New Post-Quantum IBE leveraging maturity, efficiency and security of standard schemes

Julien CAM[1,2]

[1] Kudelski Group julien.cam@nagra.com
[2] University of Rennes julien.cam@univ-rennes.fr

Abstract. Many Identity-Based Encryption (IBE) schemes rely on the hardness of the Discrete Logarithm Problem, making them vulnerable to quantum attacks. In recent years, lattice-based cryptography has emerged as a source of Post-Quantum cryptosystems, for example with Kyber, Dilithium and Falcon chosen by NIST to be standardized as ML-KEM, ML-DSA and FN-DSA. Some IBEs have also been proposed over lattices, but they can still be considered as interesting theoretical constructions, the community's attention having been more on the NIST competition than on optimizing IBEs, assessing their security, and protecting them against physical attacks. So, in this paper, we build a new IBE scheme from the basic blocks of the highly studied ML-KEM, ML-DSA and ModFalcon, to put both their maturity and efficiency at the benefits of IBE. More precisely, we leverage the Module-NTRU trapdoor from ModFalcon to enable extraction of secret keys, we use the encryption and decryption algorithms from ML-KEM, and the modular arithmetic and Number-Theoretic Transform from ML-DSA. Therefore, being able to reuse code from widely deployed schemes, our IBE is much easier to implement and maintain in practice, and can benefit from existing (and future) optimizations, hardware accelerators, and side-channel protections, with very little to no additional work. We also prove the IND-sID-CPA-security of our scheme under standard assumptions, and precisely describe the algorithms and the choice of concrete parameters.

Keywords: Post-Quantum Cryptography · Identity-Based Encryption · ML-KEM · ML-DSA · ModFalcon.

1 Introduction

1.1 Identity-Based Encryption

Let's consider two people (Alice and Bob as usual) wanting to establish a shared secret over a public channel. To do that, they will use public-key cryptography, so each of them needs to have a secret key, and to know the other's public key. However, because they don't trust the network, reliably learning the other's public key is a challenge. In practice, they mainly have two solutions: either a Public-Key Infrastructure (PKI) or an Identity-Based Encryption (IBE) scheme.

PKI: This requires a trusted Certification Authority (CA), securely connected to every participant, whose verification key is known by everyone. Having generated a key pair, Alice asks the CA to sign a *certificate* containing both her identity and her public key. Once Bob has gone through the same process, they can exchange their certificates and verify them with the CA's public key, to reliably know the other's public key.

IBE: This requires a trusted Private Key Generator (PKG), securely connected to every participant, able to compute a secret key from an identity. Then, each participant can compute the other's public key from their known identity.

Comparison: First, IBE reduces the number of messages exchanged over the network, since it does not require any certificate to be sent. Then, it uses fewer cryptographic algorithms: an IBE scheme is self-sufficient, while a PKI requires a Digital Signature Algorithm to exchange public keys (with integrity), and another scheme to exchange a shared secret (with confidentiality). Moreover, since its first proposal by Shamir in 1984, Identity-Based Cryptography has given simple solutions to many specific problems that would have been hard to solve with a PKI [Sha84]. For example, the users can send ciphertexts without having first to receive a certificate, bringing resiliency to congestion and disruption, enabling deferred delivery, and improving real-time communication [AKG+07,Nat16,Gro12]. IBE can also enable encrypted messaging inside an organization (government, company...) while allowing a unique authority to audit the communications if and when needed [Nat16]. As another example, broadcast encryption can leverage the ability to choose the public key to set it to a policy, only giving the appropriate users the corresponding secret key [TWZL08].

The quantum threat: The first IBE schemes were proposed in 2001, based on bilinear pairings over elliptic curves and quadratic residues [BF01,Coc01]. However, they could be efficiently broken by Shor's quantum algorithm [Sho94]. With the current fast development of quantum computing, we would rather follow the global transition to post-quantum schemes, for which we don't know any algorithm (neither classical nor quantum) that would break them.

1.2 Lattice-based IBE schemes

Lattice-based cryptography has emerged as a good candidate to replace number-theoretic-based cryptography in a world where attackers have (or will have) access to quantum computers. Indeed, this field has already allowed to efficiently instantiate a wide range of schemes, such as Key Encapsulation Mechanisms or Digital Signature Algorithms. In particular, Kyber, Dilithium and Falcon are three lattice-based schemes that have been chosen by the US National Institute of Standards and Technology to be standardized, as ML-KEM, ML-DSA and FN-DSA [SAB+22,LDK+22,PFH+22,Nat24b,Nat24a].

Moreover, several lattice-based IBE schemes have been proposed, relying on problems like Learning With Errors (LWE), or NTRU. In 2008, Gentry, Peikert and Vaikuntanathan proposed a generic framework to build an IBE scheme from a trapdoored one-way function [GPV08]. This framework could be instantiated, but that led to schemes that were not practical due to the size of the keys.

In 2014, Ducas, Lyubashevsky and Prest used an NTRU trapdoor to instantiate this framework, leading to the first practical lattice-based IBE [DLP14]. Although it is still one of the most efficient IBE schemes, it suffers from the inflexibility in the choice of parameters inherent to the ring setting. In an attempt to solve that, Cheon, Kim, Kim and Son tried to generalize the NTRU trapdoor to the module setting [CKKS19]. However, their work has been aborted because this generalization was concurrently proposed to improve the flexibility of the choice of parameters in Falcon, resulting in the ModFalcon scheme [CPS+20].

In parallel to the work conducted with NTRU trapdoors, Micciancio and Peikert proposed a notion of gadget trapdoors in 2012 [MP12]. The efficiency of such trapdoors have been improved in 2018 by Chen, Genise and Mukherjee, who introduced the notion of approximate trapdoors [CGM19]. Since these works also suffered from the drawbacks of the ring setting, they were extended to the module setting in 2023 by Izabachène, Prabel and Roux-Langlois, who used them to build an IBE scheme [IPR23]. However, since no standardized scheme use such approximate gadget trapdoors, these schemes have been less studied than the NTRU-based ones, which benefitted from the focus on Falcon. In this paper, we thus focus on the NTRU-based family, and more particularly [DLP14], since it is still the most practical lattice-based IBE scheme.

1.3 Contributions

Although some existing IBEs (including [DLP14], [CKKS19] and [IPR23]) are practical, they have some drawbacks, that we detail below. We address them by building an IBE from standardized schemes, in order to leverage their maturity and efficiency, but also to minimize the engineering cost of deploying an IBE scheme in practical applications, by enabling it to reuse code and scientific knowledge from widely adopted schemes. More precisely, we propose a new IBE based on ML-KEM, with some parameters and algorithms from ML-DSA, leveraging the MNTRU trapdoor from ModFalcon to enable secret key extractions.

Strong security via tight reduction: Because of their similar structure inherited from the GPV framework and (M)NTRU trapdoor, our IBE, [DLP14] and [CKKS19] all achieve the same security properties. However, this paper uses a different approach to assess the security level of an instantiation.

Indeed, the security of [DLP14] and [CKKS19] is analyzed via ad-hoc cryptanalysis, against the best attack found by the authors. So, the `IND-ID-CPA`-security of their parameters is not proven, it was believed by the authors at publication time. The problem can be illustrated by [CKKS19] proposing insecure parameters, analyzed against a non-optimal attack (see Section 5).

Although the same approach can apply to our IBE, we rather chose to give a tight proof of its security under standard assumptions, in the `IND-sID-CPA` model. Since our proof can also apply to [DLP14] and [CKKS19], we emphasize that the difference lies in the paper's redaction, not in the schemes' security.

Simple security analysis: Beyond the stronger theoretical guarantee, our approach also has practical advantages. Indeed, while the security analysis of previous NTRU-based IBEs is a manual process, we can perform the analysis

via a well-maintained and open-source tool, LatticeEstimator [APS15]. When a new attack is proposed against a lattice-based problem, we must manually assess the impact on the parameters of [DLP14], while we can simply use up-to-date LatticeEstimator to assess the impact on our instantiation. This is especially relevant because of the quick evolution of lattice-based cryptanalysis, that makes the original parameters of [DLP14] currently insecure (from LatticeEstimator).

Straightforward to implement: We build and describe our scheme in a way that makes it easy to implement. For example, all of our keys are byte-strings, and not abstract mathematical objects. In [DLP14] and [CKKS19], the master public key contains integers, which does not exist on a computer (knowing at best "integers on a fixed number of bits"). Moreover, byte-strings make the size estimations more direct and accurate. As an independent advantage of our design, while all of the above schemes have to be implemented from scratch, ours reuses code from standardized (thus widely-implemented) schemes.

Number Theoretic Transform: The NTT is a crucial operation for the efficiency of lattice-based schemes. In [DLP14], the conditions on the modulus to be compatible with NTT are presented, but the implementation provided by the authors does not use NTT, defining the modulus to be a power-of-two, incompatible with that operation. The implementation of [CKKS19] follows the same path, and the paper does not even mention NTT. Conversely, our scheme uses parameters that are fully compatible with NTT, and all of its algorithms explicitly use it to improve their efficiency.

Optimization and securing: If we choose to use an IBE scheme, it may need (for example) to run faster, to use less memory, or to resist to physical attacks (side-channel/fault-injection), depending on its application context. To achieve that, the scheme has to go through a careful (thus long and costly) analysis, to determine what is the bottleneck/leakage/vulnerability, and how we should modify the code, or which hardware we would need. Since our scheme is built to reuse existing blocks of standardized implementations, it also allows to reuse the optimizations, the hardware and the countermeasures that have been developed for it, making the above process a lot easier, or even free, in practice.

Efficiency: Our goal is to design an IBE scheme that is able to reuse as much existing code as possible. In particular, we did not try to reduce data sizes, or to increase efficiency, so we can wonder if the resulting scheme is practical or not. Fortunately, a side effect of our design is that our scheme is also close to standardized schemes in terms of efficiency (because we reuse their code) and sizes (because we reuse their formats). Since the standardized schemes have been designed to be efficient and small, our scheme inherits these good aspects.

1.4 Paper organization

We will begin with some preliminaries in Section 2. This will enable fixing the notation, introducing constants, and defining the mathematical spaces and probability distributions, so that the reader can fully understand the rest of the paper.

To transform ML-KEM into an IBE scheme, we first have to define how to map any identity to a valid public key. Then, we must enable the PKG (and only

the PKG) to generate a secret key associated with such a public key. We thus introduce a secret trapdoor in the public matrix, that allows to "invert" the key generation function of ML-KEM. These two steps are described in Section 3.

Beyond theory, one contribution of this paper is to give a very detailed and "implementation-friendly" description of the algorithms, treating keys as byte-strings, and explicitly leveraging NTT, which is achieved in Section 4. In a nutshell (it is of course not so simple in practice), the `Setup` and `Extract` algorithms come from ModFalcon, and `Encrypt` and `Decrypt` are from ML-KEM.

In Section 5, we will use this good description of the real operation of the scheme to analyze its security. More precisely, we will give a tight reduction from the decisional variants of the Ring-LWE and Module-NTRU problems to the `IND-sID-CPA` security of our IBE, in the Random Oracle Model.

Finally, we describe our choice of parameters in Section 6. We first present all the constraints that guide this choice, and then perform some experiments to determine the optimal values. In particular, we found parameters that allow to reach a standard level of provable security, to achieve negligible failure probability, and to mostly reuse existing code from standardized schemes.

2 Preliminaries

2.1 Constants, parameters and spaces

We introduce here some constants and parameters without giving them specific values for now, and refer to Section 6 for how to give them accurate values. We let d_{SK}, d_u, d_v, η_1, η_2, k, n and q denote positive integers such that n is a power of two and q is prime. Moreover, we let σ denote a positive real number. With these parameters, we are able to define some useful spaces:

- $K = \mathbb{Q}[X]/(X^n+1)$ is the cyclotomic field of degree n [CPS+20].
- $R = \mathbb{Z}[X]/(X^n+1)$ is the ring of integers of K [CPS+20].
- $R_q = \mathbb{Z}_q[X]/(X^n+1) = R/qR$ is used in the algorithms for efficiency.
- T_q is the image of R_q under the NTT (see [Nat24a,Nat24b]).
- $\mathbb{B} = \mathbb{Z}_{256}$ is the set of the possible values for a byte [Nat24a,Nat24b].
- For a ring A and $m \in \mathbb{N}^*$, A^m is the set of column vectors of size m over A.

2.2 Matrices

We will use the common notation $\mathrm{GL}_k(R_q)$ to denote the set of invertible matrices of size k over R_q. For the rest of this section, we let A be in $R^{k\times k}$.

We let A^{T} denote the transpose of A. If A is invertible, we also let $A^{-\mathrm{T}}$ denote its inverse transpose $(A^{-1})^{\mathrm{T}} = (A^{\mathrm{T}})^{-1}$. Moreover, we let $\mathrm{adj}\,(A)$ denote the adjugate of A, such that: $\mathrm{adj}\,(A) \times A = A \times \mathrm{adj}\,(A) = \det(A)\, I$.

The lattice $\Lambda(A) = \{Ax\}_{x\in R^k}$ is called the lattice generated by A. The column vectors of A, denoted $(a_i)_{i\in[\![1,k]\!]}$, form a basis of $\Lambda(A)$. The Gram-Schmidt orthogonalization of $(a_i)_{i\in[\![1,k]\!]}$ is denoted $(\tilde{a}_i)_{i\in[\![1,k]\!]}$. We also define the Gram-Schmidt norm of A, which is a crucial quantity for the security of the trapdoor sampling, to be the norm of the largest vector in the Gram-Schmidt basis $(\tilde{a}_i)_{i\in[\![1,k]\!]}$, and we will denote it $\|A\|_{\mathrm{GS}} = \max_{i\in[\![1,k]\!]} (\|\tilde{a}_i\|)$ [GPV08].

2.3 Random distributions

In order to rely on standardized schemes, our IBE will sample random elements from uniform distributions (used in ML-KEM), Centered Binomial Distributions (used in ML-KEM) and discrete Gaussian distributions (used in Falcon).

For any finite set $\mathcal{X}$, we let $x \xleftarrow{\$} \mathcal{X}$ denote the uniform sampling of an element x in $\mathcal{X}$, meaning that each element in $\mathcal{X}$ is sampled with probability $(\#\mathcal{X})^{-1}$.

For any $\eta \in \mathbb{N}^*$, we define CBD_η as the distribution of the sum of η independent random variables, each of them being the difference between two independent Bernoulli-distributed random variables [Nat24b].

For any lattice $\Lambda \subset \mathbb{Z}^m$ and any $(c, \sigma) \in \mathbb{Z}^m \times \mathbb{R}_+^*$, the distribution $\mathcal{N}_{\Lambda,c,\sigma}$ assigns to any $x \in \Lambda$ a probability proportional to the evaluation of the gaussian density function on x, that is: $\exp\left(-\|x-c\|^2/(2\sigma^2)\right)$ [CPS+20].

2.4 Trapdoor sampling

To extract secret keys, we need to sample from discrete Gaussian distributions over lattices. This is a common practice in lattice-based cryptography, and we present our sampler in Algorithm 1, based on the most standard one [GPV08].

Algorithm 1 `GaussianSampler`

Input: A matrix $M \in R^{m \times m}$, for some integer m
Input: A center $c \in R^m$
Input: A standard deviation $\sigma \in \mathbb{R}_+^*$
Output: $s \in \Lambda(M)$ sampled from $\mathcal{N}_{\Lambda(M),c,\sigma}$

1: Build $B \in \mathbb{Z}^{nm \times nm}$ by replacing each element of M by its anti-circulant matrix
2: Build $x_{nm} \in \mathbb{Z}^{nm}$ as the concatenation of the coordinate vectors of c's elements
3: **for** $i = nm$ down to 1 **do**
4: Sample $z_i \in \mathbb{Z}$ from $\mathcal{N}_{\mathbb{Z},c_i,\sigma_i}$, with $c_i = \dfrac{\langle x_i|\tilde{b}_i\rangle}{\langle \tilde{b}_i|\tilde{b}_i\rangle}$ and $\sigma_i = \dfrac{\sigma}{\langle \tilde{b}_i|\tilde{b}_i\rangle}$
5: Compute $x_{i-1} = x_i - z_i \times b_i$
6: **end for**
7: **return** $x_{nm} - x_0$

3 Overview of the scheme

3.1 Building an IBE from ML-KEM

The Key-Encapsulation Mechanism ML-KEM is derived from the Public-Key Encryption scheme K-PKE via the Fujisaki-Okamoto transform [SAB+22,FO99]. In this paper, we build an Identity-Based Encryption scheme, which can also be transformed into an Identity-Based KEM by similar techniques. So, our goal is to build an IBE scheme such that the encryption and decryption functions are the same as those of K-PKE [Nat24b]. At a high level, this IND-CPA-secure encryption scheme can be summarized as follows:

- K-PKE.`KeyGen`()
  - Sample $A \in R_q^{k \times k}$ from uniform and $e \in R_q^k$ and $s \in R_q^k$ from CBD_{η_1}.
  - With $t = A \times s + e$, output $pk = (A, t)$ and $sk = s$.
- K-PKE.`Encrypt`$((A, t), m)$
  - Sample $y \in R_q^k$ from CBD_{η_1} and $e_1 \in R_q^k$ and $e_2 \in R_q$ from CBD_{η_2}.
  - Output $\begin{cases} u = A^{\mathrm{T}} \times y + e_1 \\ v = t^{\mathrm{T}} \times y + e_2 + \lfloor q/2 \rceil m \end{cases}$
- K-PKE.`Decrypt`$(s, (u, v)) = \lfloor (v - s^{\mathrm{T}} \times u) / \lfloor q/2 \rceil \rceil$.

A natural way to define the keys for our IBE is to use A as the PKG's public key, and (t, s) as the user's key pair. However, for correctness, s and $t - A \cdot s$ must be short vectors. This means that, for any $t \in R_q^k$, the PKG must find a short $s \in R_q^k$ such that $t - A \cdot s$ is short, which is known as the Module-LWE problem. Since solving that for a uniform matrix A is hard, the public matrix of the PKG must be constructed together with a trapdoor for the Module-LWE problem (which will be the secret key of the PKG), while looking like a uniform matrix. Finally, the operation of our IBE scheme will look like this:

- `Setup`()
  - Sample a uniform-looking $PK \in R_q^{k \times k}$ with a LWE-trapdoor SK.
- `Extract`$(SK, \texttt{ID})$
  - Compute $pk \in R_q^k$ from `ID`.
  - Use SK to find a short vector $sk \in R_q^k$ such that $pk - PK \cdot sk$ is short.
- `Encrypt`$(PK, \texttt{ID}, m)$ = K-PKE.`Encrypt`$((PK, pk), m)$.
  With $pk \in R_q^k$ computed from `ID` as in `Extract`.
- `Decrypt`(sk, c) = K-PKE.`Decrypt`(sk, c).

3.2 Mapping an identity to a public key

The goal of an IBE scheme is that a user's public key can be derived from their identity. So, we must specify what is an identity and how to derive a public key from it. Usually, we consider identities as bit-strings, to include any data that can be handled by a computer, we hash it, and encode it into a valid public key in the corresponding domain [GPV08].

So, we must publicly encode a bit-string into a valid public key for ML-KEM. However, for a given $A \in R_q^{k \times k}$, such a valid public key is any $pk \in R_q^k$ that can be written as $pk = A \times sk + e_k$, with some very short $sk \in R_q^k$ and $e_k \in R_q^k$. Therefore, a valid public key is a vector that is very close to a small multiple of A, and the associated secret key is the lattice element close to this vector. Unfortunately, we cannot publicly know where are the vectors close to these elements without knowing which is this close element.

The solution is to allow larger sk and e_k [DLP14]. That way, valid public keys are vectors father away from higher multiples of A, multiplying their number until they fill the whole space. This idea is illustrated in Figure 1, using toy

parameters to compare the situations in ML-KEM (valid public keys are sparse) and in our scheme (any vector is a valid public key).

The advantage of this approach is that any encoding of a bit-string into a vector actually maps an identity to a valid public key, so we decided to reuse the `SampleNTT` function from ML-KEM, which maps a bit-string to a uniform vector in $T_q^k \simeq R_q^k$, via a cryptographic hash function. The drawback is that both the secret key sk and the error term e_k will be much larger than in ML-KEM, increasing the probability of a decryption failure. This explains why IBEs must use a larger modulus than ML-KEM [DLP14,CKKS19].

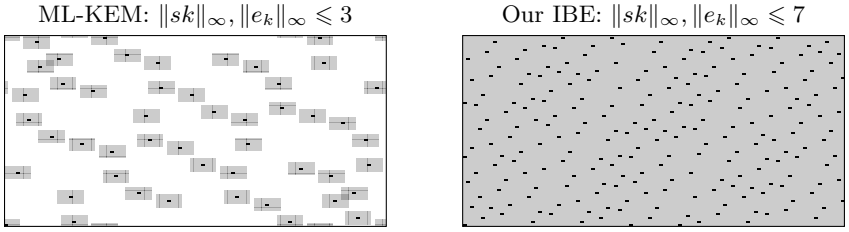

Fig. 1. Illustration of the sparsity of public keys with toy parameters ($q = 101$)

3.3 The module-NTRU trapdoor

During the extraction of a user's secret key, the PKG must use a trapdoor to invert the Module-LWE function $s \mapsto As + e$. We leverage the Module-NTRU (MNTRU) trapdoor introduced in [CPS+20,CKKS19].

In a nutshell, the generation of this trapdoor is as follows. First, we sample the coefficients of $(f, g) \in R^{k \times k} \times R^k$ from narrow centered gaussians, and we compute the public key as $h = f^{-\mathrm{T}} \times g \mod q$. Then, we compute a short solution $(F, G) \in R^k \times R$ to the MNTRU equation:

$$\det_K(f) \times G - g^{\mathrm{T}} \times \mathrm{adj}(f) \times F = q$$

The key property is that $B_{f,g}$ is a good basis of the MNTRU lattice $\Lambda_{h,q}$, with:

$$B_{f,g} = \begin{pmatrix} g^{\mathrm{T}} & G \\ -f & -F \end{pmatrix} \quad \text{and} \quad \Lambda_{h,q} = \left\{ (u, v) \in R^{k+1},\ u + hv = 0 \mod q \right\}$$

Compared to an NTRU trapdoor (the special case $k = 1$), a MNTRU trapdoor is bigger, but allows to use a smaller standard deviation σ in the `Extract` algorithm. Indeed, σ is lower-bounded by some quantity roughly proportional to $q^{1/(k+1)}$. For our chosen modulus q, using $k = 1$ would not allow to achieve negligible decryption failure probability, so we instead use $k = 2$, at the price of about doubling the data sizes (see Section 6 for more details).

4 The operation of the scheme

4.1 Setup

This algorithm basically consists in the generation of a MNTRU trapdoor. First, we sample the coefficients of $f \in R^{k \times k}$ and $g \in R^k$ from narrow integral Gaussian distributions, and we restart until $f \in \mathrm{GL}_k(R_q)$ and the MNTRU equation has a solution. Then, we complete $B_{f,g}$ with a solution of that equation, restarting if $\|B_{f,g}\|_{\mathrm{GS}}$ is too high, and finally compute $PK = f^{-\mathrm{T}} \times g \mod q$ [CPS+20].

Sampling f and g: From the original IBE over NTRU lattices, there has always been the idea to sample the coefficients of f and g from narrow integral Gaussian distributions, whose standard deviations are chosen to minimize $\|B_{f,g}\|_{\mathrm{GS}}$ [DLP14]. The authors there conjectured that we can optimally hope for $\|B_{f,g}\|_{\mathrm{GS}} \leqslant 1.17\sqrt{q}$, where the value 1.17 was determined from experiments. This was generalized to the MNTRU case for [CKKS19] and ModFalcon. Both agree that we can reach $\|B_{f,g}\|_{\mathrm{GS}} \leqslant \mathrm{GS_SLACK}_k \times q^{1/(k+1)}$, where $\mathrm{GS_SLACK}_k$ is a function of k such that $\mathrm{GS_SLACK}_1 = 1.17$. However, they disagree on the value of $\mathrm{GS_SLACK}_2$ (1.2 vs 1.17) [CKKS19,CPS+20]. From our experiments, we found that $\mathrm{GS_SLACK}_2 = 1.2$ gives the best results.

Solving the MNTRU equation: Our IBE scheme directly uses the MNTRU solver from ModFalcon [CPS+20], which is a slight modification of the NTRU solver from Falcon [PFH+22, Algorithm 6]. More precisely, this MNTRU solver first computes $\alpha = \det_K(f)$ and sets β to be the first coordinate of $g^{\mathrm{T}} \operatorname{adj}(f)$, and then uses the NTRU solver to find a short solution (F_0, G) to the NTRU equation: $\alpha \times G - \beta \times F_0 = q$. Finally, defining $F = (F_0, 0, \cdots, 0) \in R^k$, the pair (F, G) is a short solution to the MNTRU equation.

Computing the Gram-Schmidt norm of the trapdoor: For security reasons, the standard deviation σ used to extract secret keys is lower-bounded by a function of $\|B_{f,g}\|_{\mathrm{GS}}$. There, we can use the fact (explained above) that we can expect $\|B_{f,g}\|_{\mathrm{GS}} \leqslant \mathrm{GS_SLACK}_k \times q^{1/(k+1)}$ with good probability. So, we can instead lower-bound σ by a function of $\mathrm{GS_SLACK}_k \times q^{1/(k+1)}$, allowing this parameter to be independent of the secret trapdoor. Thus, in practice, we should ensure that the above inequality holds, restarting if it does not. However, solving the MNTRU equation is a costly operation. Fortunately, the MNTRU structure allows to compute $\|B_{f,g}\|_{\mathrm{GS}}$ only from f and g, allowing to restart before pointlessly solving the equation if it is too high [CPS+20].

Encoding the master secret key: In many lattice-based IBE schemes, the master secret key is a matrix of (theoretically unbounded) integers. This poses problems for implementation, as it is not clear how to represent such objects on a computer. For that reason, we choose to instead encode the master secret key as a byte array. To do that, we leverage the encoding function already defined in ML-KEM, allowing to encode a list of integers into a byte array, using a predefined number of bits per integer [Nat24b]. So, we need to tell the encoder in advance the number d_{SK} of bits it should use to encode each coefficient of the master secret key. Unfortunately, the coefficients of f and g are unbounded, since they are sampled from Gaussian distributions over the integers.

One solution is to bound the tail of a Gaussian distribution, which allows us to choose d_{SK} to make the probability of an overflow negligible. However, we now face another problem on F and G. These are a solution to the MNTRU equation, and not the output of a Gaussian sampler with known parameters. For that reason, we use the same approach as in Falcon [PFH+22, Section 4.4.3]: we run the MNTRU solver on a large number of pairs (f, g), and define d_{SK} as the maximum bit-size of the coefficients of F and G. The results on our parameters can be found in Section 6.

Another problem we encounter when encoding the coefficients of the master secret key is that they can either be positive or negative, while the encoder from ML-KEM expects non-negative integers. To fix that, we add $2^{d_{SK}-1}$ to each coefficient. We restart the `Setup` if the result is outside $[\![0, 2^{d_{SK}} - 1]\!]$, and, otherwise, we encode it into d_{SK} bits.

Finally, ML-KEM only specifies the encoding of a vector of polynomials. So, we need to give an order for encoding the polynomial matrix f. In our scheme, we chose to use row-major order, i.e., we first encode the coefficients of the first row, then the second row, and so on. We can also note that, since the MNTRU solver outputs F as a vector with only one non-zero coordinate, we can only encode F_0 into the key.

The Number-Theoretic Transform: The use of NTT accelerates polynomial multiplications, so it is used in the format of the public matrix in ML-KEM. Therefore, we want (and need) to output the master public key in NTT domain, so that it can directly be used as a ML-KEM public matrix.

Algorithm 2 `Setup`

Output: A master secret key $SK \in \mathbb{B}^{(k+1)^2\, n\, d_{SK}/8}$
Output: A master public key $PK \in \mathbb{B}^{kn\lfloor \log_2(q)+1 \rfloor/8}$

repeat
  repeat
    repeat
      for $j = 1$ to k **do**
        Sample $g_j \in R$ from $\mathcal{N}_{\mathbb{Z},\sigma_j}$, with $\sigma_j = \dfrac{\text{GS_SLACK}_k \times q^{1/(k+1)}}{\sqrt{n(k-j+2)}}$
        for $i = 1$ to k **do**
          Sample the coefficients of $f_{i,j} \in R$ from $\mathcal{N}_{\mathbb{Z},\sigma_j}$
        end for
      end for
    until $\|B_{f,g}\|_{\text{GS}} \leqslant \text{GS_SLACK}_k \times q^{1/(k+1)}$ and $f \in \text{GL}_k(R_q)$
  until $\perp \neq$ `SolveMNTRU`$(f, g) = (F, G)$
  Define (f', g', F'_0, G') by adding $2^{d_{SK}-1}$ to every coefficient of (f, g, F_0, G)
until Every coefficient of (f', g', F'_0, G') is in $[\![0, 2^{d_{SK}} - 1]\!]$
Compute $h = f^{-\mathrm{T}} \times g \mod q$
Compute $SK =$ `ByteEncode`$_{d_{SK}}(f', g', F'_0, G')$
Compute $PK =$ `ByteEncode`$_{\lfloor \log_2(q)+1 \rfloor}$(`NTT`$(h_1), \cdots,$ `NTT`$(h_k))$
return SK and PK

4.2 Extract

On input a public key $pk \in R_q$ derived from an identity, the goal of the `Extract` algorithm is to find a secret key $sk \in R_q^k$ such that both sk and $pk - PK^{\mathrm{T}} \times sk$ are small polynomials. Our algorithm follows the usual approach of trapdoor sampling, first finding an arbitrary solution to the equation, and then using the trapdoor to reduce that solution while preserving the equation [GPV08].

More precisely, we search for a small $s \in R_q^{k+1}$ such that: $pk = (1, PK)^{\mathrm{T}} \times s$. To do that, we will first remark that $t = (pk, 0, \cdots, 0) \in R_q^{k+1}$ is such that: $(1, PK)^{\mathrm{T}} \times t = pk$. Then, we will use the MNTRU trapdoor SK, which is a good basis of $\Lambda = \{(u, v) \in R \times R^k,\ u + PK^{\mathrm{T}} \times v = 0 \mod q\}$, to sample a $c \in \Lambda$ that is close to t. Finally, we will compute $s = (s_0, \cdots, s_k) = t - c \mod q$ and $sk = (s_1, \cdots, s_k)$. That way, we will have:

$$\begin{aligned} s_0 + PK^{\mathrm{T}} \times sk &= (1, PK)^{\mathrm{T}} \times s \\ &= (1, PK)^{\mathrm{T}} \times t - (1, PK)^{\mathrm{T}} \times c \\ &= pk - 0 \mod q \end{aligned}$$

For security, it is important to never issue two different secret keys for a single public key. Indeed, the difference between two such secret keys would be a short element of Λ. The simplest (from a theoretical point of view) way to tackle this is to remember every generated secret key, although a pseudo-random Gaussian sampler would be much better (simpler, lighter, and more secure).

Moreover, the output of the trapdoor sampler must be indistinguishable from a discrete Gaussian distribution that is independent of the trapdoor, to guarantee that the sampler does not leak the trapdoor. We will refer to ModFalcon, that uses a Rényi divergence argument to prove this if the standard deviation σ is greater than some multiple of $\|SK\|_{\mathrm{GS}}$ [CPS+20].

For compatibility with encryption and decryption from ML-KEM, and in order to optimize these algorithms, we directly compute the public key pk (from `ID`) in NTT domain, and we also output the secret key sk in NTT domain.

Algorithm 3 `Extract`

Input: A master secret key $SK \in \mathbb{B}^{(k+1)^2\ n\ d_{SK}/8}$
Input: A user identity $\texttt{ID} \in \mathbb{B}^{32}$
Output: A user secret key $sk \in \mathbb{B}^{kn\lfloor \log_2(q)+1 \rfloor/8}$

1: **if** `ID` has already been queried **then**
2: **return** The previously computed sk
3: **end if**
4: $pk = \texttt{SampleNTT}(\texttt{ID} \parallel 0 \parallel 0)$.
5: $t = (\texttt{NTT}^{-1}(pk), 0, \cdots, 0) \in R^{k+1}$
6: Subtract $2^{d_{SK}-1}$ from each coefficient of $\texttt{ByteDecode}_{d_{SK}}(SK)$ and build $B_{f,g}$
7: With $c \leftarrow \texttt{GaussianSampler}(B_{f,g}, t, \sigma)$, compute $s = (s_0, \cdots, s_k) = t - c \mod q$
8: **return** $sk = \texttt{ByteEncode}_{\lfloor \log_2(q)+1 \rfloor}(\texttt{NTT}(s_1), \cdots, \texttt{NTT}(s_k))$

4.3 Encrypt

The original encryption algorithm from [CKKS19] is based on a usual Ring-LWE encryption [LPR12]. More precisely, it first samples small polynomials $y \in R_q$, $e_1 \in R_q^k$ and $e_2 \in R_q$, and then computes the ciphertext as:

$$\big(y \times PK + e_1,\ y \times pk + e_2 + \lfloor q/2 \rceil m\big) \in R_q^k \times R_q$$

Our original objective is to reuse the `Encrypt` and `Decrypt` from ML-KEM. However, in the K-PKE encryption scheme, the ciphertext is rather computed by sampling small $y \in R_q^k$, $e_1 \in R_q^k$ and $e_2 \in R_q$, and then computing [Nat24b]:

$$\big(A^{\mathrm{T}} \times y + e_1,\ t^{\mathrm{T}} \times y + e_2 + \lfloor q/2 \rceil m\big) \in R_q^k \times R_q$$

In order to reuse the `Encrypt` from ML-KEM to compute a ciphertext for our IBE, we use a little trick: if the first row of A (resp. t) is PK (resp. pk), and the other rows are all zeros, then we can check that the ciphertext computed by ML-KEM indeed corresponds to the ciphertext from [CKKS19] described above.

Moreover, in ML-KEM, the public matrix A is expanded to a uniform matrix from a seed in the public key. In an IBE scheme, that is not possible, because the authority must know a trapdoor for this matrix. However, finding a trapdoor for a uniform matrix A is a hard problem (lattice reduction), and finding a seed that generates a chosen matrix A is also hard (inverting a hash function). Consequently, our scheme cannot use the same format to store the public matrix, and the construction of the matrix will be a little different.

Algorithm 4 `Encrypt`

Input: A master public key $PK \in \mathbb{B}^{kn\lfloor \log_2(q)+1 \rfloor/8}$
Input: A user identity $\texttt{ID} \in \mathbb{B}^{32}$
Input: A message $m \in \mathbb{B}^{n/8}$
Output: A ciphertext $c \in \mathbb{B}^{n(d_u k + d_v)/8}$

Create $\hat{A} \in T_q^{k \times k}$, whose first row is $\texttt{ByteDecode}_{\lfloor \log_2(q)+1 \rfloor}(PK) \in T_q^k$
Create the vector $\hat{t} \in T_q^k$, whose first element is $\texttt{SampleNTT}(\texttt{ID} \parallel 0 \parallel 0) \in T_q$
for $i = 1$ to k **do** ▷ Sample y and e_1 in R_q^k
    $y_i \leftarrow \texttt{SamplePolyCBD}_{\eta_1}$
    $e_{1,i} \leftarrow \texttt{SamplePolyCBD}_{\eta_2}$
end for
$e_2 \leftarrow \texttt{SamplePolyCBD}_{\eta_2}$
$\hat{y} = \texttt{NTT}(y)$
$u = \texttt{NTT}^{-1}(\hat{A}^{\mathrm{T}} \circ \hat{y}) + e_1$
$\mu = \texttt{Decompress}_1(\texttt{ByteDecode}_1(m))$
$v = \texttt{NTT}^{-1}(\hat{t}^{\mathrm{T}} \circ \hat{y}) + e_2 + \mu$
$c_1 = \texttt{ByteEncode}_{d_u}(\texttt{Compress}_{d_u}(u))$
$c_2 = \texttt{ByteEncode}_{d_v}(\texttt{Compress}_{d_v}(v))$
return $c = (c_1 \parallel c_2)$

4.4 Decrypt

The `Decrypt` algorithm is already the exact same in ML-KEM and [CKKS19], so we can simply pass our ciphertext encrypted like in [CKKS19] to the `Decrypt` algorithm of ML-KEM.

Algorithm 5 `Decrypt`

Input: A user secret key $sk \in \mathbb{B}^{kn\lfloor \log_2(q)+1 \rfloor/8}$
Input: A ciphertext $c \in \mathbb{B}^{n(d_u k+d_v)/8}$
Output: A message $m \in \mathbb{B}^{n/8}$

$c_1 = c[0 : d_u \times k \times n \,/\, 8]$
$c_2 = c[d_u \times k \times n \,/\, 8 : n \times (d_u \times k + d_v) \,/\, 8]$
$u' = \texttt{Decompress}_{d_u}(\texttt{ByteDecode}_{d_u}(c_1))$
$v' = \texttt{Decompress}_{d_v}(\texttt{ByteDecode}_{d_v}(c_2))$
$\hat{s} = \texttt{ByteDecode}_{\lfloor \log_2(q)+1 \rfloor}(sk)$
$w = v' - \texttt{NTT}^{-1}(\hat{s}^{\mathrm{T}} \circ \texttt{NTT}(u'))$
return $m = \texttt{ByteEncode}_1(\texttt{Compress}_1(w))$

4.5 Proof-of-concept implementation

We implemented the algorithms described in the previous subsections in Python, and the result is available on a public GitHub repository [Cam25]. This clean proof-of-concept implementation has several purposes: illustrate how easy it is to implement this IBE from standardized schemes, check that everything works as expected, and give a (very rough) idea of the running time. Indeed, even if a C implementation would be needed to precisely measure the running time and memory usage, this Python implementation still allows to check that our scheme is roughly practical, since encryption and decryption both take approximately 15ms on a laptop with a 12th gen Intel Core i7 (2.2GHz).

5 Provable security

In this section, we tightly bound our IBE's security to standard problems over lattices. To achieve that, we will extensively use security games, which oppose a challenger $\mathcal{C}$ generating a random instance of some problem or scheme, and an adversary $\mathcal{A}$ trying to solve the problem or to break some security property. At the end of the game, the challenger outputs $\top$ if the adversary succeeded, and $\bot$ otherwise. In this paper, the adversary will have to guess a bit $b \in \{0, 1\}$. So, we define the (normalized) advantage of an adversary $\mathcal{A}$ in a game $\mathcal{G}$ as:

$$\mathrm{Adv}_{\mathcal{G}}(\mathcal{A}) = 2\left|\mathbb{P}(\top) - 1/2\right|$$

A problem is said to be hard, or a scheme is said to be secure (for some definition of security), if no polynomial-time adversary has non-negligible advantage in the corresponding game [BF01].

5.1 Security models for IBE

The definition of the game is what defines the security property, which is denoted by an acronym containing first the target of the adversary (what they try to do), and then their possibilities (how they can do it).

Regarding encryption, there mainly are two targets that are used in practice. One-Wayness (`OW`) is the weakest security property (it is the hardest to break, so the easiest to achieve), asking $\mathcal{A}$ to recover the message hidden in a random ciphertext c. Indistinguishability (`IND`) is the stronger property that we want in practice. Here, $\mathcal{A}$ first chooses two messages m_0 and m_1 and sends them to the challenger. $\mathcal{C}$ picks a random bit b and sends an encryption c of m_b. The goal of $\mathcal{A}$ is then to guess the value of b.

In any security game related to public-key cryptography, $\mathcal{C}$ generates a key pair and gives the public key to $\mathcal{A}$. Then, for usual Public-Key Encryption, we mainly have two possibilities. In a Chosen-Plaintext Attack (`CPA`), $\mathcal{A}$ can only use the public key they received to encrypt messages. In the stronger Chosen-Ciphertext Attack (`CCA`) model, $\mathcal{A}$ is given access to a so-called decryption oracle, which means that $\mathcal{A}$ can ask $\mathcal{C}$ to decrypt any ciphertext (except c).

When considering Identity-Based Encryption, $\mathcal{A}$ can choose the challenge identity $\texttt{ID}^*$ that $\mathcal{C}$ will use to encrypt the challenge ciphertext, but we have here two possible security models. In the Selective-Identity model (`sID`), $\mathcal{A}$ chooses $\texttt{ID}^*$ at the very beginning of the game, before $\mathcal{C}$ has even generated a key pair. In the stronger Full-Identity model (`ID`), $\mathcal{A}$ chooses $\texttt{ID}^*$ just before receiving the challenge ciphertext c. Regardless of this choice, $\mathcal{A}$ is given access to an extraction oracle, which means that $\mathcal{A}$ can ask $\mathcal{C}$ to get the secret key associated to any identity (except $\texttt{ID}^*$) [GH05].

In our paper, we leverage the widely-used Random Oracle Model, which is a way to handle an ideal hash function in security proofs. Concretely, we assume the underlying hash functions to be cryptographically-secure, preventing $\mathcal{A}$ from relying on any particular structure on their outputs. Therefore, we model that by giving $\mathcal{A}$ access to a so-called random oracle, which means that $\mathcal{A}$ can ask $\mathcal{C}$ for the hash of any identity [BR93].

Putting it all together, the strongest security model for a PKE scheme is `IND-CCA`. However, in 1999, Fujisaki and Okamoto proposed a way to build an `IND-CCA`-secure scheme given an `IND-CPA`-secure one, with little impact on performances [FO99]. Therefore, many Key-Establishment Mechanisms, including ML-KEM, are built from an `IND-CPA`-secure PKE scheme with this transform [Nat24b]. Since it also applies to IBE, we can focus on the `CPA` model like it was done for [DLP14].

Therefore, regarding IBE, we would want to prove the `IND-ID-CPA`-security, enabling to reach `IND-ID-CCA`-security via the FO-transform. However, this model is not really convenient for security proofs. This mainly comes from the impossibility for $\mathcal{C}$ to choose a special value for the hash of the challenge identity, since $\mathcal{A}$ could query the hash of $\texttt{ID}^*$ before sending it to $\mathcal{C}$.

One solution can be to try to guess which of the queried identity will be the challenge, but this engenders a loss in the tightness of the reduction [GPV08].

Another solution can be to analyze the best attack we find against the scheme, but this results in a (debatable) empirical hint, not a proof [DLP14]. For example, [CKKS19] based its security on a non-optimal attack, leading to an overly optimistic analysis of the security level (more details in Section 5.2). In [DLP14], the best attack directly tries to decrypt the challenge ciphertext, without leveraging any oracle, nor the ability to choose the challenge identity according to the public key and the previous answers of the oracles. Strictly speaking, their analysis is therefore more related to the weakest model (`OW-sID-CPA`) than to the strong target `IND-ID-CPA`. Since their analysis also applies to our scheme, this suggests that stronger adversaries have no further possibilities in practice.

In this paper, we prove that our scheme is `IND-sID-CPA`-secure as soon as the decisional variants of the Ring-LWE and Module-NTRU problems are hard. Interestingly, [DLP14] and our IBE being very similar, they satisfy the same security properties in practice. So, our proof also applies for the `IND-sID-CPA`-security of [DLP14] (replacing MNTRU by NTRU), as well as the analysis from [DLP14] also gives the same confidence in the `IND-ID-CPA`-security of our IBE.

5.2 The underlying hard problems

The security of our IBE scheme relies on the decisional variants of two lattice-related problems: Ring-LWE and Module-NTRU. These two problems are currently conjectured post-quantum because we don't know any algorithm able to win the game with non-negligible advantage, neither for a classical nor for a quantum computer.

Decisional Ring-LWE

- $\mathcal{C}^{\text{RLWE}}$ samples uniform $a \in R_q^{k+1}$.
- $\mathcal{C}^{\text{RLWE}}$ samples uniform $c \in \{0,1\}$.
- If $c = 0$:
  - $\mathcal{C}^{\text{RLWE}}$ samples uniform $b \in R_q^{k+1}$.
- If $c = 1$:
  - $\mathcal{C}^{\text{RLWE}}$ samples $s \in R_q$ from CBD_{η_1}.
  - $\mathcal{C}^{\text{RLWE}}$ samples $e \in R_q^{k+1}$ from CBD_{η_2}.
  - $\mathcal{C}^{\text{RLWE}}$ computes $b = a \times s + e$.
- $\mathcal{C}^{\text{RLWE}}$ sends a and b to $\mathcal{A}^{\text{RLWE}}$.
- $\mathcal{A}^{\text{RLWE}}$ sends $c' \in \{0,1\}$ to $\mathcal{C}^{\text{RLWE}}$.
- $\mathcal{C}^{\text{RLWE}}$ outputs $\top$ if $c = c'$, and $\bot$ otherwise.

Decisional Module-NTRU

- $\mathcal{C}^{\text{MNTRU}}$ samples uniform $c \in \{0,1\}$.
- If $c = 0$:
  - $\mathcal{C}^{\text{MNTRU}}$ samples uniform $h \in R_q^k$.
- If $c = 1$:
  - $\mathcal{C}^{\text{MNTRU}}$ computes $h \in R_q^k$ as in Algorithm 2.
- $\mathcal{C}^{\text{MNTRU}}$ sends h to $\mathcal{A}^{\text{MNTRU}}$.
- $\mathcal{A}^{\text{MNTRU}}$ sends $c' \in \{0,1\}$ to $\mathcal{C}^{\text{MNTRU}}$.
- $\mathcal{C}^{\text{MNTRU}}$ outputs $\top$ if $c = c'$, and $\bot$ otherwise.

The role of the dimension parameter We want to highlight here the ambiguous role of k in the previously defined problems, and the implications for the security of schemes based upon them.

The Module-NTRU problem can be solved by modelling it as a lattice-reduction problem, where the lattice dimension is proportional to nk. Since the hardness of lattice reduction increases with the dimension, the security increases together with k. This fact explains why ModFalcon becomes harder to break when k grows.

However, regarding Ring-LWE, k corresponds to the number of samples given to the adversary. As a consequence, increasing k gives more knowledge to the adversary, and thus the security can only decrease. This explains why an IBE based on Ring-LWE encryption and MNTRU trapdoor becomes no more secure when k increases (contrary to what is stated in [CKKS19]), especially since the best attack is to break the Ring-LWE encryption itself.

5.3 Game 0 - The IND-sID-CPA security model

The security property that we want our IBE scheme to achieve is called "*Indistinguishability of ciphertexts under a selective identity chosen plaintext attack*", and is denoted $\texttt{IND-sID-CPA}$ [GH05]. We describe the associated security game in Figure 2. In that model, the adversary can query the random oracle and the extraction oracle in any order and as many times as needed, before choosing the challenge messages and after having received the challenge ciphertext.

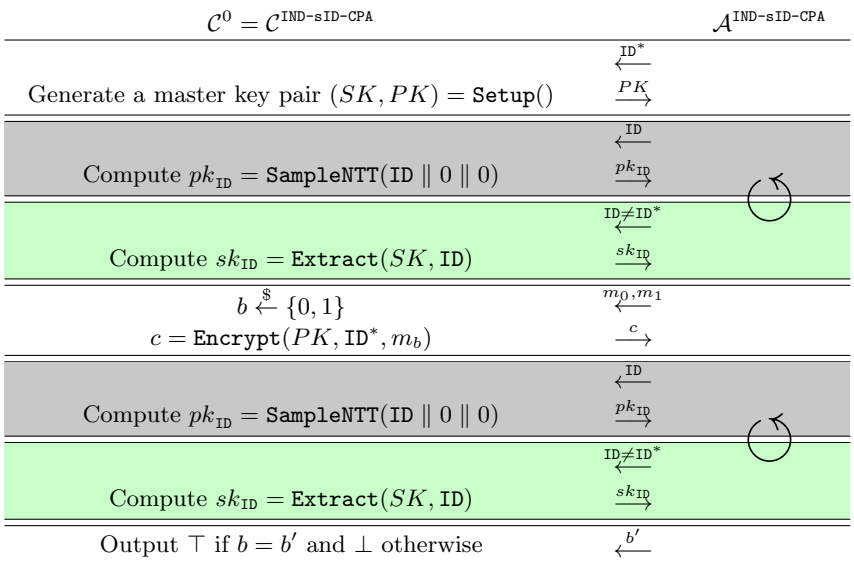

Fig. 2. The IND-sID-CPA security game

5.4 Game 1 - Not using the trapdoor

When asked for the hash of a new identity $\texttt{ID}$, the new challenger $\mathcal{C}^1$ samples small $sk_{\texttt{ID}} \in R_q^k$ and $e \in R_q$ from discrete Gaussian distribution $\mathcal{N}_{R,\sigma}$, and computes the public key $pk_{\texttt{ID}} = PK \cdot sk_{\texttt{ID}} + e$. Finally, $\mathcal{C}^1$ stores $(\texttt{ID}, pk_{\texttt{ID}}, sk_{\texttt{ID}})$ in a hash table $\mathscr{H}$, and sends $pk_{\texttt{ID}}$ to $\mathcal{A}^{\texttt{IND-sID-CPA}}$. When asked for the hash of an identity that is already in $\mathscr{H}$, $\mathcal{C}^1$ answers with the corresponding $pk_{\texttt{ID}}$.

When asked for a secret key associated to an identity $\texttt{ID} \neq \texttt{ID}^*$, $\mathcal{C}^1$ checks if $\texttt{ID}$ is already in $\mathscr{H}$. If not, it executes the random oracle as described above. Then, it sends the corresponding $sk_{\texttt{ID}}$ to $\mathcal{A}^{\texttt{IND-sID-CPA}}$.

The fact that the adversary cannot tell which of $sk_{\texttt{ID}}$ or $pk_{\texttt{ID}}$ has been generated from the other comes from how close the output of the trapdoor sampler is from a discrete Gaussian. In Section 6, we will choose parameters so that the distance between these two distributions is below $2^{-\lambda}$, where λ is the security parameter. Therefore, we can write:

$$\left|\mathrm{Adv}_{\mathcal{G}_1}\left(\mathcal{A}^{\texttt{IND-sID-CPA}}\right) - \mathrm{Adv}_{\mathcal{G}_0}\left(\mathcal{A}^{\texttt{IND-sID-CPA}}\right)\right| \leqslant 2^{-\lambda}$$

5.5 Game 2 - Sampling the master public key at random

This game changes the generation of the master key pair. Since $\mathcal{C}^2$ (like $\mathcal{C}^1$ before) does not need to really extract keys, we can forget the unused SK. Then, the real difference from Game 1 is that PK is generated uniformly at random. The fact that the adversary cannot tell the difference comes from the Decisional-MNTRU problem. More precisely, we have:

$$\left|\mathrm{Adv}_{\mathcal{G}_2}\left(\mathcal{A}^{\texttt{IND-sID-CPA}}\right) - \mathrm{Adv}_{\mathcal{G}_1}\left(\mathcal{A}^{\texttt{IND-sID-CPA}}\right)\right| \leqslant \mathrm{Adv}\left(\mathcal{A}^{\mathrm{MNTRU}}\right)$$

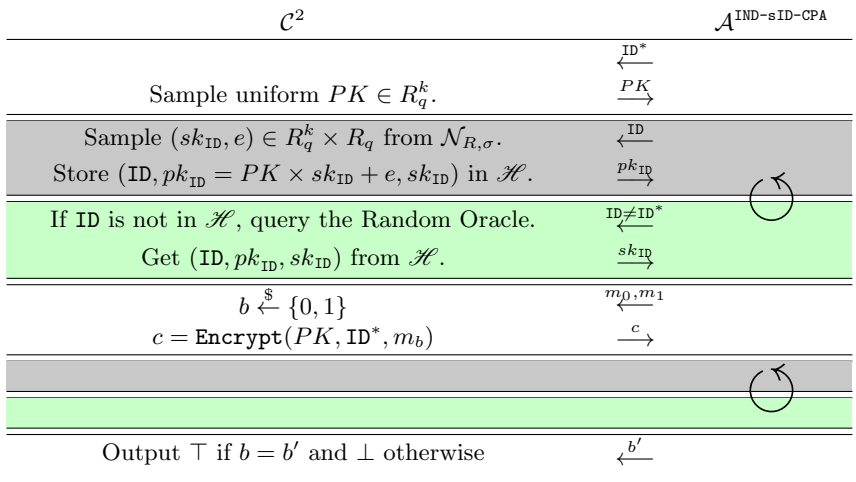

Fig. 3. Game 2 - Uniform master public key

5.6 Game 3 - Reduction from Decisional-RLWE

In Figure 4, we describe how a reduction $\mathcal{R}$ attacking the Decisional-RLWE problem can emulate the challenger $\mathcal{C}^2$ in order to leverage $\mathcal{A}^{\texttt{IND-sID-CPA}}$.

$\mathcal{C}^{\text{RLWE}}$		$\mathcal{R}$		$\mathcal{A}^{\texttt{IND-sID-CPA}}$
$c \xleftarrow{\$} \{0,1\}$	$\xrightarrow{a,b}$	Define $PK = (a_i)_{i\in[\![1,k]\!]}$.	$\xleftarrow{\texttt{ID}^*}$	
		Store $(\texttt{ID}^*, pk_{\texttt{ID}^*} = a_0, 0)$ in $\mathscr{H}$.	$\xrightarrow{PK}$	
		Sample $(sk_{\texttt{ID}}, e) \in R_q^k \times R_q$ from $\mathcal{N}_{R,\sigma}$.	$\xleftarrow{\texttt{ID}}$	
		Store $(\texttt{ID}, pk_{\texttt{ID}} = PK \times sk_{\texttt{ID}} + e, sk_{\texttt{ID}})$ in $\mathscr{H}$.	$\xrightarrow{pk_{\texttt{ID}}}$	
		If ID is not in $\mathscr{H}$, query the Random Oracle.	$\xleftarrow{\texttt{ID}\neq\texttt{ID}^*}$	
		Get $(\texttt{ID}, pk_{\texttt{ID}}, sk_{\texttt{ID}})$ from $\mathscr{H}$.	$\xrightarrow{sk_{\texttt{ID}}}$	
		$\tilde{c} \xleftarrow{\$} \{0,1\}$.	$\xleftarrow{m_0,m_1}$	
		$c^* = (b_1, \cdots, b_k, b_0 + \lfloor q/2 \rceil m_{\tilde{c}})$	$\xrightarrow{c^*}$	
		Sample $(sk_{\texttt{ID}}, e) \in R_q^k \times R_q$ from $\mathcal{N}_{R,\sigma}$.	$\xleftarrow{\texttt{ID}}$	
		Store $(\texttt{ID}, pk_{\texttt{ID}} = PK \times sk_{\texttt{ID}} + e, sk_{\texttt{ID}})$ in $\mathscr{H}$.	$\xrightarrow{pk_{\texttt{ID}}}$	
		If ID is not in $\mathscr{H}$, query the Random Oracle.	$\xleftarrow{\texttt{ID}\neq\texttt{ID}^*}$	
		Get $(\texttt{ID}, pk_{\texttt{ID}}, sk_{\texttt{ID}})$ from $\mathscr{H}$.	$\xrightarrow{sk_{\texttt{ID}}}$	
$c = c' \ ? \top : \bot$	$\xleftarrow{c'}$	Set $c' = 1$ if $\tilde{c} = \tilde{c}'$ and $c' = 0$ otherwise	$\xleftarrow{\tilde{c}'}$	

Fig. 4. Game 3 - Reduction from Decisional-RLWE

The main subtlety lies in the computation of c^*, which should be a valid encryption of the challenge message if $c = 1$. In that case, since $b = as + e'$, we can denote $y = s$, $e_1 = (e'_i)_{i\in[\![1,k]\!]}$ and $e_2 = e'_0$. Then, we indeed have:

$$(u, v) = \left(PK^{\mathrm{T}} \times y + e_1,\ pk_{\texttt{ID}^*}^{\mathrm{T}} \times y + e_2 + \lfloor q/2 \rceil m_{\tilde{c}}\right) = \left((b_i)_{i\in[\![1,k]\!]},\ b_0 + \lfloor q/2 \rceil m_{\tilde{c}}\right)$$

The advantage of $\mathcal{R}$ in solving the decisional version of the Ring-LWE problem can be expressed as follows:

$$\begin{aligned}
\mathrm{Adv}_{\mathrm{RLWE}}(\mathcal{R}) &= 2\,|\mathbb{P}(\top) - 1/2| \\
&= 2\,|\mathbb{P}(c = 0) \times \mathbb{P}(\top \mid c = 0) + \mathbb{P}(c = 1) \times \mathbb{P}(\top \mid c = 1) - 1/2| \\
&= |\mathbb{P}(\top \mid c = 0) + \mathbb{P}(\top \mid c = 1) - 1| \\
&= |\mathbb{P}(\tilde{c} \neq \tilde{c}' \mid c = 0) + \mathbb{P}(\tilde{c} = \tilde{c}' \mid c = 1) - 1|
\end{aligned}$$

If $c = 0$, then $\mathcal{A}^{\texttt{IND-sID-CPA}}$ faces a theoretically-secure one-time pad, so its probability to find the value of the bit $\tilde{c}$ is strictly equal to $1/2$.

If $c = 1$, then $\mathcal{A}^{\texttt{IND-sID-CPA}}$ faces Game 2, so its probability to find $\tilde{c}$ is related to its advantage in that game.

Finally, we obtain the following result:

$$\begin{aligned}\mathrm{Adv}_{\mathrm{RLWE}}(\mathcal{R}) &= |1/2 + \mathbb{P}(\tilde{c} = \tilde{c}' \mid c = 1) - 1| \\ &= 2\,|\mathbb{P}(\tilde{c} = \tilde{c}' \mid c = 1) - 1/2|/2 \\ &= \mathrm{Adv}_{\mathcal{G}_2}\left(\mathcal{A}^{\texttt{IND-sID-CPA}}\right)/2\end{aligned}$$

5.7 The adversary's advantage

Composing previous inequalities, we can prove that:

$$\left|\mathrm{Adv}_{\mathcal{G}_2}\left(\mathcal{A}^{\texttt{IND-sID-CPA}}\right) - \mathrm{Adv}\left(\mathcal{A}^{\texttt{IND-sID-CPA}}\right)\right| \leqslant \mathrm{Adv}\left(\mathcal{A}^{\mathrm{MNTRU}}\right) + 2^{-\lambda}$$

As a consequence, we can finally write:

$$\mathrm{Adv}_{\texttt{IND-sID-CPA}}\left(\mathcal{A}^{\texttt{IND-sID-CPA}}\right) \leqslant 2\ \mathrm{Adv}_{\mathrm{RLWE}}(\mathcal{R}) + \mathrm{Adv}_{\mathrm{MNTRU}}\left(\mathcal{A}^{\mathrm{MNTRU}}\right) + 2^{-\lambda}$$

So, the existence of an efficient adversary against the `IND-sID-CPA` security of our scheme would imply the existence of an efficient adversary against the decisional variant of Ring-LWE or Module-NTRU. Stated differently, our scheme is `IND-sID-CPA`-secure if both of these problems are hard, which is currently believed to hold, even if we had a quantum computer.

6 Concrete parameters

As in [DLP14], we focus on the `CPA`-secure IBE for simplicity and fair comparison. To choose concrete parameters, we must take into account some criteria. Some of them are common to all cryptosystems (security, data sizes, running time), some are specific to lattice-based cryptosystems (NTT, failure probability), and some are specific to our scheme, or our target use-case (code-reusability).

6.1 The security of the scheme

The first criterion is of course that the security must be sufficient (we target here a security level of 128 bits, comparable to ML-KEM-512). To check that, we leverage the reduction from Ring-LWE and Module-NTRU from Section 5, and LatticeEstimator, a tool that evaluates the security of an instance of some problem over lattices [APS15, commit 6b25d9d]. Indeed, since we know that our scheme is at least as secure as the corresponding instances of Ring-LWE and Module-NTRU, we can use LatticeEstimator to evaluate the security of these instances, to get a lower bound on the security of our scheme. We target here a security level of 128 bits (comparable to ML-KEM-512), but we can apply the same methodology to choose parameters for any security level.

6.2 The reusability of the key functions

Modular arithmetic is present all over our scheme, and it can be a crucial point for the efficiency of the implementation. The choice of the modulus q is thus important, all the more since some operations could be protected or optimized specifically for some moduli. Following our goal of reusing the existing optimizations and protections, we want to use a standard modulus q, namely either 3 329 (from ML-KEM) or 8 380 417 (from ML-DSA).

The Number Theoretic Transform, or NTT, is also a core operation in our scheme. It uses a primitive m-th root ζ of -1 modulo q to quickly split a polynomial in $R_q = \mathbb{Z}_q[X]/_{\langle X^n+1\rangle}$ into a tuple of m lower-degree polynomials in $T_q = \prod \mathbb{Z}_q[X]/_{\langle X^{n/m}-\zeta^\alpha\rangle}$. To use NTT, there must exist such a ζ in $\mathbb{Z}_q$, which only depends on n and q. More precisely, we can only find a primitive m-th root of -1 (i.e. a primitive $2m$-th root of 1) for m up to 128 modulo 3 329, and for m up to 4 096 modulo 8 380 417. Since both ML-KEM and ML-DSA use $n = 256$, this means we are able to split an element of $R_{8\,380\,417}$ into n scalars, but we can only split an element of $R_{3\,329}$ into $n/2$ degree-1 polynomials.

Since the main use of NTT is to quickly multiply polynomials, its characteristics mainly influence the `BaseCaseMultiply` function, used to multiply two polynomials in NTT domain, of degree n/m. In ML-DSA, this function performs $n = 256$ products of two degree-0 polynomials, and in ML-KEM, it performs $n/2 = 128$ products of two degree-1 polynomials. In order to reuse that code, we must find parameters that allow splitting to degree at most 1. Stated differently, we need that $n/m \in \{1, 2\}$, which means that we need $n \leqslant 256$ with $q = 3\,329$, or $n \leqslant 8\,192$ with $q = 8\,380\,417$.

Moreover, NTT seems to be a great target for physical attacks, as illustrated in [Car24]. Protections are thus being developed for ML-KEM and ML-DSA, and we want to be able to reuse them in our IBE, even if they protect the `BaseCaseMultiply` function.

6.3 The failure probability

Last but not least, we must check that the failure probability is negligible. While there is no universal definition of "negligible", we use the same target as ML-KEM, which is "under 2^{-128}". The failure probability estimation notably allows to define the compression level of the ciphertext. Indeed, ML-KEM allows to reduce each coefficient (which is in $\mathbb{Z}_q$) of u (resp. v) to d_u (resp. d_v) bits.

First, we must validate that our choice of n and q can achieve negligible failure probability. To do that, we estimate that probability when there is no ciphertext compression, i.e. with $d_u = d_v = \lfloor \log_2(q) + 1 \rfloor$. If it is indeed negligible, then our choice of parameters is compatible with negligible failure probability. We can then enable some compression of the ciphertext, by (basically) dropping some low-order bits of its coefficients. To parameterize that, we follow the same approach as in previous NTRU-based IBE schemes, which only keep 3 bits of each coefficient of the vector v [DLP14,CKKS19].

6.4 The final choice of parameters

The degree n and modulus q: As explained before, we want q to be either 3 329 or 8 380 417, and n to be a power of 2 such that the NTT is compatible with an existing one. We thus summarize some possible combinations in Table 1.

Table 1. Choosing parameters to reuse the Number Theoretic Transform

n	q	$\mathbb{Z}_q$ arithmetic	NTT domain T_q	`BaseCaseMultiply`	Security
256	3 329	ML-KEM	$\prod_{i=0}^{127} \mathbb{Z}_q[X]/_{\langle X^2-17^{2i+1}\rangle}$	ML-KEM	69 bits
256	8 380 417	ML-DSA	$\prod_{i=0}^{255} \mathbb{Z}_q[X]/_{\langle X-1753^{2i+1}\rangle}$	ML-DSA	40 bits
1 024	8 380 417	ML-DSA	$\prod_{i=0}^{1023} \mathbb{Z}_q[X]/_{\langle X-1306^{2i+1}\rangle}$	ML-DSA	140 bits

First, we can see that the ideal case, with the standardized degree $n = 256$, does not provide enough security in our scheme, with any of our two modulus of interest, so we must increase n. However, as explained before, using $q = 3\,329$ would require a new implementation of `BaseCaseMultiply`. Therefore, we will set $q = 8\,380\,417$, which requires to increase n to 1 024 for security.

Fortunately, the existence of the primitive 2 048-th root of unity $\zeta = 1\,306$ in $\mathbb{Z}_q$ allows to reuse the implementation of `BaseCaseMultiply` from ML-DSA, with $n/m = 1$. Interestingly, we can also set $\zeta = 10\,730$ if we prefer to reuse the implementation of `BaseCaseMultiply` from ML-KEM, with $n/m = 2$.

The standard deviation σ: Now that we have chosen n and q such that the encryption itself is secure, we must also consider the security of another important feature: the trapdoor sampling. Indeed, the output distribution must be wide enough so that the discreteness of the secret lattice remains hidden to the adversary. To do that, a classical approach is to prove that the output distribution differs negligibly from a Gaussian distribution that is independent of the trapdoor [GPV08]. For ModFalcon, this is proved using the Rényi divergence between the two distributions, which requires [CPS+20]:

$$\sigma \geqslant \frac{\mathrm{GS_SLACK}_k \times q^{1/(k+1)}}{\pi} \times \sqrt{\frac{\log\left(2(k+1)n(1+4\sqrt{256\times 2^{64}})\right)}{2}}$$

In the meantime, the larger the σ, the larger the failure probability, so we will use the lowest integer satisfying this security condition. For now, we have fixed n and q but not k, so we will consider σ as a function of k until this parameter is assigned a value. In practice, we set σ to the ceiling of the above right term.

The encryption noise (η_1 and η_2): To stay as close as possible to the encryption algorithm of ML-KEM, we use Centered Binomial Distributions, with parameters η_1 and η_2. We ran experiments with LatticeEstimator, and found out that these parameters don't seem to have a significant impact on security. Moreover, these parameters influence the failure probability of the scheme. So, we decided to take the values from ML-KEM-512: $(\eta_1, \eta_2) = (3, 2)$.

The dimension k over polynomial rings: An unfortunate relation is that k has opposite impacts on sizes and failure probability. Indeed, σ mainly depends on $q^{1/(k+1)}$, so increasing k leads to decrease the failure probability. At the same time, since it represents the number of polynomials in many objects, increasing k makes the keys and ciphertext bigger. Therefore, we would want to use $k = 1$ to minimize the size of the data (which would correspond to an NTRU trapdoor, as used in [DLP14]). However, the decryption failure probability of our scheme would then be too high, even without compressing the ciphertext. So, we set $k = 2$, which reaches negligible failure probability with $\sigma = 325$.

The ciphertext compression (d_u and d_v): At this point, the failure probability only depends on how much we compress the ciphertext, which is parameterized by d_u and d_v. As explained previously, we used the same technique as in prior NTRU-based IBE schemes, keeping only the 3 highest-order bits of the coefficients of v, which corresponds to $(d_u, d_v) = (23, 3)$. We can also note that this is consistent with what is performed in ML-KEM, where each coefficient of v is also compressed into only a few bits.

The compression of the master secret key: At the end of `Setup`, each coefficient of each polynomial in the mathematical master secret key (f, g, F, G) is encoded into d_{SK} bits to form the computer-friendly byte array SK, the algorithm restarting if any coefficient is too large. So, in order to reduce the size of SK, we want d_{SK} to be as small as possible. However, it must remain high enough so that the algorithm does not restart too often. Regarding f and g, we can apply the Chernoff bound to their known Gaussian distributions to ensure that all their (signed) coefficients will fit in 8 bits with probability at least $1 - 2^{-406}$. For F and G, we generated 100 master secret keys, and observed that all the absolute values of their 204 800 coefficients fit in 16 bits. We thus use $d_{SK} = 17$, but we could also decrease the size of SK by using $d_{SK} = 8$ for f and g, and $d_{SK} = 17$ for F and G.

Finally, we summarize in Table 2 the parameters that we have chosen to instantiate our scheme, the Gram-Schmidt slack resulting of these choices, and the security level achieved by this parameter set.

Table 2. The parameters of our scheme

n	q	k	η_1	η_2	σ	d_{SK}	d_u	d_v	GS_SLACK$_k$	Security
1 024	8 380 417	2	3	2	325	17	23	3	1.2	140 bits

6.5 Data sizes

We summarize in Table 3 the sizes of the keys and ciphertexts in our IBE scheme. We compare them to the sizes resulting of the parameter set of [DLP14] that was originally announced to provide a security level of 192 bits, since LatticeEstimator currently estimates its level of provable security to about 118 bits. We don't compare to [CKKS19], as we estimated the provable security level of its only parameter set to about 82 bits. Since the master secret key of [DLP14] contains integers, we were not able to give it a precise size, highlighting one of the benefits of our encoding to byte arrays.

We can see that our IBE, while not beating the [DLP14] scheme in terms of sizes, is still in the same order of magnitude (while being more secure), even if it was not the original objective. Therefore, the simplicity to implement, optimize and protect our scheme in a device that already embeds standardized lattice-based schemes, possibly with some existing hardware accelerators and countermeasures, can be worth the extra size of the keys and ciphertexts.

Table 3. Sizes of the keys and ciphertexts

Element	Size in our scheme (bytes)	[DLP14]
Master secret key	$(k^2+k+2) \times n \times d_{SK} \,/\, 8 = 17\,408$	Undefined
Master public key	$k \times n \times \lfloor \log_2(q) + 1 \rfloor \,/\, 8 = 5\,888$	3 456
User secret key	$k \times n \times \lfloor \log_2(q) + 1 \rfloor \,/\, 8 = 5\,888$	3 456
User public key (`ID`)	32	32
Ciphertext	$(k \times n \times d_u + n \times d_v) \,/\, 8 = 6\,272$	3 750

Even if Identity-Based Encryption has always been bigger than usual Public-Key Encryption, its properties can be interesting in specific use-cases (see Section 1). However, it is important that its cost remains acceptable in practice. That is why we compare the communication costs of our IBE to the one of ML-KEM, even if these are different kind of schemes enabling different features. For that comparison, we must consider a complete establishment of a shared secret. Indeed, while our scheme allows to directly send a ciphertext, ML-KEM requires an additional certificate. In the simple case where a (Post-Quantum) certificate (with 128-bit security) contains an identity, a public key for ML-KEM and a ML-DSA signature from the Certification Authority, then its size is $32 + 800 + 2\,420 = 3\,252$ bytes, and then the total communication cost with ML-KEM is $3\,252 + 768 = 4\,020$ bytes. This means that our scheme is again in the same order of magnitude, with a total communication only containing one ciphertext, of size 6 272 bytes. The benefits of Identity-Based Encryption can be worth this additional communication cost, in application contexts which leverage some of its special properties.

7 Future perspectives

We created a clean Python implementation of our IBE to illustrate its ability to reuse blocks from standardized schemes and to give a rough idea about its running time. Deriving a C implementation from reference codes of standards would further demonstrate our IBE's practicality, and give more accurate runtime.

Like prior IBEs, we estimated the failure probability via theoretical approximations, and set up ciphertext compression relying on existing NTRU-based IBE schemes. At the opposite, Kyber's authors estimated the failure probability via a concrete computation, using a program they designed [SAB+22,CRY21]. However, the time taken by that program grows quadratically with the modulus, making it hard to use with our parameters. So, the analysis could be refined by some optimized version of that tool, which could reduce the ciphertext size.

8 Conclusion

We built a post-quantum IBE scheme over the standardized ML-KEM and ML-DSA, and the variant ModFalcon of Falcon, under standardization.

Any method that can assess the security of one of our IBE, [DLP14] or [CKKS19] can apply to the others, so the schemes all satisfy the same security properties. However, while their parameters were analyzed against the best attack found by their authors, ours rely on a tight reduction from Ring-LWE and Module-NTRU (from Section 5). This allows our scheme to have high security guarantees, in addition to being similar to well-trusted standardized schemes. For comparison, via our evaluation technique based on LatticeEstimator, our scheme reaches a security level of 140 bits, while the highest parameter set of [DLP14] reaches 118 bits, and the instantiation in [CKKS19] only reaches 82 bits.

Our scheme can easily be instantiated with concrete parameters according to some targets, in terms of security and decryption failure probability. Targeting security above 128 bits and failure probability below 2^{-128}, our scheme is comparable in sizes to [DLP14], while being more secure and easier to deploy.

While previous schemes leave some blank space about how to concretely implement them (how to represent an integer, what precise parameters should be used), our paper gives a detailed description of the algorithms and their implementations (from Section 4) and a precise analysis of our scheme's parameters (from Section 6). All our algorithms input and output byte-strings, making both concrete implementation and sizes estimation easy.

If we want to use any of the previously existing lattice-based IBE, the chosen scheme should be implemented, optimized and protected from scratch, requiring lot of time and work, and thus being costly. Conversely, we built our scheme so that it can reuse existing code from standardized schemes, making it easy to implement and allowing it to benefit from any optimization or protection that can be applied to these well-studied standardized schemes.

Finally, in order to check that everything works as announced, that the scheme indeed inherits much of its code from standardized schemes, to mea-

sure the running time, or for any other purpose, we provide a clean open-source implementation of our scheme in Python [Cam25].

References

AKG+07. N. Asokan, Kari Kostiainen, Philip Ginzboorg, Jörg Ott, and Cheng Luo. Applicability of identity-based cryptography for disruption-tolerant networking. In *Proceedings of the 1st International MobiSys Workshop on Mobile Opportunistic Networking*, MobiOpp '07, pages 52–56, New York, NY, USA, 2007. Association for Computing Machinery.

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, October 2015.

BF01. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Berlin, Heidelberg, August 2001.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

Cam25. Julien Cam. A python implementation of the current IBE scheme, 2025. Available at https://github.com/JCamCrypto/ID-ML-KEM_MNTRU_Python.

Car24. Rafael Carrera Rodriguez. *Study of the resistance to side-channel attacks of cryptographic primitives based on lattices for post-quantum cryptography.* PhD thesis, Université de Montpellier, 2024.

CGM19. Yilei Chen, Nicholas Genise, and Pratyay Mukherjee. Approximate trapdoors for lattices and smaller hash-and-sign signatures. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 3–32. Springer, Cham, December 2019.

CKKS19. Jung Hee Cheon, Duhyeong Kim, Taechan Kim, and Yongha Son. A new trapdoor over module-NTRU lattice and its application to ID-based encryption. Cryptology ePrint Archive, Report 2019/1468, 2019.

Coc01. Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of *LNCS*, pages 360–363. Springer, Berlin, Heidelberg, December 2001.

CPS+20. Chitchanok Chuengsatiansup, Thomas Prest, Damien Stehlé, Alexandre Wallet, and Keita Xagawa. ModFalcon: Compact signatures based on module-NTRU lattices. In Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese, editors, *ASIACCS 20*, pages 853–866. ACM Press, October 2020.

CRY21. CRYSTALS team. Security estimation scripts for Kyber and Dilithium, 2021. Available at https://github.com/pq-crystals/security-estimates.

DLP14. Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 22–41. Springer, Berlin, Heidelberg, December 2014.

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*,

volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999.

GH05. David Galindo and Ichiro Hasuo. Security notions for identity based encryption. Cryptology ePrint Archive, Report 2005/253, 2005.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

Gro12. Michael Groves. MIKEY-SAKKE: Sakai-Kasahara Key Encryption in Multimedia Internet KEYing (MIKEY). RFC 6509, February 2012.

IPR23. Malika Izabachène, Lucas Prabel, and Adeline Roux-Langlois. Identity-based encryption from lattices using approximate trapdoors. In Leonie Simpson and Mir Ali Rezazadeh Baee, editors, *ACISP 23*, volume 13915 of *LNCS*, pages 270–290. Springer, Cham, July 2023.

LDK+22. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022.

LPR12. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, 2012.

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Berlin, Heidelberg, April 2012.

Nat16. National Cyber Security Center. The development of mikey-sakke, 2016.

Nat24a. National Institute of Standards and Technology. Module-Lattice-Based Digital Signature Standard. (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS) NIST FIPS 204 , 2024.

Nat24b. National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS) NIST FIPS 203 , 2024.

PFH+22. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022.

SAB+22. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022.

Sha84. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Berlin, Heidelberg, August 1984.

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

TWZL08. Chiu C. Tan, Haodong Wang, Sheng Zhong, and Qun Li. Body sensor network security: an identity-based cryptography approach. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, pages 148–153, New York, NY, USA, 2008. Association for Computing Machinery.