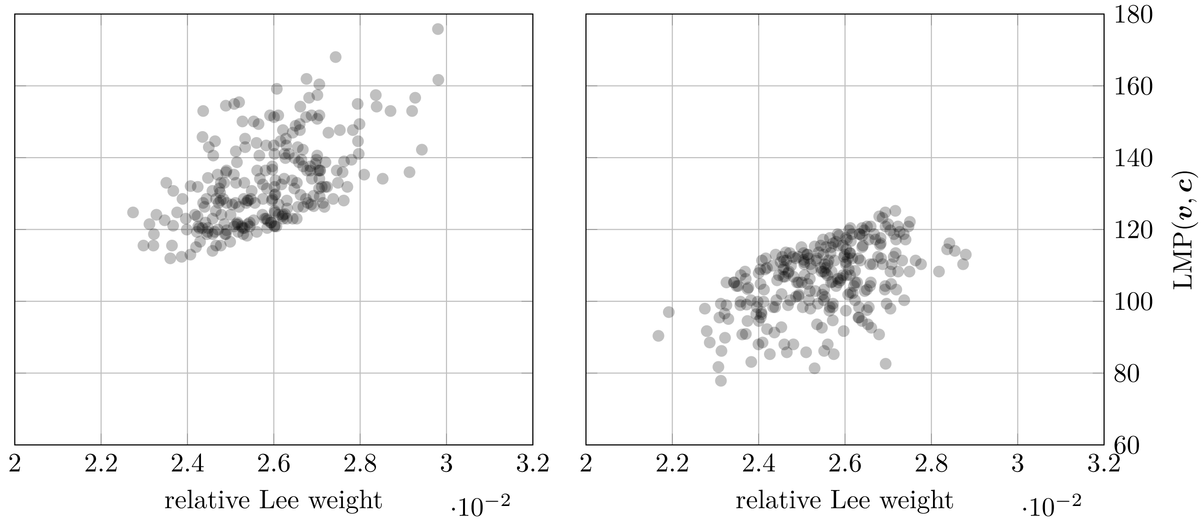

Figure 4: Evaluation of 500 signatures for simulated hashes (i.i.d uniform) before applying the *Concentrating* procedure. Unlike the previous figures, all of the displayed signatures were created using a *single key*. The vectors are divided into two (nearly equally large) groups, where the ratio between the log probability (LMP) and the Lee weight is above average (left), respectively below average (right).

after the *Concentrating* part of Algorithm 2 has been completed. The difference between the distributions for the different secret keys shall be as small as possible to minimize leakage about the secret key.

As in Figure 1, we marked the signatures for the same hashes and different secret keys, this time after the *Concentrating* procedure in Figure 2. The results show that the *Concentrating* procedure significantly reduces the leakage observable via the relative Lee weight / LMP map.

Figure 3 provides the information observable from Figure 1 and Figure 2 within a single plot to further illustrate the effect of the *Concentrating* procedure.

Similarly, we also observe that the shape of the distribution of signatures in the Lee weight / LMP space does not appear to meaningfully depend on the distribution of the same signatures after "Simple

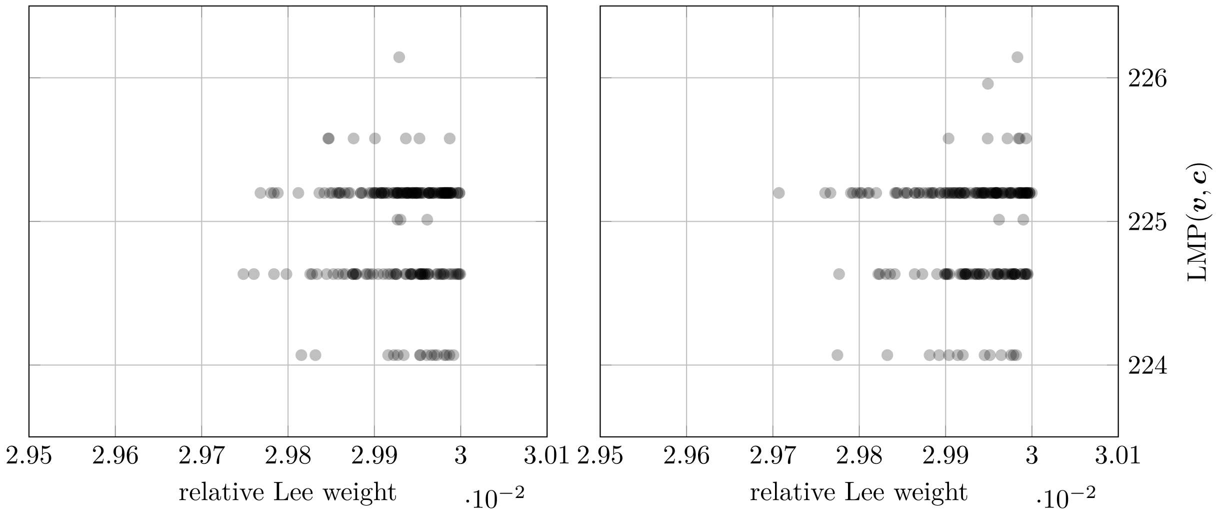

Figure 5: The same two sets of hashes for *the same key* (as in figure 4) after applying the "Concentrating" algorithm.

Signing". This is demonstrated in Figure 4 and 5 where for a single key we apply "Simple Signing" to the same set of hashes as before but split the signatures into two groups of almost equal size.

For group one (left hand side of Fig. 4) obtaining a codeword with the required LMP after application of the *Concentrating* procedure is expected to be easier than for group two (right hand side of Fig. 4) since in terms of the ratio between the log probability (LMP) and the Lee weight all of these are above average, while group two is below average. In fact, the percentage of hashes in group two that lead to a valid signature (right hand side of Fig. 5) in the end is slightly lower than for group one (left hand side of Fig. 5). However, this behaviour is to be expected for effectively every private key and, thus, this does not reveal any useful information about any chosen key in particular.

6.3 Lattice-based Attacks

Since the Lee metric is close to the Euclidean metric used in lattice-based cryptography, one has to study the known combinatorial attacks therein. In fact, the Lee metric corresponds to the L_1-norm, whereas the Euclidean metric corresponds to the L_2-norm. It is well known [47] that problems with respect to the L_2-norm can be reduced to problems with respect to any other L_p-norm. This result translates to: any algorithm solving a problem in the L_p-norm can also be used to solve the problem in the L_2-norm. Or as stated in [47]: "our main result shows that for lattice problems, the L_2-norm is the easiest." Thus, one can use the Lee-metric ISD algorithms to solve lattice-based problems in the Euclidean metric. It is unknown, whether the reverse direction is also possible, i.e., whether there exists a reduction from problems with respect to the L_1-norm to problems with respect to the L_2-norm. This is, however, exactly the direction required in order to use lattice-based algorithms to solve problems in the Lee metric.

To the best of our knowledge the only sieving algorithm in the L_1-norm is provided in [17], where the authors provide an $(1 + \varepsilon)$ approximation algorithm for the closest vector problem for all L_p-norms that runs in $(2 + 1/\varepsilon)^{\mathcal{O}(n)}$. The asymptotic cost of this algorithm does not outperform the considered Lee-metric ISD algorithms.

Another lattice-based approach is to search for the codeword of lowest Euclidean weight, e.g., using the BKZ algorithm [49]. Since we set the weight of the secret generators on the GV bound and thus assume that our code behaves like a random code, it is not known whether the codeword of lowest Euclidean weight is also the codeword of lowest Lee weight, i.e., the secret key. Under the conservative assumption that this is indeed the case, we estimate the cost of BKZ for the full rank lattice to be in $\mathcal{O}\left(2^{0.292n}\right)$. We observe that the parameter sets we choose attain the target security levels also according to this attack.

Assumption 1: Let us use BJMM to find a vector $\boldsymbol{v}$ of Lee weight w_{sig}. We assume that finding another vector $\boldsymbol{v'}$ of equal Euclidean length, i.e., $||\boldsymbol{v}||_2 = ||\boldsymbol{v'}||_2$, by using BKZ has a lower complexity than finding $\boldsymbol{v}$ using BJMM. If this assumption did not hold, then using BJMM we would be able to achieve a speedup in solving SVP compared to using BKZ, which would in turn affect all lattice-based cryptosystems.

Assumption 2: We assume that the complexity of using BKZ to find a vector having Lee weight less than or equal to w_{sig} is higher compared to using BJMM for this task.

For a Lee weight of w_{sig} the consequence of Assumption 2 not holding is that BKZ would outperform all known ISD algorithms for solving the given weight codeword finding problem at that weight.

BKZ requires orthogonal projections within the LLL step. However, the L_1 norm is not induced by a scalar product and, therefore, we assume that the best way to use BKZ for finding short vectors in the L_1 norm is to use it for finding short vectors in L_2 norm and to hope that those are also short enough in the L_1 norm. We assume that using BJMM to find short vectors in the L_1 norm is more efficient than this.

6.4 Description of Expected Security Strength

We assume that the used Hash functions are cryptographically secure.

The best known attack to find a codeword of given Lee weight given our public key $\boldsymbol{G}_{pub}$ is Information Set Decoding using the quantum, amortized BJMM algorithm in the Lee metric.

Recall that the choice to set w_{key} on the Lee-metric GV bound is necessary, to treat the public code as a random code and thus estimate the BKZ algorithms cost at $2^{0.292n}$.

We choose $p = 65\,521$, in order to set the Lee weight w_{key} of the secret generators on the Lee-metric GV bound and still have a large enough distance to the Lee weight of the signatures w_{sig}. In fact, for smaller choices of p and setting w_{key} on the Lee-metric GV bound we cannot find enough sign matches to signatures of Lee weight w_{sig} with $w_{sig} < 0.2$. The bound $w_{sig} < 0.2$ is mandatory to avoid a polynomial time cost of ISD algorithms.

For the choice of $p = 65\,521$, one cannot explicitly compute the cost of the BJMM algorithm using the program[4] due to numerical instabilities. A conservative extrapolation from results for smaller choices of p suggests that the cost for BJMM at $w_{sig} = 0.03$ lies at $2^{0.08n}$. We want to note here that Wagner's algorithm implies a cost of $2^{0.5n}$.

We choose the length n according to the BKZ algorithm on full-rank lattices, which runs with a cost of $2^{0.292n}$. We aim at the conservative classical security levels $\lambda_1 = 160, \lambda_3 = 224, \lambda_5 = 288$ and set n at least such that

$$2\lambda_i + 64 = 0.292n.$$

This choice is conservative in two ways. Not only the security levels λ_i have been chosen conservatively

[4] https://git.math.uzh.ch/isd/lee-isd/lee-isd-algorithm-complexities/-/blob/master/Lee-ISD-restricted.nb

but also assuming a loss in the security level of λ_i for each of the provided 2^{64} signature vectors is a very conservative approach within the estimation of the resulting security level. In fact, the parameters are chosen in such a way that even for the aforementioned loss of $\lambda_i + 64$ bits a security level of of at least λ_i bits is maintained for the respective parameter sets. It is possible to speed up solving the SVP using BKZ by providing the algorithm with short Euclidean lattice vectors [22]. The obtainable speedup is upper bounded by the cost of finding the provided lattice vectors since otherwise we would have found an improved lattice reduction algorithm. The exact speedup obtained from integrating the short (codeword) vectors depends on their Euclidean length, but we assume that a vector of comparable Euclidean length can be obtained at a lower cost using BKZ compared to using BJMM. We conservatively add 64 to account for the maximum possible speedup once 2^{64} signatures have been published.

In fact, we choose n even slightly larger to ensure that we reach the necessary LMP with good probability. This leads to the following lengths: $n_1 = 1318$, which ensures that $n_1/2 = 659$ is prime, $n_3 = 1982$, which ensures that $n_3/2 = 991$ and finally $n_5 = 2638$, which ensures that $n_5/2 = 1319$ is prime.

Parameter Choice I The parameter choice $p = 65\,521$, $n = 1318$, $\omega_{sig} = w_{sig}/(nM) = 0.03$, $\omega_{key} = w_{key}/(nM) = 0.001437$ leads at least to the desired quantum cost of 2^{80}, since BJMM's algorithm indicates a quantum complexity of $2^{80} = 2^{0.08n}$ operations and the BKZ algorithm requires at least a classical complexity of $2^{384} = 2^{0.92n}$.

Parameter Choice III The parameter choice $p = 65\,521$, $n = 1982$, $\omega_{sig} = w_{sig}/(nM) = 0.03$, $\omega_{key} = w_{key}/(nM) = 0.001437$ leads to the desired quantum cost of 2^{112}, since BJMM's algorithm indicates a quantum complexity of $2^{112} = 2^{0.08n}$ operations and the BKZ algorithm requires at least a classical complexity of $2^{578} = 2^{0.292n}$.

Parameter Choice V The parameter choice $p = 65\,521$, $n = 2638$, $\omega_{sig} = w_{sig}/(nM) = 0.03$, $\omega_{key} = w_{key}/(nM) = 0.001437$ leads to the desired quantum cost of 2^{144}, since BJMM's algorithm indicates a quantum complexity of $2^{144} = 2^{0.08n}$ operations and the BKZ algorithm requires at least a classical complexity of $2^{770} = 2^{0.292n}$.

7 Implementation Details

The reference implementation contains comments, which in combination with the pseudo-code in this document, should help in understanding the reference implementation. In the following sections we describe the design rationale of particular functions in the reference implementation in more detail.

7.1 Sampling of Secret Vectors during Key Generation

In order to sample the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ during Key Generation (corresponding to line 1 of Algorithm 1), we generate a uniformly random permutation of the typical Lee set (Definition 5). We implement this by storing the typical set as an array and uniformly permuting its entries using the Fisher-Yates shuffle [39]. Since a naive implementation of the algorithm leaks the permutation through its secret-dependent memory accesses, we implement a constant-time variant that has been published in [51]. The result of this algorithm, shown in Algorithm 4, is an array of indices that correspond to the permutation. In a second step these indices are applied onto the typical set in a constant time manner by always iterating over all elements of the typical set and performing a conditional assignment.

Algorithm 4: Shuffle Indices (Fisher-Yates Variant)[51, Algorithm 3]

```
Initialize pos = $[0, \dots, n/2-1]$
for $i \leftarrow n/2-1$ downto $0$ do
    pos[i] $\xleftarrow{\$} \{i, \dots, n/2-1\}$
    for $j \leftarrow i+1$ to $n/2-1$ do
        if pos[j] = pos[i] then
            pos[$j$] $\leftarrow i$
        else
            pos[$j$] $\leftarrow$ pos[$j$]
    end
end
Output: pos
```

7.2 Hashes and CSPRNGs

As FuLeeca follows the Hash-and-Sign approach, the signature is generated and verified using a digest of the message $\boldsymbol{m}$, which is then extended to a vector $\boldsymbol{c}$ before being actually used. For practical reasons, we pre-hash the message $\boldsymbol{m}$ using a corresponding SHA-3 hash function, as specified in FIPS 202 [43], with digest size of 2λ. Afterwards, we expand this message digest together with a salt using the eXtendable-Output Function (XOF) SHAKE256 from the FIPS 202 specification [43] as CSPRNG. The rationale for pre-hashing the message $\boldsymbol{m}$ to $\boldsymbol{m}'$ is that the signing procedure described in Algorithm 2 iterates multiple times until a valid signature is found. In each iteration, a new vector $\boldsymbol{c}$ must be generated that requires input from the message $\boldsymbol{m}$ and a salt.

Having to process the message multiple times is inefficient for large messages. The pre-hashing allows to reduce this effort to one digest computation with a potentially large input message and reduces the computational effort within the iteration loop. Another advantage of this approach is that the computation of the message digest can be externalized to a dedicated co-processor, which is especially relevant if large messages need to be signed on a resource constrained device.

We furthermore decided to restrict our choice to the Keccak-based SHA-3 primitives, as this allows to share resources for both the hash computation of the message and the CSPRNG, reducing the overhead of code size and HW resources in a HW/SW co-design.

7.3 Polynomial Inversion

The polynomial inversion is one of the key components of the key generation. The public key $\boldsymbol{T}$ is defined as the multiplication of the inverse of $\boldsymbol{A}^{-1}$ and $\boldsymbol{B}$. Due to the quasi-cyclic nature of $\boldsymbol{A}$ and $\boldsymbol{B}$, the inverse can be calculated as the inverse of the polynomial $\boldsymbol{a}$. As the polynomial to be reversed is part of the secret key, it is crucial that the runtime of the algorithm is independent of the input polynomial. To achieve this, we mainly follow the ideas outlined by Bernstein and Yang in [14].

7.4 Number of Concentrating Iterations

To refine the signature candidate, a number of additional additions/subtractions of key lines needs to be executed. One possibility here is to allow an undefined number of iterations and to stop as soon as the candidate reaches the targeted Lee weight. One would then check the LMP of this candidate and publish it in case the candidate reaches the desired level. The major downside of this approach is the timing side-channel it opens which might, as the number of iterations depend on the secret key and the challenge, leak information about the key. The approach we choose is to fix the number of iterations with the downside that the Lee weight as well as the LMP might not be in the specified range after this number of iterations. To increase the probability for generating a codeword in the specified range we allow additions/subtractions of rows to be undone after the codeword has reached a Lee weight greater than $w_{sig} - w_{key}$ (see lines 19 and 22 within Algorithm 2).

7.5 Scaling the Information Vector

The vector $\boldsymbol{x}$ obtained in line 9 of the signing algorithm needs to scaled to be useable as an information vector to generate a signature candidate in line 12. As this scaling factor s needs to be smaller than one and a division by an arbitrary integer is known to be challenging to implement in constant time, the factor s is approximated by a multiplication with an integer followed by right shift, emulating a division by a power of two.

7.6 Encoding and Decoding

The coefficients that constitute a signature before encoding follow a Gaussian-like distribution centered at zero. This fact allows to reduce the signature size by compressing the signature and encode it in a bitstring. For that, we use the same approach as proposed in the Falcon signature scheme [32]. That is, each coefficient is converted into its signed representation and split into a *tail* and *head*. The coefficient's sign bit is concatenated with the uncoded tail, as this tail is approximately uniformly distributed and thus cannot be compressed efficiently. The remaining bits in the coefficient's head are then encoded in a 0^k1 fashion, that is a sequence of k zeroes and a one, where k is the value of the head.

For a tail size of 9-bits, this yields signatures of 1100B, 1620B and 2130B for NIST security level I, III and V, respectively, including the salt. Note, that these sizes are already zero-padded to have a fixed signature size. The final sizes of the signatures are obtained by the worst-case size observed after generating 1000 signatures and adding a sufficient margin on top of it.

8 Detailed Performance Analysis

Table 5 shows the required clock cycles and run time in milliseconds for the reference implementation of the algorithm averaged over 100 runs. These values were obtained on an Ubuntu 22.04 machine with an Intel Comet Lake (Intel Core i7-10700) CPU at its base frequency of 2900 MHz and 64 GB of RAM using GCC version 11.3.0 and an O3 optimization. In order to generate reliable results, all dynamic performance enhancement and power management features like hyper threading, turbo boost, and dynamic undervolting of the CPU were disabled. Clock cycles are measured using the internal performance registers of the CPU using the library *libcpucycles*[5].

9 Known Answer Test Values

Known Answer Tests (KAT) have been generated and are available in the KAT folder in the submission package.

[5] The implementation is publicly available at `https://cpucycles.cr.yp.to/`.

Table 5: Runtime of the reference implementation in kilocycles and milliseconds on an Intel Comet Lake with a base frequency of 2900 MHz averaged over 100 runs.

NIST cat.	Unit	Keygen	Sign	Verify
I	kCycles	49 354	1 846 779	1260
	ms	17	636	0.43
III	kCycles	110 918	2 111 156	2447
	ms	38	727	0.84
V	kCycles	192 388	12 327 726	3789
	ms	66	4250	1.31

10 Advantages and Limitations

The major advantages of FuLeeca compared to other post-quantum secure schemes are its signature and key sizes. In terms of the sum of public-key size and signature size, we achieve better parameters than the schemes Dilithium [25], SPHINCS$^+$ [5], and the state-of-the-art of code-based signature schemes. Furthermore, our submission is neither lattice- nor hash-based and, therefore, presents an alternative in case a major breakthrough in the cryptanalysis of lattice-based schemes occurs in the future. Another major advantage of FuLeeca is that all of its elementary steps are very easy to implement in software and hardware. As we only rely on basic integer arithmetic, i.e., additions and multiplications, we do not require support for floating point operations. However, relying solely on these basic operations poses a performance drawback when compared to other alternatives, e.g. lattice-based signature schemes using NTT.

A limitation of FuLeeca is that we are not able to provide a full EUF-CMA security proof. However, we show why our scheme is secure against known attacks and furthermore heuristically observe that the scheme does not leak information about the key via the standard attack vectors. Another limitation of our scheme is that the Lee metric has not enjoyed the same attention by the cryptographic community as the Euclidean metric of lattice-based schemes. However, NIST explicitly encouraged the use of alternative metrics.

11 Acknowledgments

We would like to thank Sabine Pircher, Thomas Debris-Alazard and Wessel van Woerden for meaningful discussions.

Violetta Weger is supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 899987. Sebastian Bitzer, Georg Maringer, Stefan Ritterhoff and Antonia Wachter-Zeh were supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under Grant No. WA3907/4-1, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 801434), and acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KISK002. Patrick Karl acknowledges the financial support by the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KISK002.

12 Bibliography

[1] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the SDitH. *Cryptology ePrint Archive*, 2022.

[2] Nicolas Aragon, Paulo SLM Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. Bike: bit flipping key encapsulation. 2017.

[3] Nicolas Aragon, Olivier Blazy, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Durandal: a rank metric based signature scheme. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*, pages 728–758. Springer, 2019.

[4] Jaakko Astola. On the asymptotic behaviour of Lee-codes. *Discrete applied mathematics*, 8(1):13–23, 1984.

[5] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS$^+$, submission to the NIST post-quantum project, v.3. 2020.

[6] Marco Baldi, Marco Bianchi, Franco Chiaraluce, Joachim Rosenthal, and Davide Schipani. Using LDGM codes and sparse syndromes to achieve digital signatures. In *Post-Quantum Cryptography: 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings 5*, pages 1–15. Springer, 2013.

[7] Marco Baldi, Sebastian Bitzer, Alessio Pavoni, Paolo Santini, Antonia Wachter-Zeh, and Violetta Weger. Zero knowledge protocols and signatures from the restricted syndrome decoding problem. *Cryptology ePrint Archive*, 2023.

[8] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. LESS-FM: fine-tuning signatures from the code equivalence problem. In *Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20–22, 2021, Proceedings 12*, pages 23–43. Springer, 2021.

[9] Jessica Bariffi, Hannes Bartz, Gianluigi Liva, and Joachim Rosenthal. On the properties of error patterns in the constant Lee weight channel. In *International Zurich Seminar on Information and Communication (IZS 2022). Proceedings*, pages 44–48. ETH Zurich, 2022.

[10] Jessica Bariffi, Karan Khathuria, and Violetta Weger. Information set decoding for Lee-metric codes using restricted balls. In *Code-based Cryptography 10th International Workshop, CBCrypto 2022*. Springer.

[11] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1+1=0$ improves information set decoding. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 520–536. Springer, 2012.

[12] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with RSA and Rabin. In *Eurocrypt*, volume 96, pages 399–416. Springer, 1996.

[13] Daniel J Bernstein. Grover vs. McEliece. In *Post-Quantum Cryptography: Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings 3*, pages 73–80. Springer, 2010.

[14] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. Cryptology ePrint Archive, Paper 2019/266, 2019. https://eprint.iacr.org/2019/266.

[15] Ward Beullens. Sigma protocols for mq, pkp and sis, and fishy signature schemes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 183–211. Springer, 2020.

[16] Loïc Bidoux and Philippe Gaborit. Shorter signatures from proofs of knowledge for the SD, MQ, PKP and RSD problems. *arXiv preprint arXiv:2204.02915*, 2022.

[17] Johannes Blömer and Stefanie Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theoretical Computer Science*, 410(18):1648–1665, 2009.

[18] Eimear Byrne, Anna-Lena Horlemann, Karan Khathuria, and Violetta Weger. Density of free modules over finite chain rings. *Linear Algebra and its Applications*, 651:1–25, 2022.

[19] André Chailloux, Thomas Debris-Alazard, and Simona Etinski. Classical and Quantum algorithms for generic Syndrome Decoding problems and applications to the Lee metric, 2021. Report Number: 552.

[20] Jinkyu Cho, Jong-Seon No, Yongwoo Lee, Zahyun Koo, and Young-Sik Kim. Enhanced pqsigRM: Code-based digital signature scheme with short signature and fast verification for post-quantum cryptography. *Cryptology ePrint Archive*, 2022.

[21] Nicolas T Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a McEliece-based digital signature scheme. In *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7*, pages 157–174. Springer, 2001.

[22] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. Lwe with side information: attacks and concrete security estimation. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, pages 329–358. Springer, 2020.

[23] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I*, pages 21–51. Springer, 2019.

[24] Jean-Christophe Deneuville and Philippe Gaborit. Cryptanalysis of a code-based one-time signature. *Designs, Codes and Cryptography*, 88:1857–1866, 2020.

[25] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium – algorithm specifications and supporting documentation (version 3.1). 2021.

[26] Il'ya Isaakovich Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25(1):24–32, 1989.

[27] Thomas Espitau, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Shorter hash-and-sign lattice-based signatures. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*, pages 245–275. Springer, 2022.

[28] Jean-Charles Faugere, Valérie Gauthier-Umana, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate McEliece cryptosystems. *IEEE Transactions on Information Theory*, 59(10):6830–6844, 2013.

[29] Thibauld Feneuil. Building MPCitH-based signatures from MQ, MinRank, Rank SD and PKP. *Cryptology ePrint Archive*, 2022.

[30] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Shared permutation for syndrome decoding: New zero-knowledge protocol and code-based signature. *Designs, Codes and Cryptography*, pages 1–46, 2022.

[31] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. *Cryptology ePrint Archive*, 2022.

[32] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON: fast-fourier lattice-basd compact signatures over NTRU, specification v1.2. 2020.

[33] Danièle Gardy and Patrick Solé. Saddle point techniques in asymptotic coding theory. In *Algebraic Coding: First French-Soviet Workshop Paris, July 22–24, 1991 Proceedings*, pages 75–81. Springer, 2005.

[34] Craig Gentry. Key recovery and message attacks on ntru-composite. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*, pages 182–194. Springer, 2001.

[35] Danilo Gligoroski, Simona Samardjiska, Håkon Jacobsen, and Sergey Bezzateev. McEliece in the world of Escher. *Cryptology ePrint Archive*, 2014.

[36] Shay Gueron, Edoardo Persichetti, and Paolo Santini. Designing a practical code-based signature scheme from zero-knowledge proofs with trusted setup. *Cryptography*, 6(1):5, 2022.

[37] Anna-Lena Horlemann-Trautmann and Violetta Weger. Information set decoding in the Lee metric with applications to cryptography. *Advances in Mathematics of Communications*, 15(4), 2021.

[38] Kyungbae Jang, Anubhab Baksi, Hyunji Kim, Gyeongju Song, Hwajeong Seo, and Anupam Chattopadhyay. Quantum analysis of AES. *Cryptology ePrint Archive*, 2022.

[39] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms.* Addison-Wesley Longman Publishing Co., Inc., USA, 1997.

[40] Carl Löndahl, Thomas Johansson, Masoumeh Koochak Shooshtari, Mahmoud Ahmadian-Attari, and Mohammad Reza Aref. Squaring attacks on McEliece public-key cryptosystems using quasi-cyclic codes of even dimension. *Designs, Codes and Cryptography*, 80:359–377, 2016.

[41] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\mathcal{O}(2^{0.054n})$. In *Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17*, pages 107–124. Springer, 2011.

[42] Dustin Moody and Ray Perlner. Vulnerabilities of "McEliece in the World of Escher". In *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings 7*, pages 104–117. Springer, 2016.

[43] National Institute of Standards and Technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Technical report, jul 2015.

[44] Edoardo Persichetti. Efficient one-time signatures from quasi-cyclic codes: A full treatment. *Cryptography*, 2(4):30, 2018.

[45] Aurélie Phesso and Jean-Pierre Tillich. An efficient attack on a code-based signature scheme. In *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings 7*, pages 86–103. Springer, 2016.

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

[47] Oded Regev and Ricky Rosen. Lattice problems and norm embeddings. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing*, pages 447–456, 2006.

[48] Paolo Santini, Marco Baldi, and Franco Chiaraluce. Cryptanalysis of a one-time code-based digital signature scheme. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2594–2598. IEEE, 2019.

[49] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66:181–199, 1994.

[50] Nicolas Sendrier. Decoding One Out of Many. In *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 51–67, Berlin, Heidelberg, 2011. Springer.

[51] Nicolas Sendrier. Secure sampling of constant-weight words – application to bike. Cryptology ePrint Archive, Paper 2021/1631, 2021. https://eprint.iacr.org/2021/1631.

[52] Jacques Stern. A method for finding codewords of small weight. *Coding theory and applications*, 388:106–113, 1989.

[53] Violetta Weger, Karan Khathuria, Anna-Lena Horlemann, Massimo Battaglioni, Paolo Santini, and Edoardo Persichetti. On the hardness of the Lee syndrome decoding problem. *Advances in Mathematics of Communications*, April 2022. Publisher: Advances in Mathematics of Communications.