Quantum-safe HIBE: does it cost a LATTE?

Raymond K. Zhao[1], Sarah McCarthy[2,3], Ron Steinfeld[1], Amin Sakzad[1], and Máire O'Neill[3]

[1] Faculty of Information Technology, Monash University, Australia
`raymond.zhao,ron.steinfeld,amin.sakzad@monash.edu`

[2] Institute for Quantum Computing, University of Waterloo, Canada
`sarah.mccarthy@uwaterloo.ca`

[3] Centre for Secure Information Technologies, Queen's University Belfast, UK
`m.oneill@ecit.qub.ac.uk`

Abstract. In addition to providing quantum-safe traditional PKI, lattices support advanced primitives such as identity-based encryption (IBE). These schemes have shown promising results in terms of practicality, but still have disadvantages such as the reliance on a single master key. Hierarchical identity-based encryption (HIBE) schemes address this problem, as well as lending themselves to more realistic organisational structures. To date, several HIBE schemes over lattices have been proposed but there has been little in the way of practical evaluation.
This paper provides the first complete C implementation and benchmarking of LATTE, a promising HIBE scheme proposed by the United Kingdom (UK) The National Cyber Security Centre (NCSC) in 2017 and endorsed by European Telecommunications Standards Institute (ETSI). We also propose further optimisations for the KeyGen, Delegate, and sampling components of LATTE. As expected, the KeyGen, Extract, and Delegate components are the most time consuming, with Extract experiencing a 35% decrease in op/s from the first to second hierarchical level at 80-bit security. Our optimised implementation of the Delegate function takes 1 second at this security level on a desktop machine at 4.2GHz, significantly faster than the order of minutes estimated in the ETSI technical report. Furthermore, our optimised LATTE Encrypt/Decrypt implementation reaches speeds up to 4.6x faster than the ETSI implementation.

Keywords: lattice-based cryptography · hierarchical identity-based encryption · advanced primitives · software design

1 Introduction

One of the advantages of lattice-based cryptography (LBC) is the ability to build advanced cryptographic primitives such as identity-based encryption (IBE). An IBE scheme removes the need for a certificate repository by deriving a user's public key from their already established public identity. This provides a low latency setup with instantaneous communication capabilities. Furthermore, a hierarchy can be built into an IBE scheme to provide a more distributed workload and allow for finer grained control over private key distribution. Hierarchical identity-based encryption (HIBE) schemes extend the concept of using a personal identity as a public key to a multi-levelled scenario, such as one would find within a functioning company. HIBE has further applications such as forward-secure encryption [7] and public key broadcast encryption [13]. However, it is still unknown territory within the post-quantum field. Additionally, with the growth of the Internet of Things (IoT), which brings with it complex interconnected systems of constrained devices, there is a greater requirement for lightweight, advanced primitives unlike ever before. The long-term security considerations indicate that these should be made quantum-secure today. The aim of this paper is to assess the practicality of a quantum-safe HIBE scheme.

The DLP-IBE scheme [14] was in 2017 combined with the Bonsai tree HIBE scheme introduced in [8] to create Latte by Campbell and Groves [6]. This research was carried out by the National Cyber Security Centre (NCSC), with a view to utilising the scheme in UK public-safety communications. They are currently working with the European Telecommunications Standards Institute (ETSI) in a move towards standardising the scheme [1]. However, the proposed specification [1] only provides the Encrypt and Decrypt performance results, and it is unclear if Latte KeyGen, Delegate, and Extract are practical at all.

This paper provides the first performance benchmarking of a quantum-safe HIBE scheme, Latte, written in C.[4] We also identify bottlenecks, propose optimisations for Latte and consider its suitability for such applications. In more detail, the contributions of this paper are:

- **Optimised Latte KeyGen and Delegate algorithms:** We adapt the NTRUSolve function from Falcon [34] in order to efficiently solve the NTRU equation in our optimised Latte KeyGen algorithm. The NTRUSolve is both faster and more compact [33] than the resultant method [14] used in [1]. In addition, we adapt the technique from ModFalcon [11] and the length reduction technique by using Cramer's rule [1] in order to efficiently solve the NTRU equation for higher lattice dimensions in our optimised Latte Delegate algorithm.
- **Gaussian sampling algorithms for Latte:** We adapt the FFT sampling procedures from Falcon [34], which is faster than the Klein-GPV sampler [18] used in [1]. In addition, the proposed Latte specification [1] did not discuss the integer discrete Gaussian sampling techniques suitable for the needed standard deviations. We integrate efficient sampling techniques including FACCT [37] and COSAC [36] in our optimised Latte implementation.
- **New parameter set for Latte:** We provide slightly revised parameter sets for Latte, fixing a bug in the computation of a lattice smoothing parameter in the ETSI technical report [1], and also modify a Gaussian sampling standard deviation parameter to accommodate the more efficient FACCT

[4]The implementation source code is available at `https://gitlab.com/raykzhao/latte`.

[37] sampler for the Key Generation algorithm. Security estimates for these revised parameters is also presented.

- **Full implementation of LATTE:** We give the first complete performance results of a lattice-based HIBE scheme, including the KeyGen, Delegate, and Extract algorithms, for which the implementation results were unclear in [1]. The proposed specification [1] estimated that Delegate would have run-time in the order of minutes on a desktop machine. In this paper we show that an efficient implementation can perform the Delegate function in only a few seconds on a desktop machine.

Section 2.2 and 2.3 give the background to HIBE and the lattice-based concepts used in (H)IBE schemes. Section 3 describes the optimised LATTE HIBE scheme. Section 4 provides the security analysis. Section 5 discusses techniques in making the scheme practical for real world applications. Performance results for the scheme are given in Section 6.

2 Preliminaries

2.1 Underlying Mathematics and Notations

A lattice is a collection of integer linear combinations of a set of basis vectors. Lattices are one of the leading contenders for secure post-quantum cryptography, as indicated by the fact they account for 7 out of the 15 Round 3 candidates, and more specifically 5 of the 7 finalists in the NIST process [31]. Structure can be added to lattices, and cryptographic schemes may be built on standard lattices, ideal lattices, module lattices, or NTRU lattices. These additional structures bring advantages such as improved compactness and efficiency, but are still under analysis in terms of increased vulnerability to attacks. Popular underlying hard lattice problems believed to be secure against quantum computing attacks are the Shortest Vector Problem (SVP), Closest Vector Problem (CVP), and Learning With Errors (LWE) alongside its ring variant (over ideal lattices), Ring-LWE. These are all concerned with finding short vectors in the lattice which can be attempted to be solved by lattice reduction algorithms such as LLL [25] and BKZ [9,35]. Another common lattice problem is the NTRU assumption [21], that is, given a polynomial $\mathbf{h}$, one must find $\mathbf{f}, \mathbf{g}$ such that $\mathbf{h} = \mathbf{g} \cdot \mathbf{f}^{-1}$.

In this paper, vectors or, interchangeably through the canonical embedding, polynomials will be denoted by bold small letters like $\mathbf{f}$, matrices $\mathbf{M}$, polynomial ring mod $\mathbf{f}$ and q as $\mathcal{R}_q$, and lattices as Λ. The field of integers mod q is denoted as $\mathbb{Z}_q$. Discrete Gaussian distributions with centre c and standard deviation σ are denoted as $\mathcal{D}_{\sigma,c}$, and we omit the center i.e. $\mathcal{D}_\sigma$ if $c = 0$. The smoothing parameter of $\mathbb{Z}$ is denoted as $\eta_\epsilon(\mathbb{Z})$. The Euclidean norm of a vector/polynomial $\mathbf{f}$ is denoted $||\mathbf{f}||$. The transpose $\mathbf{f}^*$ of polynomial $\mathbf{f} = f_0 + f_1 x + \cdots + f_{N-1} x^{N-1}$ is defined as $\mathbf{f}^* = f_0 - f_{N-1} x - \cdots - f_1 x^{N-1}$. We denote $\mathbf{M}^*$ as the transpose of matrix $\mathbf{M}$ where $\mathbf{M}^*_{i,j} = (\mathbf{M}_{j,i})^*$. The Hermitian product of vectors $\mathbf{a}, \mathbf{b}$ is denoted as $\langle \mathbf{a}, \mathbf{b} \rangle$. The concatenation of several vectors $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N$ will be written as $(\mathbf{f}_1 | \mathbf{f}_2 | \cdots | \mathbf{f}_N)$. In HIBE schemes, user identities at level ℓ are here denoted ID_ℓ. A hash function from an arbitrary length input to a vector of integers of length N is written as $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^N$. An arrow $\leftarrow\!\$$ is used to show the uniform random sampling of an element from a set e.g. $\mathbf{f} \leftarrow\!\$\ \mathbb{Z}_q^N$. The operator $\oplus$ means XOR. A Gram-Schmidt orthogonalised basis is denoted as $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_N\}$. The notation $\mathcal{A}(\mathbf{f})$ refers to the anti-circulant matrix associated with polynomial $\mathbf{f}$. The notation $\lfloor k \rceil$ indicates the real number k is to be rounded to the nearest integer. The rounding $\lfloor \mathbf{f} \rceil$ of a polynomial $\mathbf{f}$ is taken to be coefficient-wise rounding. The Fast Fourier Transform (FFT) and Number Theoretic Transform (NTT) of polynomial $\mathbf{f}$ are the evaluations $\mathbf{f}(\zeta^i)$ for $i \in \{0, \ldots, N-1\}$, where ζ is the $2N$-th complex root of unity in the FFT, and ζ is the $2N$-th root of unity mod q in the NTT.

2.2 Hierarchical Identity-based Encryption

Hierarchical identity-based encryption (HIBE) schemes were introduced by Horwitz and Lynn [22] and can be considered a generalisation of an IBE scheme to multiple levels. An HIBE scheme consists of five components: Keygen, Delegate, Extract, Encrypt, and Decrypt. Figure 1 in Appendix A shows the relationship between the components of a 2-level HIBE scheme.

We begin with an introduction to non-quantum-safe HIBE schemes, which are based on elliptic curves and pairings. Gentry and Silverberg proposed the first secure HIBE scheme in the random oracle model (ROM) in 2002 [19], which was an extension of the Boneh-Franklin IBE scheme [5], a Weil-pairing based scheme, the security of which relies on the bilinear Diffie-Hellman problem. This was shown to be secure against adaptive identity and chosen ciphertext attacks, by use of the Fujisaki-Okamoto transformation [17], although the security degrades exponentially with the number of levels.

Boneh-Boyen built on this in 2004 to create a scheme which was secure without random oracles [3]. However, as both the ciphertext and private keys grew linearly with the number of levels of the hierarchy, in 2005 a scheme [4] was proposed which fixed the ciphertext size to three group elements, and curtailed private key growth to within level ℓ group elements. In 2018, an isogeny-based version of the Decisional Bilinear Diffie-Hellman-based scheme was proposed [24]. Despite isogenies possessing quantum-safe properties, this variant only serves to strengthen the existing classical security, by proving it secure under the assumption of *either* the classical version or the isogeny-based version of the problem and therefore is not necessarily quantum-safe. To the best of the author's knowledge, the only quantum-safe HIBE schemes so far proposed are based on lattices. We now introduce the schemes upon which LATTE is built.

2.3 The Ingredients of LATTE

DLP IBE Scheme: In 2014, Ducas, Lyubashevsky and Prest proposed the first efficient lattice-based identity-based encryption scheme [14]. They based their construction on the IBE scheme by Gentry, Peikert and Vaikuntanathan [18], using a variant of NTRU lattices. The underlying security problems are the

NTRU problem for key generation and R-LWE for the encryption. The ciphertexts therefore have more practical sizes than previous constructions, for example 30kbits for 192-bit classical security. The use of structured lattices also allowed for implementation optimisations such as the Number Theoretic Transform (NTT), as demonstrated by [28], whose software performance of the DLP-IBE outperformed that of the elliptic-curve based Boneh-Franklin IBE scheme.

Bonsai Trees HIBE: Cash [8] proposed the use of Bonsai trees to create a hierarchical structure for IBE. They model the hierarchical network of users as a tree, whereby arborists, or sub-key-managers, have control over the sub-trees, and have the authority to delegate user private keys. Delegation requires the knowledge of a trapdoor basis of the lattice at that level. During the process whereby keys are delegated down the tree, the lattice is extended, and therefore its dimension and hence the key and ciphertext sizes increase. The public key size is of $O(d^3kn^2)$ and ciphertext size is of $O(d^3kn)$ at depth d, for security parameter n and hash output length k. The root authority has control of the whole tree by knowing the short trapdoor basis for the master root lattice. The security of this HIBE scheme is based on LWE over standard lattices.

3 Optimised LATTE HIBE Scheme

LATTE was proposed in 2017 [6] and can be considered as a combination of the DLP-IBE scheme [14] and Bonsai Tree HIBE scheme [8] to create a hierarchical lattice-based IBE scheme. It can be shown to be ID-IND-CCA-secure, the proof for which is given in [1], based on the NTRU and R-LWE hardness assumptions. Table 13 in Appendix A summarises the inputs and outputs of the LATTE algorithms. For the optimised LATTE scheme presented in this Section and used in our software design and implementation, features of the FALCON [34] and the MODFALCON [11] signature schemes were utilised. This is the first time these features have been considered in LATTE and so the rationale for this is expanded on in Section 5. For now, it suffices to acknowledge that the sub-algorithms NTRUSolve, ffSampling and Tree are taken from FALCON and are displayed in Appendix B.

The KeyGen, given in Algorithm 1, generates an NTRU-type basis. This is performed by sampling the short basis polynomials $\mathbf{f}, \mathbf{g}$ from a Gaussian distribution. Operations are over the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle$, a variant of the NTRU ring. For the purposes of implementation optimisation, variables are stored in NTT representation where appropriate. The Gram-Schmidt norm of the associated basis is computed to ensure it is small enough to allow for short private keys to be delegated to the next level. If not, the polynomials are resampled. The rest of the basis, polynomials $\mathbf{F}, \mathbf{G}$, are computed so that they satisfy the NTRU equation, $\mathbf{fG} - \mathbf{gF} = q \mod x^N + 1$. This sub-algorithm is referred to as NTRUSolve, and its implementation will be discussed in Section 5. The solution to this is not unique, but any solution suffices provided it is short enough. This is taken care of by reduction of the coefficients. The public key

Algorithm 1: The LATTE KeyGen algorithm

```
  Input: $N, q, \sigma_0$.
  Output: $\mathbf{S}_0 \in \mathcal{R}_q^{2\times 2}, \mathbf{h}, \mathbf{b} \in \mathcal{R}_q$.
1 $\mathbf{f}, \mathbf{g}, \leftarrow \mathcal{D}_{\sigma_0}^N$.
2 $Norm \leftarrow \max\Big(\|\mathbf{g}, -\mathbf{f}\|, \Big\|\Big(\frac{q\cdot\mathbf{f}^*}{\mathbf{f}\cdot\mathbf{f}^*+\mathbf{g}\cdot\mathbf{g}^*}, \frac{q\cdot\mathbf{g}^*}{\mathbf{f}\cdot\mathbf{f}^*+\mathbf{g}\cdot\mathbf{g}^*}\Big)\Big\|\Big)$.
3 if $Norm > \sigma_0 \cdot \sqrt{2N}$ then go to Step 1;
4 $\mathbf{F}, \mathbf{G} \leftarrow \text{NTRUSolve}_{N,q}(\mathbf{f}, \mathbf{g})$.
5 if NTRUSolve is aborted then go to Step 1;
6 $\mathbf{h} \leftarrow \mathbf{g} \cdot \mathbf{f}^{-1} \mod q$ in NTT domain.
7 $\mathbf{b} \leftarrow_{\$} \mathcal{R}_q$ in NTT domain.
8 return $\mathbf{S}_0 = \left(\begin{smallmatrix} \mathbf{g} & -\mathbf{f} \\ \mathbf{G} & -\mathbf{F} \end{smallmatrix}\right), \mathbf{h}, \mathbf{b}$.
```

consists of polynomial $\mathbf{h} = \mathbf{g} \cdot \mathbf{f}^{-1}$ and polynomial $\mathbf{b}$ sampled from a uniformly random distribution over $\mathcal{R}_q$. The master public basis $\mathbf{B}_0$ and private basis $\mathbf{S}_0$ at level 0 are implicit in the polynomial master keys, as follows:

$$\mathbf{B}_0 = \begin{bmatrix} -\mathcal{A}(\mathbf{h}) & \mathbf{I}_N \\ q\mathbf{I}_N & \mathbf{0}_N \end{bmatrix}, \quad \mathbf{S}_0 = \begin{bmatrix} \mathbf{g} & -\mathbf{f} \\ \mathbf{G} & -\mathbf{F} \end{bmatrix}.$$

The Delegate process, given in Algorithm 2, creates a public/secret key pair for the next level in the tree, allowing it to become a sub key management service (sub-KMS, please refer to Appendix A). Suppose the KMS wishes to delegate a key from level $\ell - 1$ to level ℓ. Then it can extend the public basis of the user at level ℓ by placing $\mathbf{A}_\ell = H(\mathsf{ID}_1|\ldots|\mathsf{ID}_\ell)$, where H is a hash function, to the beginning of the first column and filling the extra row with $\mathbf{I}_N$ and $\mathbf{0}_N$, as shown below. The dimension of the new matrix becomes $(\ell+2)N \times (\ell+2)N$.

The corresponding private basis can then be generated. The i^{th} row $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,\ell+1})$ of the private basis is a short solution to the equation:

$$\mathbf{s}_{i,0} + \mathbf{s}_{i,1} \cdot \mathbf{h} + \mathbf{s}_{i,2} \cdot \mathbf{A}_1 + \cdots + \mathbf{s}_{i,\ell+1} \cdot \mathbf{A}_\ell = \mathbf{0} \mod q.$$

This can be found by sampling short vectors (using the Klein-GPV sampler [18] or its variant from FALCON [34]) from the $(\ell-1)$-level lattice using its secret basis, with centre vector $(-\mathbf{s}_{i,\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \ldots, \mathbf{0})$, where $\mathbf{s}_{i,\ell+1}$ is sampled from a discrete Gaussian distribution $\mathcal{D}_{\sigma_\ell}$ over $\mathcal{R}$. A check is made to ensure the GS-norm of the sampled lattice vector is within the bound $\sigma_\ell \cdot \sqrt{(\ell+2)N}$ to ensure the delegated basis will be of sufficient quality.

The remainder of the Delegate algorithm, in which the bottom row $(\mathbf{s}_{\ell+1,0}, \mathbf{s}_{\ell+1,1}, \ldots, \mathbf{s}_{\ell+1,\ell+1})$ is generated, is a higher-dimensional analogue of LATTE KeyGen. The resulting matrix has a determinant of size q. The final row is then reduced similarly to as in the KeyGen component to ensure the basis is of the required quality for further delegation. Cramer's rule is utilised here to find the reduction coefficients, the details of which are given in Appendix C.

Algorithm 2: The LATTE Delegate algorithm (from level $\ell - 1$ to ℓ)

```
    Input: $N, q, \sigma_\ell, \mathbf{S}_{\ell-1}, H : \{0,1\}^* \to \mathbb{Z}_q^N, \mathsf{ID}_\ell$.
    Output: $\mathbf{S}_\ell \in \mathcal{R}_q^{(\ell+2)\times(\ell+2)}$.
 1  $\mathbf{A}_\ell \leftarrow H(\mathsf{ID}_1|\dots|\mathsf{ID}_\ell)$ in NTT domain.
 2  $T_{\ell-1} \leftarrow \mathrm{Tree}(\mathbf{S}_{\ell-1}, \sigma_\ell)$.
 3  for $i \in \{0,\dots,\ell\}$ do
 4      $\mathbf{s}_{i,\ell+1} \leftarrow \mathcal{D}_{\sigma_\ell}^N$.
 5      $\mathbf{t} \leftarrow (-\mathbf{s}_{i,\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \dots, \mathbf{0}) \cdot \mathbf{S}_{\ell-1}^{-1}$.
 6      $\mathbf{z} \leftarrow \mathrm{FFT}^{-1}(\mathrm{ffSampling}(\mathbf{t}, T_{\ell-1}))$.
 7      $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \dots, \mathbf{s}_{i,\ell}) \leftarrow (\mathbf{t} - \mathbf{z}) \cdot \mathbf{S}_{\ell-1}$.
 8      if $\|(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \dots, \mathbf{s}_{i,\ell}, \mathbf{s}_{i,\ell+1})\| > \sqrt{(\ell+2)N} \cdot \sigma_\ell$ then resample;
 9  Set $\mathbf{M} = (\mathbf{s}_{i,j})$, for $0 \le i \le \ell$ and $1 \le j \le \ell + 1$.
10  if $\mathbf{M}$ is not invertible then go to Step 3;
11  $\mathbf{u} \leftarrow \mathrm{adj}(\mathbf{M}) \cdot (\mathbf{s}_{0,0}, \mathbf{s}_{1,0}, \dots, \mathbf{s}_{\ell,0})^\mathsf{T}$.
12  $(\mathbf{F}_\ell, \mathbf{G}_\ell) \leftarrow \mathrm{NTRUSolve}_{N,q}(\det(\mathbf{M}), \mathbf{u}_0)$ where $\mathbf{u}_0$ is the first coordinate of $\mathbf{u}$.
13  if NTRUSolve is aborted then go to Step 3;
14  $(\mathbf{s}_{\ell+1,0}, \dots, \mathbf{s}_{\ell+1,\ell+1}) \leftarrow (\mathbf{G}_\ell, \mathbf{F}_\ell, \mathbf{0}, \dots, \mathbf{0})$.
15  Set $\mathbf{C} = (\mathbf{c}_{i,j})$, where $\mathbf{c}_{i,j} = \mathbf{s}_{j,0} \cdot \mathbf{s}_{i,0}^* + \dots + \mathbf{s}_{j,\ell+1} \cdot \mathbf{s}_{i,\ell+1}^*$, $0 \le i, j \le \ell$.
16  Let $\mathbf{k} = (\mathbf{k}_i)_{0 \le i \le \ell}$ be the solution to $\mathbf{C} \cdot \mathbf{k} = \mathbf{d}$. By Cramer's rule, $\mathbf{k}_i = \frac{\det(\mathbf{C}_i(\mathbf{d}))}{\det(\mathbf{C})}$,
     where $\mathbf{C}_i(\mathbf{d})$ is the matrix $\mathbf{C}$ with its $i^{th}$ column replaced by
     $\mathbf{d}_i = \mathbf{s}_{\ell+1,0} \cdot \mathbf{s}_{i,0}^* + \dots + \mathbf{s}_{\ell+1,\ell+1} \cdot \mathbf{s}_{i,\ell+1}^*$.
17  for $i \in \{0,\dots,\ell\}$ do
18      $(\mathbf{s}_{\ell+1,0}, \dots, \mathbf{s}_{\ell+1,\ell+1}) = (\mathbf{s}_{\ell+1,0}, \dots, \mathbf{s}_{\ell+1,\ell+1}) - \lfloor \mathbf{k}_i \rceil \cdot (\mathbf{s}_{i,0}, \dots, \mathbf{s}_{i,\ell+1})$.
19  return $\mathbf{S}_\ell = (\mathbf{s}_{i,j})$, for $0 \le i, j \le \ell + 1$.
```

Generalising to level ℓ, the public basis $\mathbf{B}_\ell$ and the private basis $\mathbf{S}_\ell$, respectively become:

$$\mathbf{B}_\ell = \begin{bmatrix} -\mathcal{A}(\mathbf{A}_\ell) & \mathbf{0}_N & \cdots & \mathbf{0}_N & \mathbf{I}_N \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ -\mathcal{A}(\mathbf{h}) & \mathbf{I}_N & \cdots & \mathbf{0}_N & \mathbf{0}_N \\ q\mathbf{I}_N & \mathbf{0}_N & \cdots & \mathbf{0}_N & \mathbf{0}_N \end{bmatrix}, \quad \mathbf{S}_\ell = \begin{bmatrix} \mathbf{s}_{0,0} & \mathbf{s}_{0,1} & \cdots & \mathbf{s}_{0,\ell+1} \\ \mathbf{s}_{1,0} & \mathbf{s}_{1,1} & \cdots & \mathbf{s}_{1,\ell+1} \\ \vdots & \vdots & \iddots & \vdots \\ \mathbf{s}_{\ell+1,0} & \mathbf{s}_{\ell+1,1} & \cdots & \mathbf{s}_{\ell+1,\ell+1} \end{bmatrix}.$$

In the LATTE Extract algorithm (Algorithm 3), the user private key is a short solution $(\mathbf{t}_0, \mathbf{t}_1, \dots, \mathbf{t}_{\ell+1})$ to:

$$\mathbf{t}_0 + \mathbf{t}_1 \cdot \mathbf{h} + \mathbf{t}_2 \cdot \mathbf{A}_1 + \dots + \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell = \mathbf{b} \mod q, \tag{1}$$

where $\mathbf{A}_i = H(\mathsf{ID}_1|\dots|\mathsf{ID}_i)$ for $i = 1, \dots, \ell$. Again, this is found using the Klein-GPV style sampler over the short basis from the previous level. An extended version of traditional R-LWE encryption/decryption [27] is used for ciphering messages as given in Algorithms 4 and 5, respectively. A random *seed* is sampled and used together with a Key Derivation Function (KDF) to one-time-pad the message. The *seed* is encoded[5] and then encrypted using Ring-LWE and

[5]The Encode/Decode are the same as described in [1].

Algorithm 3: The LATTE Extract algorithm (from level $\ell - 1$ to user at level ℓ)

```
   Input: $N, q, \sigma_\ell, \mathbf{S}_{\ell-1}, H : \{0,1\}^* \to \mathbb{Z}_q^N, \mathsf{ID}_\ell$.
   Output: $\mathbf{t}_0, \dots, \mathbf{t}_{\ell+1} \in \mathcal{R}_q$.
 1 $\mathbf{A}_\ell \leftarrow H(\mathsf{ID}_1 | \dots | \mathsf{ID}_\ell)$ in NTT domain.
 2 $\mathbf{t}_{\ell+1} \leftarrow \mathcal{D}_{\sigma_\ell}^N$.
 3 $T_{\ell-1} \leftarrow \text{Tree}(\mathbf{S}_{\ell-1}, \sigma_\ell)$.
 4 $\mathbf{t} \leftarrow (\mathbf{b} - \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell, \mathbf{0}, \dots, \mathbf{0}) \cdot \mathbf{S}_{\ell-1}^{-1}$.
 5 $\mathbf{z} \leftarrow \text{FFT}^{-1}(\text{ffSampling}(\mathbf{t}, T_{\ell-1}))$.
 6 $(\mathbf{t}_0, \mathbf{t}_1, \dots, \mathbf{t}_\ell) \leftarrow (\mathbf{t} - \mathbf{z}) \cdot \mathbf{S}_{\ell-1}$.
 7 return $\mathbf{t}_0, \dots, \mathbf{t}_{\ell+1} \in \mathcal{R}_q$ in NTT domain.
```

Algorithm 4: The LATTE Encrypt algorithm (at level ℓ)

```
   Input: $N, q, \sigma_e, \mathbf{h}, \mathbf{b}, \text{KDF}, \mathsf{ID}_\ell, \mu \in \{0,1\}^{256}$.
   Output: $Z \in \{0,1\}^{256}, \mathbf{C}_1, \dots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathcal{R}_q$.
 1 $seed \leftarrow \{0,1\}^{256}$.
 2 $Z \leftarrow \mu \oplus \text{KDF}(seed)$.
 3 Sample $\mathbf{e}, \mathbf{e}_1, \dots, \mathbf{e}_\ell, \mathbf{e}_h, \mathbf{e}_b$ from a binomial distribution with center 0 and
     standard deviation $\sigma_e$ using the seed $\text{KDF}(seed|Z)$.
 4 for $i \in \{1, ..., \ell\}$ do
 5     $\mathbf{C}_i \leftarrow \mathbf{A}_i \cdot \mathbf{e} + \mathbf{e}_i$ where $\mathbf{A}_i = H(\mathsf{ID}_1 | \dots | \mathsf{ID}_i)$ in NTT domain.
 6 $\mathbf{C}_h \leftarrow \mathbf{h} \cdot \mathbf{e} + \mathbf{e}_h$.
 7 $\mathbf{m} \leftarrow \text{Encode}(seed)$.
 8 $\mathbf{C}_b \leftarrow \mathbf{b} \cdot \mathbf{e} + \mathbf{e}_b + \mathbf{m}$.
 9 return $Z \in \{0,1\}^{256}, \mathbf{C}_1, \dots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathcal{R}_q$ in NTT domain.
```

Algorithm 5: The LATTE Decrypt algorithm (at level ℓ)

```
   Input: $N, q, \sigma_e, \mathbf{h}, \mathbf{b}, \text{KDF}, \mathsf{ID}_\ell, Z, (\mathbf{C}_1, \dots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b), (\mathbf{t}_0, \dots, \mathbf{t}_{\ell+1})$.
   Output: $\mu'$.
 1 $\mathbf{V} \leftarrow \mathbf{C}_b - \mathbf{C}_h \cdot \mathbf{t}_1 - \mathbf{C}_1 \cdot \mathbf{t}_2 - \dots - \mathbf{C}_\ell \cdot \mathbf{t}_{\ell+1}$.
 2 $seed' \leftarrow \text{Decode}(\mathbf{V})$.
 3 Sample $\mathbf{e}', \mathbf{e}'_1, \dots, \mathbf{e}'_\ell, \mathbf{e}'_h, \mathbf{e}'_b$ from a binomial distribution with center 0 and
     standard deviation $\sigma_e$ using the seed $\text{KDF}(seed'|Z)$.
 4 for $i \in \{1, \dots, \ell\}$ do
 5     $\mathbf{C}'_i \leftarrow \mathbf{A}_i \cdot \mathbf{e}' + \mathbf{e}'_i$ where $\mathbf{A}_i = H(\mathsf{ID}_1 | \dots | \mathsf{ID}_i)$ in NTT domain.
 6 $\mathbf{C}'_h \leftarrow \mathbf{h} \cdot \mathbf{e}' + \mathbf{e}'_h$.
 7 $\mathbf{m}' \leftarrow \text{Encode}(seed')$.
 8 $\mathbf{C}'_b \leftarrow \mathbf{b} \cdot \mathbf{e}' + \mathbf{e}'_b + \mathbf{m}'$.
 9 Check $(\mathbf{C}'_1, \dots, \mathbf{C}'_\ell, \mathbf{C}'_h, \mathbf{C}'_b)$ agrees with $(\mathbf{C}_1, \dots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b)$.
10 return $\mu' = Z \oplus \text{KDF}(seed')$.
```

sent. The ciphertext consists of the encrypted message Z, deterministically sampled ephemeral public keys $\mathbf{C}_1, \dots, \mathbf{C}_\ell, \mathbf{C}_h$ and the encrypted *seed*, $\mathbf{C}_b$. This is a variant of the Fujisaki-Okamoto transform [17] to protect against invalid ciphertexts. The Decrypt process takes the user private key to decrypt the *seed* and reconstruct the message. This works as follows by operations over $\mathcal{R}_q$:

$$\begin{aligned}\mathbf{V} &= \mathbf{C}_b - \mathbf{t}_1 \cdot \mathbf{C}_h - \mathbf{t}_2 \cdot \mathbf{C}_1 - \cdots - \mathbf{t}_{\ell+1} \cdot \mathbf{C}_\ell \\ &= (\mathbf{b} \cdot \mathbf{e} + \mathbf{e}_b + \mathbf{m}) - \mathbf{t}_1(\mathbf{h} \cdot \mathbf{e} + \mathbf{e}_h) - \mathbf{t}_2(\mathbf{A}_1 \cdot \mathbf{e} + \mathbf{e}_1) - \cdots - \mathbf{t}_{\ell+1}(\mathbf{A}_\ell \cdot \mathbf{e} + \mathbf{e}_\ell) \\ &= \mathbf{e}_b + \mathbf{m} - \mathbf{t}_1 \cdot \mathbf{e}_h - \mathbf{t}_2 \cdot \mathbf{e}_1 - \cdots - \mathbf{t}_{\ell+1} \cdot \mathbf{e}_\ell + \mathbf{t}_0 \cdot \mathbf{e},\end{aligned}$$

where the first equality is derived by substituting $\mathbf{C}_b, \mathbf{C}_h, \mathbf{C}_i$ with their definitions, and the second equality holds based on (1). By construction, the error and private key terms are small enough so that $\mathbf{m}$ is decoded successfully to recover the *seed*. From the *seed*, the message is straightforwardly recovered from Z, which is sent as part of the ciphertext.

4 Theoretical Analysis

4.1 Key and Ciphertext Sizes

The key and ciphertext sizes for Latte are given in Table 14 in Appendix D. The method taken for calculating these is also given in Appendix D.

4.2 Security Analysis and Parameter Sets

The security of each component of Latte depends on an associated lattice problem and so the computational security of each of these must be considered to derive parameters, with the most vulnerable component determining the overall security for a given parameter set. The global parameters for the scheme are dimension N and modulus q, but we will also need to consider level-specific parameters, namely the standard deviation used for sampling at each level, σ_ℓ. The six security constraints to be considered are:

- Gaussian sampler security
- Decryption failure
- Master key recovery (breaking the NTRU problem/finding short vectors in the NTRU lattice)
- Delegated key recovery (finding short vectors in the lattice)
- User key recovery (solving closest vector problem)
- Message recovery (breaking the R-LWE encryption scheme)

These are discussed in detail in [1]. For the sake of completeness, we give them in the following as well but will only state the mathematical conditions which must be satisfied, and recompute the security levels using our own optimisation code. We first summarise the differences between our security analysis and that of [1]. Any other differences are negligible and due to precision variations in the attack costing script.

Summary of differences compared to [1]: There are two main differences:

- We find that the discrete Gaussian statistical parameter $\epsilon = 2^{-22.5}/(\ell+1)N$ used by σ_ℓ in [1] was miscalculated. The Kullback-Liebler divergence between the sampled distribution and the ideal discrete Gaussian distribution is bounded by approximately $8((\ell+1)N)^2\epsilon^2$. Choosing $\epsilon = 2^{-25.5}/(\ell+1)N$ ensures the divergence is at most 2^{-48}, as specified by the proposed LATTE specification [1]. If the sampled distribution has a KL-divergence of 2^{-48} from the ideal distribution then using the sampler at most 2^{47} times will only reduce the security of the scheme by up to one bit [32]. However, in [1], the $\epsilon = 2^{-22.5}/(\ell+1)N$ would only ensure the KL-divergence is at most 2^{-42}.
- To accommodate the use of the FACCT sampler in KeyGen, as described in Section 5, we modify the value of σ_0. This also has an effect on the subsequent σ_ℓ, and therefore difficulty of the underlying lattice problems and success of each attack.

Gaussian Sampler Security: The statistical security of the Gaussian sampler used for sampling short vectors from lattice cosets in extraction and delegation to level ℓ is determined by the standard deviation of the sampler σ_ℓ and its relation to the Gram-Schmidt norm of the input basis. As this property of the basis is determined from the master key generation and any previous delegations, i.e. $||\mathbf{B}||_{GS} \leq \sqrt{(\ell+2)N}\cdot\sigma_\ell$, we can draw the following condition based on the relationship of the standard deviations at each level:[6]

$$\sigma_{\ell+1} \geq \eta_\epsilon(\mathbb{Z})\sqrt{(\ell+2)N}\cdot\sigma_\ell,$$

taking ϵ as $2^{-25.5}/(\ell+2)N$ in order to make the Kullback-Leibler divergence of the sampler from the discrete Gaussian is at most 2^{-48}. However, we also require the sampled vectors to be short for the purposes of keeping the underlying lattice problem hard. Therefore, we can set:

$$\sigma_{\ell+1} = \eta_\epsilon(\mathbb{Z})\sqrt{(\ell+2)N}\cdot\sigma_\ell,$$

where $\sigma_0 \approx 1.17\sqrt{\frac{q}{2N}}$. σ_0 is chosen to be this as it minimises the Gram-Schmidt norm of the master basis (resulting in short user private keys in the single-level IBE), as deduced in [14]. Table 1 gives the σ_ℓ used for each parameter set, which can be computed indirectly from (N, q).

Decryption Failure: To protect against attacks which exploit random decryption failures, we must bound the error term incurred in the R-LWE encryption scheme. The probably that the error term is too large is derived in [1], based on the method of [2]. Essentially, the decryption failure rate cannot exceed $2^{-\lambda}$, where λ is the security level in bits of the scheme. For each parameter set and level, we can compute the probability of decryption failure. This is given in Table 2.

[6]We take the smoothing parameter $\eta_\epsilon(\mathbb{Z})$ to be defined as $\eta_\epsilon(\mathbb{Z}) = \frac{1}{\pi}\sqrt{\frac{\ln(2+2/\epsilon)}{2}}$.

Table 1: LATTE σ_ℓ

Set	σ_ℓ		
	$\ell = 0$	$\ell = 1$	$\ell = 2$
LATTE-0	3388.8	85613.5	2670753.4
LATTE-1	106.2	5513.3	-
LATTE-2	106.2	7900.2	-
LATTE-3	6777.6	351968.4	22559988.0
LATTE-4	9583.7	713170.8	65489528.1

Table 2: LATTE Decryption Failure Probabilities

Set	**Failure Probability**	
	$\ell = 1$	$\ell = 2$
LATTE-0	$2^{-\inf}$	2^{-19}
LATTE-1	2^{-124}	-
LATTE-2	2^{-249}	-
LATTE-3	$2^{-\inf}$	2^{-90}
LATTE-4	$2^{-\inf}$	2^{-178}

Table 3: LATTE Estimated Cost of Master Key Recovery

Set	β	**Classical Security**	**Quantum Security**
LATTE-0	236	85	79
LATTE-1	974	301	275
LATTE-2	1501	455	414
LATTE-3	973	301	274
LATTE-4	1501	455	414

Master Key Recovery: The security of the master key recovery depends upon the difficulty of finding the short vector $(\mathbf{g}, \mathbf{f})$ in the lattice, given the public NTRU basis. The attack is successful if the projection of the short vector onto the vector space spanned by the final β Gram-Schmidt vectors is shorter than the length of the $(2N - \beta + 1)^{th}$ Gram-Schmidt vector. This corresponds to minimising block size β, for:

$$\sigma_0 \sqrt{\beta} \leq GH(\beta)^{(2\beta - 2N)/(\beta - 1)} \cdot \det(\Lambda_0)^{1/2N}.$$

The minimum solutions to this inequality for different parameter sets is given in Table 3. The work required to find the shortest vector using this block size with the BKZ2.0 algorithm is also given.

Delegated Key Recovery: For delegated key recovery, the attacker must find a short sequence of vectors in $\Lambda_{\ell-1}$. This can reduce to solving SVP in the master lattice Λ_0 to find a vector of length $\sigma_\ell \cdot \sqrt{2N}$. Table 4 gives the minimum block size β required (as per below (2)) for a successful attack using BKZ2.0 and the classical and quantum cost of these attacks which depend on N and q.

$$\sigma_\ell \cdot \sqrt{2N} \leq GH(\beta)^{(2N)/(\beta - 1)} \cdot \det(\Lambda_0)^{1/2N}. \tag{2}$$

User Key Recovery: User key recovery requires finding a short solution to:

$$\mathbf{t}_0 + \mathbf{t}_1 \cdot \mathbf{h} + \mathbf{t}_2 \cdot \mathbf{A}_1 + \cdots + \mathbf{t}_{\ell+1} \cdot \mathbf{A}_\ell = \mathbf{b},$$

Table 4: LATTE Estimated Cost of Delegated Key Recovery

Set	ℓ	β	Classical Security	Quantum Security
LATTE-0	1	179	69	64
	2	1	17	17
LATTE-1	1	1020	314	287
LATTE-2	1	1051	323	295
LATTE-3	1	1021	315	287
	2	388	130	119
LATTE-4	1	1051	323	295
	2	907	281	257

Table 5: LATTE Estimated Cost of User Key Recovery

Set	ℓ	β	Classical Security	Quantum Security
LATTE-0	1	116	50	47
	2	1	17	17
LATTE-1	1	784	245	224
LATTE-2	1	1051	323	295
LATTE-3	1	785	246	224
	2	326	112	103
LATTE-4	1	1051	323	295
	2	783	245	224

Table 6: Cost of Primal Message Recovery Attack

Set	m	β	Classical Security	Quantum Security
LATTE-0	86	50	31	30
LATTE-1	1018	423	140	128
LATTE-2	1962	967	299	273
LATTE-3	998	232	84	78
LATTE-4	2037	561	180	165

Table 7: Cost of Dual Message Recovery Attack

Set	m	β	Classical Security	Quantum Security
LATTE-0	88	50	31	30
LATTE-1	1039	422	140	128
LATTE-2	1974	964	298	272
LATTE-3	1005	232	84	78
LATTE-4	2101	560	180	165

which reduces to solving the CVP in the master lattice Λ_0, with the vector having length $\sigma_\ell \cdot \sqrt{2(\ell+2)} \cdot \sqrt{2N}$. To do this, it is required to minimise (3) (see below) over β. Table 5 gives the minimum block size β required for a successful attack, and the classical and quantum cost of these attacks. This depends only on N.

$$\sigma_\ell \cdot \sqrt{2(\ell+2)} \cdot \sqrt{2N} \leq GH(\beta)^{(2N)/(\beta-1)} \cdot \det(\Lambda_0)^{1/2N}. \tag{3}$$

Message Recovery: Message recovery requires breaking the underlying R-LWE encryption scheme. There are two attacking methods to consider for this. Message recovery depends on solving an extended version of R-LWE, which reduces to an instance of the primal-CVP or dual-SVP. In the primal-CVP attack, the ephemeral private keys are recovered via a close vector problem. In the dual-SVP attack an attempt is made to distinguish the ciphertext elements from uniformly random polynomials in $\mathcal{R}_q$.

The minimal block size β needed for a successful attack, and the cost of these attacks are given in Tables 6 and 7, depending on (N, q). The code to populate Tables 6 and 7 is that used in [2]. By considering the cost of all attacks covered in this section, the security levels in Table 8 could be derived.

Table 8: Latte Parameters

Set	Security	N	q
LATTE-0	16	256	$2^{32} - 2^{20} + 1$
LATTE-1	128	1024	$2^{24} - 2^{14} + 1$
LATTE-2	256	2048	$2^{25} - 2^{12} + 1$
LATTE-3	80	1024	$2^{36} - 2^{20} + 1$
LATTE-4	160	2048	$2^{38} - 2^{26} + 1$

Setting up Parameters: The parameter sets are given in Table 8. These are the parameters recommended in the original specification [1]. We have extended the security estimates from [1] to give them on a per-level basis. The security decreases as we move down the hierarchy. However, it turns out that each parameter set's security is determined by the message recovery capabilities, which remain constant down the levels. Therefore our parameter security conclusions match that of [1].

Parameter sets Latte-0, 1, and 2 are only applicable to a single level, essentially an IBE rather than HIBE, version of the scheme. Latte-3 and 4 can be used for up to two levels. The reason we cannot use these parameters beyond these levels is that the decryption failure rate exceeds the target security level. In fact, the failure rate is so high it renders the scheme completely insecure and also not suitable for use.

5 Software Design Features and Considerations

5.1 Techniques from Falcon and ModFalcon

The design of Latte presented in this paper utilises techniques from the signature scheme Falcon [34]. The two schemes are closely related; they are instantiated over the same type of lattice and share key generation and sampling procedures. Falcon makes use of the "tower of rings" structure to find a solution to the NTRU equation $\mathbf{fG} - \mathbf{gF} = q \mod x^N + 1$, for a given $\mathbf{f}$ and $\mathbf{g}$ in the NTRUSolve sub-algorithm of KeyGen, and in the lattice Gaussian sampling (ffSampling) component of Latte Delegate and Extract. The tower of rings approach utilises the fact that computations over polynomials $\mathbf{f}, \mathbf{g} \in \mathbb{C}[x]/\langle x^{N/2}+1\rangle$ are equivalent to computations over $\mathbf{f}(x^2), \mathbf{g}(x^2) \in \mathbb{C}[x]/\langle x^N + 1\rangle$. When $N = 2^k$, for some $k \in \mathbb{Z}$, this can be applied repeatedly so that computations are performed over polynomials of degree 1. This brings advantages in terms of both memory usage and speed [33].

In addition, in Latte Delegate, to complete the delegated basis $\mathbf{S}_\ell$ for lattice dimension higher than $2N$, we adapt the technique from ModFalcon [11]. Let $\mathbf{S}_\ell = \begin{pmatrix} \mathbf{v}^\top & \mathbf{M} \\ \mathbf{G}_\ell & \mathbf{F}'_\ell \end{pmatrix}$ be the delegated basis, where $\mathbf{G}_\ell = \mathbf{s}_{\ell+1,0}$, $\mathbf{F}'_\ell = (\mathbf{s}_{\ell+1,1}, \dots, \mathbf{s}_{\ell+1,\ell+1})$,

$\mathbf{v} = (\mathbf{s}_{0,0}, \mathbf{s}_{1,0}, \ldots, \mathbf{s}_{\ell,0})$, and $\mathbf{M} = (\mathbf{s}_{i,j})$ for $0 \le i \le \ell$ and $1 \le j \le \ell + 1$. By Schur complement, if $\mathbf{M}$ is invertible, we have:

$$\begin{aligned}\det(\mathbf{S}_\ell) &= \det(\mathbf{M}) \cdot \det(\mathbf{G}_\ell - \mathbf{F}'_\ell \cdot \mathbf{M}^{-1} \cdot \mathbf{v}) = \det(\mathbf{M}) \cdot (\mathbf{G}_\ell - \mathbf{F}'_\ell \cdot \mathbf{M}^{-1} \cdot \mathbf{v}) \\ &= \det(\mathbf{M}) \cdot \mathbf{G}_\ell - \mathbf{F}'_\ell \cdot \operatorname{adj}(\mathbf{M}) \cdot \mathbf{v}.\end{aligned}$$

Let $\mathbf{F}'_\ell = (\mathbf{F}_\ell, \mathbf{0}, \ldots, \mathbf{0})$. We have $\det(\mathbf{S}_\ell) = \det(\mathbf{M}) \cdot \mathbf{G}_\ell - \mathbf{F}_\ell \cdot \mathbf{u}_0$ where $\mathbf{u}_0$ is the first coordinate of $\mathbf{u} = \operatorname{adj}(\mathbf{M}) \cdot \mathbf{v}$. In order to fill the bottom row $(\mathbf{s}_{\ell+1,0}, \ldots, \mathbf{s}_{\ell+1,\ell+1})$ of $\mathbf{S}_\ell$, if $\mathbf{M}$ is invertible, we can use the same NTRUSolve algorithm as in LATTE KeyGen to find $\mathbf{F}_\ell, \mathbf{G}_\ell$ such that $\det(\mathbf{M}) \cdot \mathbf{G}_\ell - \mathbf{F}_\ell \cdot \mathbf{u}_0 = q$, and we simply resample when $\det(\mathbf{M}) = 0$. However, the coefficient size of the output $\mathbf{F}_\ell, \mathbf{G}_\ell$ will be approximately the coefficient size of the input $\det(\mathbf{M}), \mathbf{u}_0$, which is in the order of $q^{\ell+1}$. To make the infinity norm of $\mathbf{S}_\ell$ less than q, we need further length reduction by using Cramer's rule (see Appendix C).

5.2 Discrete Gaussian Sampling over the Integers

In LATTE KeyGen, $\mathbf{f}, \mathbf{g}$ may need to be resampled multiple times due to the norm check and possible failure to find solutions of the NTRU equation. In order to sample $2N$ coordinates efficiently from $\mathcal{D}_{\sigma_0}$, we employ the FACCT sampler [37], which is fast and compact even for larger σ_0 used in LATTE-3 and 4. However, since the FACCT sampler can only sample with $\sigma = k\sqrt{1/(2\ln 2)}$ where k is a positive integer, we slightly increase $\sigma_0 \approx 1.17\sqrt{q/(2N)}$ in LATTE parameters by setting $k = \left\lceil 1.17\sqrt{q/(2N)}/\sqrt{1/(2\ln 2)} \right\rceil$.

Let $\mathbf{S}_\ell = \mathbf{L} \cdot \tilde{\mathbf{S}}_\ell$ be the GSO decomposition of the delegated basis $\mathbf{S}_\ell \in \mathcal{R}^{(\ell+2)\times(\ell+2)}$, where $\mathbf{L}$ is unit lower triangular and rows $\tilde{\mathbf{s}}_i$ of $\tilde{\mathbf{S}}_\ell$ are pairwise orthogonal. We find that the Euclidean norm of the last GSO vector $\tilde{\mathbf{s}}_{\ell+1}$ is very small compared to $\tilde{\mathbf{s}}_0, \ldots, \tilde{\mathbf{s}}_\ell$. This is because rows $\mathbf{s}_0, \ldots, \mathbf{s}_\ell$ of $\mathbf{S}_\ell$ are sampled with a large σ_ℓ but $\det(\mathbf{S}_\ell \cdot \mathbf{S}^*_\ell) = \prod_{i=0}^{\ell+1} \langle \tilde{\mathbf{s}}_i, \tilde{\mathbf{s}}_i \rangle$ is constant and always equal to q^2 [11]. The experiment results in Fig.3 of [10] also verified that $\|\tilde{\mathbf{s}}_{\ell+1}\|$ decreases significantly by increasing $\|\mathbf{s}_0\|$ for $\mathbf{S}_\ell \in \mathcal{R}^{3\times3}$. In this case, the ratio between the maximal and minimal standard deviation σ' used by the integer discrete Gaussian sampling subroutine in ffSampling is very large and the isochronous sampler [23] used by FALCON [34] will be inefficient for our scheme, since the rejection rate of [23] is proportional to $(\max(\sigma')/\min(\sigma'))$. In order to sample with σ' in a broad range, we employ the COSAC sampler [36] instead, which is scalable to large σ' without sacrificing the efficiency.

To accelerate the LATTE Encrypt and Decrypt speed, we sample the ephemeral keys $\mathbf{e}, \mathbf{e}_1, \ldots, \mathbf{e}_\ell, \mathbf{e}_h, \mathbf{e}_b$ from a binomial distribution with center 0 and small standard deviation $\sigma_e = 2.0$ instead of $\mathcal{D}_{\sigma_e}$ used by [1]. Sampling from a binomial distribution is much faster than sampling from $\mathcal{D}_{\sigma_e}$ and the impact on security is negligible in the encryption [2].

Table 9: Proof of Concept Latte Performance Results (op/s) from [1] (Scaled to 4.2GHz)

	$\ell = 1$		$\ell = 2$	
Set	**Enc**	**Dec**	**Enc**	**Dec**
LATTE-1	2911	2987	-	-
LATTE-2	1335	1351	-	-
LATTE-3	1892	1774	1455	1474
LATTE-4	709	668	568	541

Table 10: Our Optimised Latte Performance Results (op/s) at 4.2GHz

		$\ell = 1$			$\ell = 2$			
Set	**KeyGen**	**Ext**	**Enc**	**Dec**	**Del**	**Ext**	**Enc**	**Dec**
LATTE-1	4.7	5.2	12868.5	11163.8	-	-	-	-
LATTE-2	1.5	1.9	6157.9	5287.7	-	-	-	-
LATTE-3	3.7	5.2	6231.7	5207.4	1.0	3.4	5162.4	4360.4
LATTE-4	1.0	1.9	3078.8	2573.3	0.3	1.2	2558.3	2158.2

6 Performance Results

The first published specification of Latte is [1]. However, it only provided the Encrypt and Decrypt performance results, as displayed in Table 9, scaled and converted into op/s at 4.2GHz. Here, we give the first full performance results for our optimised variant of Latte, including KeyGen, Extract, and Delegate.

To avoid confusion for the reader, Table 12 in Appendix A indicates the HIBE functions that can be performed at each hierarchical level. We consider a user at level ℓ can extract the user's private key by using the master key at level $\ell - 1$. For example, a single-level IBE (Latte-0 and 1) is the same as considering only level 1 without the delegation. Delegation is from level $\ell - 1$ to level ℓ. A user at level $\ell + 1$ can then extract the user's private key by using this delegated key, and this user's key pair can be used to encrypt and decrypt at level $\ell + 1$. Table 12 also indicates the dimension of the keys at each level.

We employ the gmp [20], mpfr [16], and mpc [15] libraries for multiprecision integer, floating-point, and complex number arithmetic, respectively. The precision of floating-point and complex numbers in our implementation is $\lambda = 256$ bits. In addition, we use the AES-256 CTR mode with hardware AES-NI instructions as the pseudorandom generator, and use SHAKE-256 [30] as the KDF in Latte Encrypt and Decrypt. The performance results have been obtained from a desktop machine with an Intel i7-7700K CPU at 4.2GHz, with both hyper-threading and TurboBoost disabled. We use the gcc 10.2.0 compiler with compiling options `-O3 -march=native` enabled. Results are given in Table 10.

Table 11: Performance Results (op/s) for the DLP IBE Scheme from [1] (Scaled to 4.2GHz)

Set	**Security**	n	$\log_2 q$	**KeyGen**	**Ext**	**Enc**	**Dec**
DLP-0	80	512	22	14.7	873.2	8731.8	6202.9
DLP-3	192	1024	22	4.9	454.1	2639.8	1621.6

As expected, the KeyGen, Extract, and Delegate processes are the most time consuming components of the scheme, and this increases as security and therefore lattice dimension increase. The trend down the hierarchical levels is that the Extract, Encrypt, and Decrypt all become more time consuming as hierarchical level increases. For Extract in Latte-3 and 4, this corresponds to about 35% decrease in op/s from level 1 to level 2. On the other hand, for the Encrypt and Decrypt, our implementation is 2.9x–4.6x faster compared to the previous performance results from [1]. The speedup might be due to: (1) We change the distribution of the ephemeral keys from discrete Gaussian distribution to binomial distribution. (2) We only perform NTT for the ephemeral keys and $\mathbf{m}$ during the Encrypt and Decrypt, since other inputs are already in the NTT domain. In addition, our optimised Latte Delegate only takes about 1–3 seconds on a desktop machine at 4.2GHz, which is practical and much faster than the estimated run-time (in the order of minutes) for the Delegate in the un-optimised variants of Latte [1].

6.1 Comparison to DLP IBE

Performance results of the single-level DLP IBE scheme from [1] (converted to op/s at 4.2GHz) are given in Table 11. Since the decryption in the DLP IBE did not include ciphertext validation, for a fair comparison with Latte, we use the sum of DLP encryption and decryption run-time to compute the op/s of decryption in Table 11. We focus on the comparison between Latte-1 and DLP-3, since the sizes of parameters N and q are similar. The KeyGen speed of our Latte-1 implementation is similar to DLP-3, and the Encrypt/Decrypt speed is 4.9x–6.9x faster in our implementation. However, the speed of Extract in our implementation is very slow. For example, our Latte-1 Extract implementation is about 87x slower than DLP-3 extraction. This is mainly because we use 256-bit multiprecision in the key sub-algorithms (Tree and ffSampling) and in the COSAC sampler used by our implementation, while it is sufficient to use less than 64-bit precision in DLP extraction [28].

6.2 Limitations

Our current implementation is not constant-time, since the arithmetic in KeyGen, Delegate, and Extract relies on multiprecision libraries [15,16,20]. In addition, the multiprecision arithmetic is also the bottleneck in the Latte Extract, and we find that the speeds of LATTE KeyGen, Delegate, and Extract are linearly proportional to the precision in our experiment. To overcome both limitations, a concrete precision requirement analysis is planned as another future work to optimise the implementation.

References

1. Quantum-Safe Identity-based Encryption. Technical report, The European Telecommunications Standards Institute, Sophia-Antipolis, France (2019)
2. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: USENIX Security Symposium. pp. 327–343. USENIX Association (2016)
3. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 3027, pp. 223–238. Springer (2004)
4. Boneh, D., Boyen, X., Goh, E.: Hierarchical identity based encryption with constant size ciphertext. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 3494, pp. 440–456. Springer (2005)
5. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: CRYPTO. Lecture Notes in Computer Science, vol. 2139, pp. 213–229. Springer (2001)
6. Campbell, P., Groves, M.: Practical post-quantum hierarchical identity-based encryption (2017), available at `https://www.qub.ac.uk/csit/FileStore/Filetoupload,785752,en.pdf`
7. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 2656, pp. 255–271. Springer (2003)
8. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 6110, pp. 523–552. Springer (2010)
9. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 7073, pp. 1–20. Springer (2011)
10. Cheon, J.H., Kim, D., Kim, T., Son, Y.: A new trapdoor over Module-NTRU lattice and its application to ID-based encryption. IACR Cryptol. ePrint Arch. **2019**, 1468 (2019)
11. Chuengsatiansup, C., Prest, T., Stehlé, D., Wallet, A., Xagawa, K.: ModFalcon: Compact signatures based on Module-NTRU lattices. In: AsiaCCS. pp. 853–866. ACM (2020)
12. Cramer, Gabriel: Introduction a l'analyse des lignes courbes algebriques par Gabriel Cramer. chez les freres Cramer & Cl. Philibert (1750)
13. Dodis, Y., Fazio, N.: Public key broadcast encryption for stateless receivers. In: Digital Rights Management Workshop. Lecture Notes in Computer Science, vol. 2696, pp. 61–80. Springer (2002)
14. Ducas, L., Lyubashevsky, V., Prest, T.: Efficient identity-based encryption over NTRU lattices. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 8874, pp. 22–41. Springer (2014)
15. Enge, A., Gastineau, M., Théveny, P., Zimmermann, P.: mpc — A library for multiprecision complex arithmetic with exact rounding. INRIA, 1.1.0 edn. (Jan 2018), `http://mpc.multiprecision.org/`

16. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw. **33**(2), 13 (2007)
17. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: CRYPTO. Lecture Notes in Computer Science, vol. 1666, pp. 537–554. Springer (1999)
18. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC. pp. 197–206. ACM (2008)
19. Gentry, C., Silverberg, A.: Hierarchical ID-based cryptography. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 2501, pp. 548–566. Springer (2002)
20. Granlund, T., Team, G.D.: GNU MP 6.0 Multiple Precision Arithmetic Library. Samurai Media Limited, London, GBR (2015)
21. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: ANTS. Lecture Notes in Computer Science, vol. 1423, pp. 267–288. Springer (1998)
22. Horwitz, J., Lynn, B.: Toward hierarchical identity-based encryption. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 2332, pp. 466–481. Springer (2002)
23. Howe, J., Prest, T., Ricosset, T., Rossi, M.: Isochronous Gaussian sampling: From inception to implementation. In: PQCrypto. Lecture Notes in Computer Science, vol. 12100, pp. 53–71. Springer (2020)
24. Koshiba, T., Takashima, K.: New assumptions on isogenous pairing groups with applications to attribute-based encryption. In: ICISC. Lecture Notes in Computer Science, vol. 11396, pp. 3–19. Springer (2018)
25. Lenstra, A., Lenstra, H., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. **261**, 515–534 (1982)
26. Luzhnica, G.: Pairing based cryptography and implementation in Java. Master Thesis (2011)
27. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer (2010)
28. McCarthy, S., Smyth, N., O'Sullivan, E.: A practical implementation of identity-based encryption over NTRU lattices. In: IMACC. Lecture Notes in Computer Science, vol. 10655, pp. 227–246. Springer (2017)
29. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In: CRYPTO. Lecture Notes in Computer Science, vol. 2139, pp. 41–62. Springer (2001)
30. NIST: SHA-3 standard: Permutation-based hash and extendable-output functions. `https://doi.org/10.6028/NIST.FIPS.202` (2015)
31. NIST: Post-quantum crypto project. `https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization` (2016)
32. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: CHES. Lecture Notes in Computer Science, vol. 8731, pp. 353–370. Springer (2014)
33. Pornin, T., Prest, T.: More efficient algorithms for the NTRU key generation using the field norm. In: Public Key Cryptography (2). Lecture Notes in Computer Science, vol. 11443, pp. 504–533. Springer (2019)
34. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-Fourier lattice-

based compact signatures over NTRU. Tech. rep., National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`
35. Schnorr, C., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Math. Program. **66**, 181–199 (1994)
36. Zhao, R.K., Steinfeld, R., Sakzad, A.: COSAC: compact and scalable arbitrary-centered discrete Gaussian sampling over integers. In: PQCrypto. Lecture Notes in Computer Science, vol. 12100, pp. 284–303. Springer (2020)
37. Zhao, R.K., Steinfeld, R., Sakzad, A.: FACCT: fast, compact, and constant-time discrete Gaussian sampler over integers. IEEE Trans. Computers **69**(1), 126–137 (2020)

A HIBE Scheme Description

Figure 1 shows the components of an HIBE scheme and how they interact. The steps are as follows:

1. **KeyGen:** The master key generator establishes the master public and private keys.
2. **Delegate:** Through a delegation function, the master key generator creates a public/private key pair for the sub key manager. This gives it the ability to delegate further key pairs, and extract user private keys at that level.
3. **Delegate:** The sub key manager delegates a further public/private key to the next level of the hierarchy.
4. **Extract:** The extractor uses their public/private key pair to extract and share user public/private keys, as in the single-level IBE scheme.
5. **Encrypt/Decrypt:** Encryption/decryption works as a regular encryption scheme, such as R-LWE encryption.

An HIBE scheme is said to be IND-CCA-secure if it is indistinguishable under chosen ciphertext attacks; that is, an adversary with the ability to decrypt any other ciphertext does not possess an advantage in decrypting the challenge ciphertext. ID-IND-CCA further implies the adversary has access to an extraction oracle which allows them to extract keys for other identities before committing to the challenge identity, yet gain no advantage. The challenge consists of the ciphertext *and* the identity under which it is encrypted.

Table 12 indicates the notational practises used to identity each level of the hierarchy.

Advantages and additional capabilities of HIBE over IBE include:

- HIBE limits damage performed and access gained by compromise of a key management service (KMS), as an attacker would only gain control over those users controlled by the sub-KMS, rather than all the users in the system.
- HIBE creates finer-grained control over key management within an organisation as responsibility of distributing keys can be allocated to groups further down the hierarchy. Similarly, for public-safety communications, the headquarters could retain control of the central KMS, whilst allowing local provisioning of users by sub-KMSs situated in regional emergency stations.

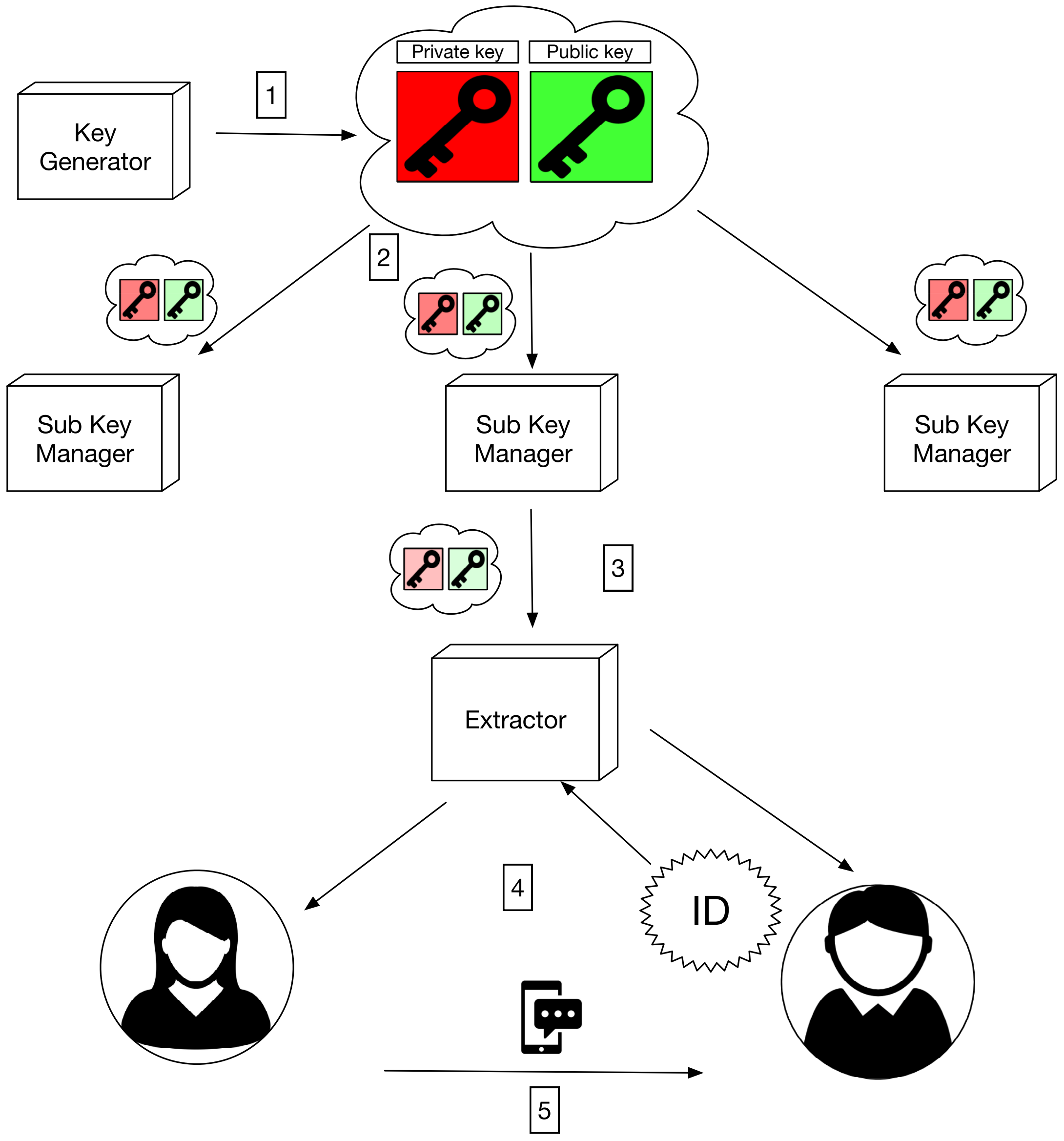

Fig. 1: A 2-level HIBE scheme

Table 12: Explanation of Notational Practice of HIBE Functions

Level	**Function**
Level 0	Master KeyGen $2N \times 2N$
Level 1	Extracting with $2N \times 2N \rightarrow$ Enc/Dec
	Delegating to $3N \times 3N$
Level 2	Extracting with $3N \times 3N \rightarrow$ Enc/Dec

Table 13: LATTE Algorithm Summary

Algorithm	**Inputs**	**Outputs**
KeyGen	$N, q \in \mathbb{Z}, \sigma_0 \in \mathbb{R}$	$\mathbf{S}_0 \in \mathcal{R}_q^{2\times 2}, \mathbf{h}, \mathbf{b} \in \mathcal{R}_q$
Delegate	$\sigma_\ell \in \mathbb{R}, \mathbf{S}_{\ell-1} \in \mathcal{R}_q^{(\ell+1)\times(\ell+1)}, \mathsf{ID}_\ell$ $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^N$	$\mathbf{S}_\ell \in \mathcal{R}_q^{(\ell+2)\times(\ell+2)}$
Extract	$\sigma_\ell \in \mathbb{R}, \mathbf{S}_{\ell-1} \in \mathcal{R}_q^{(\ell+1)\times(\ell+1)}, \mathsf{ID}_\ell$ $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^N$	$\mathbf{t}_0, \ldots, \mathbf{t}_{\ell+1} \in \mathcal{R}_q$
Encrypt	$\sigma_e \in \mathbb{R}, \mathbf{h}, \mathbf{b} \in \mathcal{R}_q$, KDF, $\mathsf{ID}_\ell, \mu \in \{0,1\}^{256}$, $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^N$	$Z \in \{0,1\}^{256}$, $\mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathcal{R}_q$
Decrypt	$Z \in \{0,1\}^{256}$, $\mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathcal{R}_q$, $\mathbf{t}_0, \ldots, \mathbf{t}_{\ell+1} \in \mathcal{R}_q$	$\mu' \in \{0,1\}^{256}$

- HIBE reduces the workload of the master key generator.
- HIBE can be extended into more advanced primitives such as generating short-lived keys for portable computing devices [22] and turning the NNL broadcast encryption system [29] into a public-key broadcast system [13].

B Sub-Algorithms from FALCON

This section presents sub-algorithms from FALCON [34]. Readers may refer to the FALCON specification [34] for subroutines (ffLDL*, splitfft, mergefft, etc.) used by these algorithms.

C Cramer's Rule

Cramer's rule [12] is used for solving systems of linear equations. Considering a system of N equations with N unknowns $\mathbf{x}$, represented as $\mathbf{A}\mathbf{x} = \mathbf{b}$. Cramer's rule states that the solution can be written as $\mathbf{x}_i = \frac{\det(\mathbf{A}_i)}{\det(\mathbf{A})}$, where $\mathbf{A}_i$ is the matrix formed by replacing the i-th column of $\mathbf{A}$ by the column vector $\mathbf{b}$.

Algorithm 6: $\mathrm{NTRUSolve}_{N,q}$ ([34])

```
Input: $\mathbf{f}, \mathbf{g} \in \mathbb{Z}[x] \big/ \langle x^N + 1 \rangle$.
Output: $\mathbf{F}, \mathbf{G} \in \mathbb{Z}[x] \big/ \langle x^N + 1 \rangle$ such that $\mathbf{fG} - \mathbf{gF} = q \mod x^N + 1$.
1  if $N = 1$ then
2      Compute $u, v \in \mathbb{Z}$ such that $u\mathbf{f} - v\mathbf{g} = \gcd(\mathbf{f}, \mathbf{g})$.
3      if $\gcd(\mathbf{f}, \mathbf{g}) \neq 1$ then abort;
4      $(\mathbf{F}, \mathbf{G}) \leftarrow (vq, uq)$.
5      return $(\mathbf{F}, \mathbf{G})$.
6  else
7      $\mathbf{f}' \leftarrow N(\mathbf{f})$.
8      $\mathbf{g}' \leftarrow N(\mathbf{g})$.
9      $(\mathbf{F}', \mathbf{G}') \leftarrow \mathrm{NTRUSolve}_{N/2,q}(\mathbf{f}', \mathbf{g}')$.
10     $\mathbf{F} \leftarrow \mathbf{F}'(x^2) \cdot \mathbf{f}'(x^2) / \mathbf{f}(x)$.
11     $\mathbf{G} \leftarrow \mathbf{G}'(x^2) \cdot \mathbf{g}'(x^2) / \mathbf{g}(x)$.
12     $\mathbf{k} \leftarrow \left\lfloor \frac{\mathbf{F} \cdot \mathbf{f}^* + \mathbf{G} \cdot \mathbf{g}^*}{\mathbf{f} \cdot \mathbf{f}^* + \mathbf{g} \cdot \mathbf{g}^*} \right\rceil \in \mathcal{R}$.
13     $\mathbf{F} \leftarrow \mathbf{F} - \mathbf{k} \cdot \mathbf{f}$ and $\mathbf{G} \leftarrow \mathbf{G} - \mathbf{k} \cdot \mathbf{g}$.
14     return $(\mathbf{F}, \mathbf{G})$.
```

Algorithm 7: The ffSampling Tree computation algorithm ([34])

```
Input: $\mathbf{S}_\ell, \sigma_\ell$.
Output: Tree $T_\ell$.
1  $\mathbf{G}_\ell \leftarrow \mathbf{S}_\ell \cdot \mathbf{S}_\ell^*$.
2  $T \leftarrow \mathrm{ffLDL}^*(\mathrm{FFT}(\mathbf{G}_\ell))$.
3  For each leaf of $T_\ell$, leaf.value $\leftarrow \sigma_\ell / \sqrt{\text{leaf.value}}$.
4  return $T_\ell$.
```

The formulae for the reduction coefficients in the KeyGen and Delegate process come directly from Cramer's Rule applied to the system $\mathbf{Ax} = \mathbf{b}$, where, in the first level, $\mathbf{A}$ is the 2×2 matrix whose (i,j)-entry is the Hermitian product $\langle \mathbf{s}_i, \mathbf{s}_j \rangle$ of the i^{th} and j^{th} rows of the delegation matrix, and where $\mathbf{b}$ is the two-dimensional column vector whose i^{th} coefficient is $\langle \mathbf{s}_2, \mathbf{s}_i \rangle$. This result generalises to arbitrary levels; i.e., for any given number of levels $\ell \geq 1$, the reduction of the vector $\mathbf{s}_{\ell+1}$ is effected by replacing it with $\mathbf{s}_{\ell+1} - \lfloor \mathbf{k}_0 \rceil \mathbf{s}_0 - \cdots - \lfloor \mathbf{k}_\ell \rceil \mathbf{s}_\ell$, where the $\mathbf{k}_i$ are the coefficients of the solution $\mathbf{x}$ to the system $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is the $(\ell+1) \times (\ell+1)$ matrix whose (i,j)-entry is the Hermitian product $\langle \mathbf{s}_i, \mathbf{s}_j \rangle$ of the i^{th} and j^{th} rows of the delegation matrix, and where $\mathbf{b}$ is the $(\ell+1)$-dimensional column vector whose i-th coefficient is $\langle \mathbf{s}_{\ell+1}, \mathbf{s}_i \rangle$.

Algorithm 8: The ffSampling algorithm ([34])

```
Input: $\mathbf{t} = (\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_\ell)$ in FFT format, tree $T$.
Output: $\mathbf{z} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_\ell)$ in FFT format.
1  if $n = 1$ then
2      $\sigma' \leftarrow T$.value.
3      $z_0 \leftarrow \mathcal{D}_{\sigma', t_0}$.
4      $z_1 \leftarrow \mathcal{D}_{\sigma', t_1}$.
5      return $\mathbf{z} = (z_0, z_1)$.
6  else
7      $m \leftarrow$ number of children of $T$.
8      for $i \leftarrow m, \ldots, 0$ do
9          $T_j \leftarrow j$-th child of $T$.
10         $\mathbf{t}'_j \leftarrow \mathbf{t}_j + \sum_{i=j+1}^{m} (\mathbf{t}_i - \mathbf{z}_i) \cdot T.\text{value}_{i,j}$.
11         $\mathbf{t}'_j \leftarrow \text{splitfft}(\mathbf{t}'_j)$.
12         $\mathbf{z}'_j \leftarrow \text{ffSampling}(\mathbf{t}'_j, T_j)$.
13         $\mathbf{z}_j \leftarrow \text{mergefft}(\mathbf{z}'_j)$.
14     return $\mathbf{z} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_m)$.
```

D Key and Ciphertext Size Calculations

The Latte keys and ciphertexts are mainly collections of polynomials in $\mathcal{R}$. The degree of each polynomial is N and the number of bits in each coefficient is $\kappa = \lceil \log_2 q \rceil$. The parameters N and q are dependent on the security level required, and values for these are given in Table 14. The key/ciphertext bit-size is equal to $N \cdot \kappa \cdot$ *number of polynomials*, plus any additional bit strings sent, in the case of the ciphertext. Furthermore, we usually consider the key and ciphertext sizes in bytes, and so when the total bit-size is computed, it will be divided by 8 to give the size in bytes.

D.1 Master Keys

The master public key consists of two polynomials $\mathbf{h}, \mathbf{b} \in \mathcal{R}_q$. Therefore the bit-size is $2N \cdot \kappa$. The master private key $\mathbf{S}_0$ consists of $(\mathbf{f}, \mathbf{g}, \mathbf{F}, \mathbf{G})$. However, $\mathbf{F}$ and $\mathbf{G}$ can be recomputed on the fly from $\mathbf{f}$ and $\mathbf{g}$ using NTRUSolve. The solution is not unique but as long as it is a short solution, it will suffice. However, this is not efficient and so this research considers the entire $(\mathbf{f}, \mathbf{g}, \mathbf{F}, \mathbf{G})$ to be stored as the private key. Therefore, the master private key is of size $4N \cdot \kappa$.

D.2 Delegated Keys

The delegated public key can be straightforwardly generated using the master public key and the chain of user IDs along which the delegation process is happening. Although this can be efficiently generated on the fly, given the user ID chain, we will consider it being stored as the polynomials

Table 14: LATTE Master Key, Delegated Key and Ciphertext Sizes (Bytes)

Set	**Master Public Key**	**Master Private Key**	**User Private Key**		**Ciphertext**	
			$\ell = 1$	$\ell = 2$	$\ell = 1$	$\ell = 2$
LATTE-0	2048	4096	3072	4096	3104	4128
LATTE-1	6144	12288	9216	-	9248	-
LATTE-2	12800	25600	19200	-	19232	-
LATTE-3	9216	18432	13824	18432	13856	18464
LATTE-4	19456	38912	29184	38912	29216	38944

Set	**Delegated Public Key**	**Delegated Private Key**
LATTE-0	3072	9216
LATTE-3	13824	41472
LATTE-4	29184	87552

$\mathbf{h}, H(\mathsf{ID}_1), H(\mathsf{ID}_2), ..., H(\mathsf{ID}_{\ell+1})$, which translates as $(\ell+2)$ polynomials in $\mathcal{R}$, and so the total bit-size of the delegated public key is $(\ell+2) \cdot N \cdot \kappa$. The delegated private key generated from level $\ell - 1$ to level ℓ, to be passed onto users at level $\ell + 1$, is a $(\ell+2) \times (\ell+2)$ matrix of polynomials in $\mathcal{R}_q$. Its size is therefore $(\ell+2) \cdot (\ell+2) \cdot N \cdot \kappa$.

D.3 User Private Keys

The user public key is entirely dependent on the identity, and so we only examine the size of the user private key. In LATTE for a user at level ℓ, this is a tuple of $(\ell+2)$ polynomials in $\mathcal{R}_q$ and so is of bit size $(\ell+2) \cdot N \cdot \kappa$ (which is coincidentally equivalent to the delegated public key size).

D.4 Ciphertexts

Let's consider the ciphertext at level ℓ. This consists of $\ell + 2$ polynomials $\mathbf{C}_1, \ldots, \mathbf{C}_\ell, \mathbf{C}_h, \mathbf{C}_b \in \mathcal{R}_q$ along with a 256-bit string Z (which is essentially the encrypted message). Therefore, at level ℓ, the bit-size of the full ciphertext is $(\ell+2) \cdot N \cdot \kappa + 256$.

E Comparison to a Non-quantum-safe HIBE

Pairings-based HIBE scheme performance results from [26] (converted to op/s at 4.2GHz) are given in Table 15. Parameter s is the bit size of the field, which is comparable to the bit-length of RSA modulus by providing the same security. Although not directly comparable, these results give a good indication of the feasibility of LATTE at levels 1 or 2.

Table 15: Performance Results (op/s) for the Gentry-Silverberg HIBE Scheme (2-level) Using Java v1.6 [26] (Scaled to 4.2GHz)

s	**Security**	**length of m**	**Enc**	**Dec**
1024	80	160	11.5	6.7
3072	128	256	0.8	0.5