An efficient Failure Probability Estimator for LWE-based encryption, and application to IBE

Julien CAM[1,2]

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

Abstract. In recent years, lattice-based cryptography has proven to be a source of post-quantum cryptosystems. In particular, the blueprint of Learning With Errors (LWE) encryption enabled the creation of Kyber, chosen by NIST to be standardized as ML-KEM. Because of the structure of LWE encryption, the decryption of a ciphertext can give an incorrect plaintext. This event (a decryption failure) happens randomly with some probability. Since this is a problem for both functionality and security, we must ensure that this failure probability is cryptographically negligible. Usually, this is assessed with theoretical approximations and upper-bounds. The authors of Kyber used a Python program to compute that probability more precisely, enabling to fine-tune the compression of the ciphertext, hence reducing its size. However, because of its performances, this program has been limited to Kyber, and all the other schemes are still studied theoretically. In this paper, we design a more efficient Failure Probability Estimator, that we implement in low-level languages (C and CUDA C++), to reach higher parameters used in real schemes based on structured LWE. We illustrate some benefits of our tool with two Identity-Based Encryption schemes. For the first scheme, our tool allows to reduce the size of the ciphertext by about 20%. For the second one, we reduce the failure probability (thus improving security) without impacting the scheme (key size, ciphertext size, etc.).

Keywords: Post-Quantum Cryptography · LWE-based Encryption · ML-KEM · Decryption failure probability · Identity-Based Encryption.

1 Introduction

1.1 LWE-based encryption

In 2005, Regev proposed an encryption scheme based on a new lattice-related problem, LWE. This work has served as a foundation for many lattice-based encryption schemes, which became the main solution to replace number-theoretic-based schemes in order to resist to attacks that could leverage quantum computers in the future. Structured variants (Ring-LWE, Module-LWE) allowed to reduce the sizes and improve the efficiency, enabling highly practical schemes like Kyber, now standardized by NIST as ML-KEM [SAB+22,Nat24].

Since these structured variants are the main source of practical schemes, we will focus on them in the rest of the paper. More precisely, since Ring-LWE can be seen as a particular case of Module-LWE (with module dimension 1), we will focus on MLWE in the rest of this paper. Assuming that Bob wants to send a message m to Alice, the MLWE encryption scheme works as follows:

- We define the polynomial ring $R_q = \mathbb{Z}_q[X]/_{(X^n+1)}$.
- (`KeyGen`) Alice generates her keys:
  - Sample a uniform matrix $A \in R_q^{k \times k}$, and small $s \in R_q^k$ and $e_k \in R_q^k$.
  - Compute $t = A \times s + e_k$.
- Alice sends her public key (A, t) to Bob.
- (`Encrypt`) Bob hides the message $m \in \{0,1\}^n$ into a ciphertext:
  - Sample small $y \in R_q^k$, $e_1 \in R_q^k$ and $e_2 \in R_q$.
  - Compute $u = A^{\mathrm{T}} \times y + e_1$ and $v = t^{\mathrm{T}} \times y + e_2 + \lfloor q/2 \rceil m$.
- Bob sends the ciphertext $c = (u, v)$ to Alice.
- (`Decrypt`) Alice recovers the message from the ciphertext:
  - Compute $w = v - s^{\mathrm{T}}u$, and $m' = \left\lfloor \dfrac{w}{\lfloor q/2 \rceil} \right\rceil$.

During the encryption algorithm, we can first observe an encoding step, when $m \in \{0,1\}^n$ is converted into a polynomial $\lfloor q/2 \rceil m$ with coefficients in $\{0, \lfloor q/2 \rceil\}$. The security of the scheme is ensured by the addition of small noise (e_k, e_1 and e_2), which the decryption algorithm must remove to recover the original encoding of the message. We illustrate this process in Figure 1.

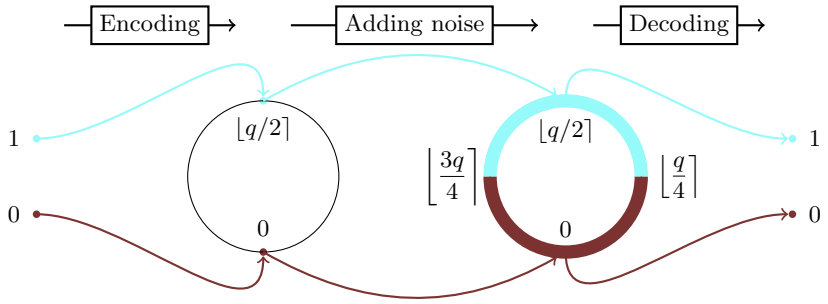


Fig. 1: Adding and removing noise modulo q

1.2 Decryption failure

Because of the structure of LWE-based encryption, relying on the addition of small noise during encryption that must be removed during decryption, the question of *decryption correctness* is a critical point. Indeed, the small noise terms get added and multiplied during the operation of the scheme, so the level of noise can increase throughout the computation.

During MLWE decryption, Alice computes $w = \mu + E$, where $\mu = \lfloor q/2 \rceil m$ is the meaningful encoding of the message, and $E \in R_q$ is the error polynomial that Alice wants to remove. To do that, she performs a modular rounding (illustrated as the decoding phase in Figure 1):

$$\forall\, i \in [\![0, n-1]\!], \quad w_i \mapsto \begin{cases} 0 & \text{if } |w_i| < q/4 \\ 1 & \text{otherwise} \end{cases}$$

So that Alice recovers the expected message m at the end, all the coefficients of E must be below $\lfloor q/4 \rfloor$, so that they are all rounded to 0. If $\|E\|_\infty \geqslant \lfloor q/4 \rfloor$, a noisy encoding w_i of a bit m_i will be interpreted as a noisy encoding of $1 - m_i$, leading to (at least) one bit-flip in the recovered message m'. That case where $m \neq m'$ is called a *decryption failure*.

In the MLWE encryption scheme, we have:

$$E = e_k^{\mathrm{T}} \times y - s^{\mathrm{T}} \times e_1 + e_2$$

Therefore, E is a function of e_1, e_2, e_k, s and y, which are sampled at random during the execution (this holds in general for LWE encryption). As a consequence, a decryption failure is a random event, which can occur with some probability. We call *failure probability* the probability of a decryption failure, when e_1, e_2, e_k, s and y are sampled at random from their respective distributions, which depend on the scheme we consider.

Since E is a function of the secret (s, e_k) and of the noise (e_1, e_2, y) controlled by the adversary, being able to create a decryption failure creates an attack path for an adversary. For that reason, any LWE-based encryption scheme must ensure that an attacker cannot trigger a decryption failure. Ideally, we would want to prove that all the coefficients of E are necessarily small enough, but it could be impossible (for example if the error terms are not trivially bounded), or not desirable for performances. Indeed, a "cryptographically negligible" failure probability (like 2^{-128}) is enough to prevent an attack, and this relaxation can allow smaller parameters.

1.3 The case of Kyber/ML-KEM

Compared to the MLWE encryption scheme we presented earlier, Kyber contains optimizations like ciphertext compression. Indeed, in order to reduce the size of the ciphertext, Bob does not directly send (u, v), but rather an approximation $c = (\texttt{Compress}_{d_u}(u), \texttt{Compress}_{d_v}(v))$, where d_u and d_v are fixed parameters. Then, Alice computes $u' = \texttt{Decompress}_{d_u}(c_1)$ and $v' = \texttt{Decompress}_{d_v}(c_2)$, to compute $w = v' - s^{\mathrm{T}}u'$ before decoding the result to binary [Nat24].

In practice, this compression consists in representing each coefficient of the input polynomial on a fixed number of bits (d_u bits for the coefficients of u, and d_v for the coefficients of v). We model this by introducing new error terms, $e_u \in R_q^k$ and $e_v \in R_q$, such that: $u' = u + e_u$ and $v' = v + e_v$. We define $\mathcal{D}_u$ (resp. $\mathcal{D}_v$) as the distribution of the coefficients of e_u (resp. e_v) over uniformly random choice of $u \in R_q^k$ (resp. $v \in R_q$).

In order to efficiently sample the small terms e_1, e_2, e_k, s and y, ML-KEM uses Centered Binomial Distributions [SAB+22,Nat24]. The scheme also leverages the Number Theoretic Transform, but its operation is transparent to this paper's considerations, so we omit it for clarity. Finally, we can give a more accurate representation of the K-PKE scheme underlying ML-KEM:

- $\texttt{KeyGen}()$
  - Sample the coefficients of $A \in R_q^{k \times k}$ from uniform.
  - Sample the coefficients of $s \in R_q^k$ from CBD_{η_1}.
  - Sample the coefficients of $e_k \in R_q^k$ from CBD_{η_1}.
  - Compute $t = A \times s + e_k$.
- $\texttt{Encrypt}((A, t), m)$
  - Sample the coefficients of $y \in R_q^k$ from CBD_{η_1}.
  - Sample the coefficients of $e_1 \in R_q^k$ from CBD_{η_2}.
  - Sample the coefficients of $e_2 \in R_q$ from CBD_{η_2}.
  - Compute $u = A^{\text{T}} \times y + e_1$ and $c_1 = \texttt{Compress}_{d_u}(u)$.
  - Compute $v = t^{\text{T}} \times y + e_2 + \lfloor q/2 \rceil m$ and $c_2 = \texttt{Compress}_{d_v}(v)$.
- $\texttt{Decrypt}(s, c)$
  - Compute $u' = \texttt{Decompress}_{d_u}(c_1) = A^{\text{T}} \times y + e_1 + e_u$.
  - Compute $v' = \texttt{Decompress}_{d_v}(c_2) = t^{\text{T}} \times y + e_2 + \lfloor q/2 \rceil m + e_v$.
  - Compute $w = v' - s^{\text{T}} u' = \lfloor q/2 \rceil m + e_k^{\text{T}} \times y - s^{\text{T}}(e_1 + e_u) + e_2 + e_v$.
  - Compute $m' = \texttt{Compress}_1(w)$.

From that description, we see that the final error term that should be rounded to 0 for correct decryption is:

$$E = e_k^{\text{T}} \times y - s^{\text{T}} \times (e_1 + e_u) + e_2 + e_v$$

1.4 Performing operations on distributions

To find the best parameters for their scheme, the authors of Kyber computed its actual failure probability, by allowing to perform some operations on probability distributions over $\mathbb{Z}_q$ [SAB+22,CRY21]. This allows to compute the distribution of each coefficient of E, in order to deduce the probability that one of them is large enough to trigger a failure.

Definition 1 (Sum). *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two probability distributions over $\mathbb{Z}_q$. We define their sum $\mathcal{D}_1 + \mathcal{D}_2$ as the distribution such that, if $X \sim \mathcal{D}_1$ and $Y \sim \mathcal{D}_2$ are two independent random variables in $\mathbb{Z}_q$, then their sum $Z = X + Y$ has distribution $\mathcal{D}_1 + \mathcal{D}_2$.*

Definition 2 (Product). *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two probability distributions over $\mathbb{Z}_q$. We define their product $\mathcal{D}_1 \times \mathcal{D}_2$ as the distribution such that, if $X \sim \mathcal{D}_1$ and $Y \sim \mathcal{D}_2$ are two independent random variables in $\mathbb{Z}_q$, then their product $Z = X \times Y$ has distribution $\mathcal{D}_1 \times \mathcal{D}_2$.*

Definition 3 (Scalar product). *Let $\mathcal{D}$ be a probability distributions over $\mathbb{Z}_q$, and $\alpha \in \mathbb{N}^*$. We define the scalar product $\alpha \cdot \mathcal{D}$ as the sum of α copies of $\mathcal{D}$.*

1.5 The estimator for Kyber's failure probability

The authors of Kyber implemented a Python program to perform the previously defined operations on probability distributions over $\mathbb{Z}_q$ [SAB+22,CRY21]. More precisely, they used Algorithm 1 and Algorithm 2 to perform these operations, representing a distribution $\mathcal{D}$ over $\mathbb{Z}_q$ by a list D such that:

$$\forall\, x \in [\![0, q-1]\!], \quad D[x] = \mathbb{P}_{\mathcal{D}}(\{x\})$$

Algorithm 1 `AddDistributions`

Input: A list D_1 representing a distribution $\mathcal{D}_1$.
Input: A list D_2 representing a distribution $\mathcal{D}_2$.
Output: A list D_3 representing the sum $\mathcal{D}_3 = \mathcal{D}_1 + \mathcal{D}_2$.

```
D3 = [0, ··· , 0]                                   ▷ q elements
for x = 0 to q − 1 do
    for y = 0 to q − 1 do
        z = x + y  mod q
        D3[z] = D3[z] + D1[x] × D2[y]
```

Algorithm 2 `MultiplyDistributions`

Input: A list D_1 representing a distribution $\mathcal{D}_1$.
Input: A list D_2 representing a distribution $\mathcal{D}_2$.
Output: A list D_3 representing the product $\mathcal{D}_3 = \mathcal{D}_1 \times \mathcal{D}_2$.

```
D3 = [0, ··· , 0]                                   ▷ q elements
for x = 0 to q − 1 do
    for y = 0 to q − 1 do
        z = x × y  mod q
        D3[z] = D3[z] + D1[x] × D2[y]
```

To compute the distribution of one coefficient of E, we first have to compute its expression from the error terms that are sampled from known distributions. We already computed the following expression:

$$E = \sum_{i=0}^{k-1} e_{k,i} \times y_i - \sum_{i=0}^{k-1} s_i \times (e_{1,i} + e_{u,i}) + e_2 + e_v$$

Now, we consider two polynomials A and B in R_q, such that all the coefficients $(a_i)_{i \in [\![0,n-1]\!]}$ of A have distribution $\mathcal{D}_A$, all the coefficients $(b_i)_{i \in [\![0,n-1]\!]}$ of B have distribution $\mathcal{D}_B$, and all of these coefficients are pairwise independent. The product $C = A \times B$ is given by:

$$C = \sum_{i=0}^{n-1} \left[\sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{n-1} a_j b_{i+n-j} \right] X^i$$

Therefore, each coefficient of C is a sum of n products of one coefficient of A multiplied by a coefficient of B. Each coefficient of A and B appears exactly once in the expression of each coefficient. So, all the coefficients of C have the same distribution ($\mathcal{D}_C = n \times \mathcal{D}_A \times \mathcal{D}_B$).

Finally, all the coefficients E_i of E have the following distribution:

$$\mathcal{D} = (k \times n) \cdot [\mathrm{CBD}_{\eta_1} \times \mathrm{CBD}_{\eta_1}] - (k \times n) \cdot [\mathrm{CBD}_{\eta_1} \times (\mathrm{CBD}_{\eta_2} + \mathcal{D}_u)] + \mathrm{CBD}_{\eta_2} + \mathcal{D}_v$$

So, their Python program allowed the authors of Kyber to compute that distribution $\mathcal{D}$, and then to compute $P = \mathbb{P}(|E_0| > \lfloor q/4 \rfloor) = \cdots = \mathbb{P}(|E_{n-1}| > \lfloor q/4 \rfloor)$. Finally, since they share the same basic terms, the coefficients of E are not pairwise independent, so they could only upper-bound the final failure probability of the complete scheme. Using the union bound $\mathbb{P}(A \cup B) \leqslant \mathbb{P}(A) + \mathbb{P}(B)$, we can guarantee that this failure probability satisfies:

$$\mathbb{P}(\text{Failure}) = \mathbb{P}(\|E\|_\infty > \lfloor q/4 \rfloor) \leqslant n \times P$$

1.6 Using the estimator to find the best parameters

Many parameters of the Kyber scheme must be carefully chosen in order to satisfy the security requirement, like n, q, k, η_1 and η_2. These parameters influence the "mathematical scheme", without the compression optimization, which is parameterized by d_u and d_v.

The parameters d_u and d_v tell how much the ciphertext will be compressed, by setting how many bits will be used to represent each coefficient. As such, the larger they are, the bigger the ciphertext. However, we cannot decrease these parameters too much, because the smaller they are, the higher the failure probability. So, the goal is to find the best values, which minimize the ciphertext size while keeping a negligible failure probability.

While there is no universally recognized definition of "negligible", the authors of Kyber targeted "below 2^{-128}", which guarantees a very low failure probability at the limit of 2^{64} decryption queries defined by NIST. For example, considering that limit, there is a chance of 2^{-100} of getting one failure in Kyber768 without any particular effort by the attacker [SAB+22].

Some works have explored optimizations to search more efficiently for a failing ciphertext [DVV18,DGJ+19,BS20,DRV20]. In particular, "directional failure boosting" reduces the cost of finding multiple such ciphertexts once we have found one, reducing the cost of a full attack to barely more than the cost of triggering the first failure [SAB+22]. As a consequence, a scheme should stay on the safe side and it should be hard to trigger even one failure.

Once they defined this target, they explored the possible compression parameters to find the pairs that keep the failure probability below it. Then, they can find which pair allows for the lowest ciphertext size, while satisfying security requirements about negligible failure probability [SAB+22,Nat24].

1.7 Contributions

In this paper, we design an optimized Failure Probability Estimator, which enables the idea of the authors of Kyber to reach new use-cases. Moreover, we demonstrate the ability of our tool to improve two Identity-Based Encryption schemes, including the most widely recognized, [DLP14].

The program from the authors of Kyber allows to precisely compute the failure probability of their scheme. Moreover, it can easily be adapted to other MLWE-based cryptosystems, since they are all concerned with decryption correctness. Therefore, their estimator could help to compute the failure probability of any MLWE-based scheme [CRY21].

However, although that program is efficient on the parameters of Kyber, it scales badly, preventing from using it for schemes with larger parameters. Kyber is one of the lattice-based schemes that enjoys the smallest parameters ever created. Thanks to that, the authors of Kyber did not need a very efficient tool to compute its failure probability.

Unfortunately, we can see that both Algorithm 1 and Algorithm 2 perform about q^2 iterations to compute the result of the corresponding operation. In reality, we presented a simplified version of the algorithms used by the authors, which does not include an optimization described in Section 2.3. However, even if it is less obvious with it, the conclusion is still the same: the number of iterations grows quadratically with the modulus q.

Because of these drawbacks, this estimator was only used for Kyber, the rest of the LWE-based schemes still being studied via theoretical (potentially loose) approximations. Indeed, fields like Identity-Based Encryption, or Fully Homomorphic Encryption, use larger moduli and wider distributions, which makes the original Python estimator from the authors of Kyber completely inefficient.

On the theoretical side, we leveraged the symmetry of the distributions to reduce the memory footprint, we leveraged their small support to skip unnecessary computations, and our major improvement is the ability to parallelize, which allows to greatly reduce the running time of the estimator. To allow that, we change the way our estimator computes the operations, since the method used by the authors of Kyber is not compatible with parallelization.

On the implementation side, we implemented our new estimator both in C and in CUDA C++. These low level languages inherently improve performances compared to the original estimator implemented in Python. Moreover, the CUDA implementation enables to exploit the parallelization power of GPUs.

With these efficient implementations of our optimized Failure Probability Estimator, we are able to reach much higher parameters, that are used in real applications. Compared to theoretical upper-bounds, enabling concrete computation allows a more precise estimation of the failure probability, which can in turn allow to fine-tune the ciphertext compression. For the first IBE scheme we studied, this allows to reduce the ciphertext size by about 20%. For the second one (the reference [DLP14]), this enables to decrease the failure probability (thus improving security) with a very tiny impact (encryption must also compress u).

2 An optimized estimator for failure probability

2.1 Enabling parallelism

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two probability distributions over $\mathbb{Z}_q$. Following Definition 1, we can express the law of their sum $\mathcal{D}_3 = \mathcal{D}_1 + \mathcal{D}_2$ by:

$$\forall\, z \in \mathbb{Z}_q, \quad \mathbb{P}_3(z) = \sum_{x \in \mathbb{Z}_q} \mathbb{P}_1(x)\mathbb{P}_2(z-x)$$

Therefore, a program computing $\mathcal{D}_3$ from $\mathcal{D}_1$ and $\mathcal{D}_2$ could use parallelism to compute $\mathbb{P}_3(z)$ for multiple $z \in \mathbb{Z}_q$ at the same time. However, this is not straightforward with the original Python program because of concurrent writing. Indeed, Algorithm 1 loops through the values of x and $y = z - x$. Since we can find distinct pairs (x_1, y_1) and (x_2, y_2) such that $z = x_1 + y_1 = x_2 + y_2$, without any guarantee that all these pairs are handled by a single thread, several threads could try to write to a single location ($D_3[z]$) at the same time, resulting in an unwanted behavior. That is why we rather use Algorithm 3.

Algorithm 3 `AddDistributions`

Input: A list D_1 representing a distribution $\mathcal{D}_1$.
Input: A list D_2 representing a distribution $\mathcal{D}_2$.
Output: A list D_3 representing the sum $\mathcal{D}_3 = \mathcal{D}_1 + \mathcal{D}_2$.

1: $D_3 = [0, \cdots, 0]$ ▷ *q elements*
2: **for** $z = 0$ to $q - 1$ **do** ▷ *Can be split across multiple threads*
3: **for** $x = 0$ to $q - 1$ **do**
4: $y = z - x \mod q$
5: $D_3[z] = D_3[z] + D_1[x] \times D_2[y]$

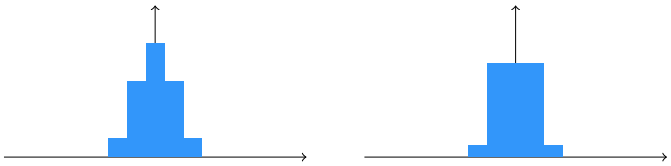
Now, we let $\mathcal{D}_4$ denote the product of $\mathcal{D}_1$ and $\mathcal{D}_2$. We have:

$$\forall\, z \in \mathbb{Z}_q^\times, \quad \mathbb{P}_4(z) = \sum_{x \in \mathbb{Z}_q^\times} \mathbb{P}_1(x)\mathbb{P}_2(zx^{-1})$$

We can also leverage parallelism with the same kind of implementation as for the addition, but the benefit is much smaller, for several reasons. First, as illustrated before in the case of Kyber, computing the distribution of the final error involves few multiplications (2 in that case). Then, if we leverage the above formula, we have to compute modular inverses, which incurs a non-negligible overhead degrading the overall performance, in addition to adding complexity to the implementation. Finally, since each multiplication involves at least one error that is sampled in the scheme from some narrow distribution, this algorithm can already be made very efficient with another technique, that we will detail in Section 2.3.

2.2 Exploiting symmetry

Earlier, we gave the expression of the distribution of the coefficients of E for Kyber. To compute that distribution, we need to compute operations on distributions starting with the Centered Binomial Distribution, and the distribution of the error resulting from ciphertext compression. Each of these distributions will be stored as an array of q probabilities, and each operation will have to loop through q^2 pairs.



(a) Centered Binomial (b) Compression error

Fig. 2: The distributions used in Kyber

From Figure 2, we see that both basis distributions are symmetric, which means that for any $z \in \mathbb{Z}_q$, we have $\mathbb{P}(z) = \mathbb{P}(-z)$. Moreover, we can check that both the sum and the product of two symmetric distributions are also symmetric, so that our program will only manipulate symmetric distributions. So, we can store only half of the probabilities, and divide by 4 the number of pairs.

When q is odd, the law of the sum $\mathcal{D}_3$ of two symmetric distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ over $\mathbb{Z}_q$ can be expressed as:

$$\mathbb{P}_3(z) = \mathbb{P}_1(0)\mathbb{P}_2(z) + \sum_{x=1}^{\lfloor q/2 \rfloor} \mathbb{P}_1(x)\Big[\mathbb{P}_2(z+x) + \mathbb{P}_2(z-x)\Big]$$

When q is even, it can be expressed as:

$$\mathbb{P}_3(z) = \mathbb{P}_1(0)\mathbb{P}_2(z) + \sum_{x=1}^{\lfloor q/2 \rfloor} \mathbb{P}_1(x)\Big[\mathbb{P}_2(z+x) + \mathbb{P}_2(z-x)\Big] - \mathbb{P}_1(q/2)\mathbb{P}_2(z+q/2)$$

The details for computing the product $\mathcal{D}_4$ of $\mathcal{D}_1$ and $\mathcal{D}_2$ is given in Appendix A. We can perform that computation as described in Algorithm 5.

2.3 Avoiding useless computations

In Kyber like in many LWE-based encryption schemes, many distributions used in the scheme have small support, which means that a random variable with that distribution can only take values very close to 0 (as illustrated in Figure 2). For example, Kyber's Centered Binomial Distributions only allow values in $[\![-2, 2]\!]$

Algorithm 4 `AddDistributions`

Input: A list D_1 representing a symmetric distribution $\mathcal{D}_1$.
Input: A list D_2 representing a symmetric distribution $\mathcal{D}_2$.
Output: A list D_3 representing the symmetric distribution $\mathcal{D}_3 = \mathcal{D}_1 + \mathcal{D}_2$.

$D_3 = [0, \cdots, 0]$ ▷ $\lfloor q/2 \rfloor + 1$ *elements*
for $z = 0$ to $\lfloor q/2 \rfloor$ **do** ▷ *Can be split across multiple threads*
    $D_3[z] = D_1[0] \times D_2[z]$
    for $x = 1$ to $\lfloor q/2 \rfloor$ **do**
        $y_1 = \min(z - x \mod q, \quad q - z + x \mod q)$
        $y_2 = \min(z + x, \quad q - z - x)$
        $D_3[z] = D_3[z] + D_1[x] \times \big(D_2[y_1] + D_2[y_2]\big)$
    if q is even **then**
        $D_3[z] = D_3[z] - D_1[q/2] \times D_2[q/2 - z]$

Algorithm 5 `MultiplyDistributions`

Input: A list D_1 representing a symmetric distribution $\mathcal{D}_1$.
Input: A list D_2 representing a symmetric distribution $\mathcal{D}_2$.
Output: A list D_3 representing the symmetric distribution $\mathcal{D}_3 = \mathcal{D}_1 \times \mathcal{D}_2$.

$D_3 = [0, \cdots, 0]$ ▷ $\lfloor q/2 \rfloor + 1$ *elements*
$D_3[0] = D_1[0] + D_2[0] - D_1[0] \times D_2[0]$ ▷ $\mathbb{P}[(X = 0) \cup (Y = 0)]$
for $x = 1$ to $\lceil q/2 \rceil - 1$ **do**
    for $y = 1$ to $\lceil q/2 \rceil - 1$ **do**
        $z_1 = x \times y \mod q$
        $z_2 = q - z_1 \mod q$
        $z = \min(z_1, z_2)$
        if $z_1 = z_2$ **then**
            $c = 4$
        else
            $c = 2$
        $D_3[z] = D_3[z] + c \times D_1[x] \times D_2[y]$
if q is even **then**
    for $x = 1$ to $q/2 - 1$ **do**
        $z = x \times q/2 \mod q$
        $D_3[z] = D_3[z] + 2 \times \big(D_1[x] \times D_2[q/2] + D_1[q/2] \times D_2[x]\big)$
    $D_3[q^2/4] = D_3[q^2/4] + D_1[q/2] \times D_2[q/2]$

or $[\![-3, 3]\!]$, depending on the parameters. So, when computing the distribution $\mathcal{D} = \mathrm{CBD}_{\eta_1} \times \mathrm{CBD}_{\eta_1}$, Algorithm 5 will consider $\lfloor q/2 \rfloor^2 = 2768896$ pairs (x, y), but only $4\eta_1^2 \leqslant 36$ of them have non-zero probability. As a consequence, this algorithm wastes a lot of time multiplying 0 with something, and adding the result (so, 0) to the corresponding cell of the output distribution.

To avoid that, the authors of Kyber represented probabilities with dictionaries, whose keys are the values with non-zero probability. To avoid the complexity of managing such dictionaries in low-level languages like C, where it is not a native data type, we rather store a bound β with each distribution $\mathcal{D}$, such that:

$$\forall\, x \in [\![0, \lfloor q/2 \rfloor]\!], \qquad x > \beta \quad \Rightarrow \quad \mathbb{P}(x) = 0$$

The bounds are then passed together with each distribution, resulting in the sum given in Algorithm 6 and the product from Algorithm 7.

Algorithm 6 `AddDistributions`

Input: A list D_1 representing a symmetric distribution $\mathcal{D}_1$.
Input: A bound β_1 on $\mathcal{D}_1$.
Input: A list D_2 representing a symmetric distribution $\mathcal{D}_2$.
Input: A bound β_2 on $\mathcal{D}_2$.
Output: A list D_3 representing the symmetric distribution $\mathcal{D}_3 = \mathcal{D}_1 + \mathcal{D}_2$.
Output: A bound β_3 on $\mathcal{D}_3$.

$D_3 = [0, \cdots, 0]$ ▷ $\lfloor q/2 \rfloor + 1$ *elements*
$\beta_3 = \min(\lfloor q/2 \rfloor, \beta_1 + \beta_2)$
for $z = 0$ to β_3 **do** ▷ *Can be split across multiple threads*
  for $x = \max(0, z - \beta_2)$ to $\min(\beta_1, z)$ **do** ▷ *Void if Start > End*
    $D_3[z] = D_3[z] + D_1[x] \times D_2[z - x]$
  for $x = z + 1$ to $\min(\beta_1, \beta_2 + z)$ **do** ▷ *Void if Start > End*
    $D_3[z] = D_3[z] + D_1[x] \times D_2[x - z]$
  for $x = 1$ to $\min(\beta_1, \beta_2 - z)$ **do** ▷ *Void if Start > End*
    $D_3[z] = D_3[z] + D_1[x] \times D_2[x + z]$
  for $x = \max(q - z - \beta_2, \lfloor q/2 \rfloor - z + 1)$ to β_1 **do** ▷ *Void if Start > End*
    $D_3[z] = D_3[z] + D_1[x] \times D_2[q - x - z]$
  if q is even, $\beta_1 = q/2$ and $\beta_2 \geqslant q/2 - z$ **then**
    $D_3[z] = D_3[z] - D_1[q/2] \times D_2[q/2 - z]$

2.4 The resulting estimator

We implemented our optimized estimator in two versions: a C implementation targeting CPUs, and a CUDA C/C++ implementation targeting GPUs. Both of them use the `double` type to avoid underflow, since its non-zero 11-bit exponent biased by 1 023 allows to handle numbers down to 2^{-1022}. We provide this tool in an open source project organized in one module that implements our optimizations to quickly compute on distributions, one module for each supported scheme that uses it to compute the final error distribution, and a `main` program that computes the resulting failure probability of the chosen scheme [Cam26].

Algorithm 7 `MultiplyDistributions`

Input: A list D_1 representing a symmetric distribution $\mathcal{D}_1$.
Input: A bound β_1 on $\mathcal{D}_1$.
Input: A list D_2 representing a symmetric distribution $\mathcal{D}_2$.
Input: A bound β_2 on $\mathcal{D}_2$.
Output: A list D_3 representing the symmetric distribution $\mathcal{D}_3 = \mathcal{D}_1 \times \mathcal{D}_2$.
Output: A bound β_3 on $\mathcal{D}_3$.

$D_3 = [0, \cdots, 0]$ ▷ $\lfloor q/2 \rfloor + 1$ *elements*
$D_3[0] = D_1[0] + D_2[0] - D_1[0] \times D_2[0]$ ▷ $\mathbb{P}[(X = 0) \cup (Y = 0)]$
$\beta_3 = \min(\lfloor q/2 \rfloor, \beta_1 \times \beta_2)$
for $x = 1$ to $\min(\beta_1, \lceil q/2 \rceil - 1)$ **do**
    for $y = 1$ to $\min(\beta_2, \lceil q/2 \rceil - 1)$ **do**
        $z_1 = x \times y \mod q$
        $z_2 = q - z_1 \mod q$
        $z = \min(z_1, z_2)$
        if $z_1 = z_2$ **then**
            $c = 4$
        else
            $c = 2$
        $D_3[z] = D_3[z] + c \times D_1[x] \times D_2[y]$
if q is even **then**
    if $\beta_1 = q/2$ **then** ▷ $x = q/2$
        for $y = 1$ to $\min(\beta_2, q/2 - 1)$ **do**
            $z = q/2 \times y \mod q$
            $D_3[z] = D_3[z] + 2 \times D_1[q/2] \times D_2[y]$
    if $\beta_2 = q/2$ **then** ▷ $y = q/2$
        for $x = 1$ to $\min(\beta_1, q/2 - 1)$ **do**
            $z = x \times q/2 \mod q$
            $D_3[z] = D_3[z] + 2 \times D_1[x] \times D_2[q/2]$
    if $\beta_1 = \beta_2 = q/2$ **then** ▷ $x = y = q/2$
        $D_3[q^2/4] = D_3[q^2/4] + D_1[q/2] \times D_2[q/2]$

To quantify the speed-up achieved by our optimizations, we run the implementations on various schemes while measuring the running time, which allowed to build the results presented in Table 1. Both the original Python estimator from the authors of Kyber (the CRYSTALS team) and the CPU version of our tool (with 12 threads) were run on a laptop with a 12th gen Intel Core i7 (2.2GHz). The GPU version of our tool was run on a server with 2 Nvidia RTX 4500 Ada Generation and 6 Nvidia GeForce RTX 2080 Ti.

A difficulty lives in the huge gap that exists between all the versions. For example, using ML-KEM for the comparison is not very precise, since all versions are very fast (the GPUs probably spend more time copying than actually computing). Conversely, IBE schemes like [DLP14] and [Cam25] use such large parameters that the original estimator takes way too much time, which illustrates the interest of our work, but prevents us from comparing the running times on them. Finally, we found out that [CKKS19], which can somewhat be seen as a weak version of [Cam25], gives an interesting compromise between these two extreme situations, although this scheme is probably broken [Cam25].

Table 1: The running times of the estimators (in seconds)

Scheme	CRYSTALS	CPU	GPU
ML-KEM-512	0.75	0.05	0.70
ML-KEM-768	0.57	0.05	0.70
ML-KEM-1024	0.58	0.05	0.70
[CKKS19]	9 461	50	4
[Cam25]			143
[DLP14]-80			124
[DLP14]-192			28 241

Using a few GPUs like we did for building Table 1 allows our estimator to reach practical use-cases such as Identity-Based Encryption schemes (more details about IBE in Section 3) within a reasonable execution time. Moreover, computing the failure probability of such a scheme has to be done only once, during the design phase. Once our estimator has allowed to find the best parameters for the scheme, we can use the scheme with these optimized compression parameters, and the estimator does not need to run anymore.

3 Example application: Identity-Based Encryption

Identity-Based Encryption is a special kind of Public-Key Encryption, where the user is able to arbitrarily choose a public key. So that any user cannot compute any secret key, computing a secret key from a public key must be a generally hard problem, that only a trusted authority can solve thanks to a secret trapdoor.

Among lattice-based IBEs, the most widely recognized scheme is [DLP14], which uses NTRU lattices to provide the authority with a trapdoor. The extraction of a user's secret key relies on trapdoor sampling, which results in (s, e_k) being sampled from a discrete Gaussian distribution, rather than from a Centered Binomial Distribution as in Kyber.

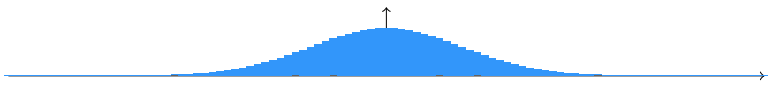


Fig. 3: A discrete Gaussian distribution

We represented in Figure 3 the density function of an example discrete Gaussian distribution. We can see that this new distribution is symmetric, so it is compatible with our improved Failure Probability Estimator. However, we also see that it does not have small support, which has two main consequences.

First, the error terms e_k and s will probably be significantly larger than in Kyber. So, the final error term E will be much larger, which will dramatically increase the failure probability $\mathbb{P}(\|E\|_\infty > \lfloor q/4 \rfloor)$. This would break both the scheme's functionality and its security. To compensate that effect, IBE schemes must use a larger modulus q.

Second, the optimization based on bounding the distributions (described in Section 2.3, and also used by the authors of Kyber) will not help with this distribution. The number of iterations required by one operation will then be bounded by $\lfloor q/2 \rfloor$ rather than by the end of the distribution's support. As a consequence, the computational cost of the estimator will increase, especially since we also increase the modulus q, as explained above.

For these reasons, the estimator used for Kyber is completely inefficient for IBE schemes. That is why Identity-Based Encryption is a field where our optimized estimator can help to precisely compute the failure probability, since the currently used techniques rely on theoretical approximations and upper bounds.

Since the IBE schemes we consider rely on the same foundations as Kyber, their structure (which is inherited from LWE-encryption) are very similar, and thus a decryption failure would likely have the same consequences, enabling the same attacks with about the same cost and success probability. So, as in Kyber, we choose to target a failure probability below 2^{-128}.

3.1 An IBE scheme based on ML-KEM

Recently, an Identity-Based Encryption scheme was built with the objective of reusing the encryption and decryption algorithms from ML-KEM, the standard built by NIST from Kyber [Cam25]. Thanks to that objective, its structure is similar to that of Kyber that we already presented, which makes it easy to explain the application of our tool to an IBE scheme.

In the case of Kyber, the final error term was given by:

$$E = e_k^{\mathrm{T}} \times y - s^{\mathrm{T}} \times (e_1 + e_u) + e_2 + e_v$$

where e_u and e_v were created by compression, and all the other terms had Centered Binomial Distributions. In the case of [Cam25], there only exist two differences. First, as explained before, e_k and s have a discrete Gaussian distribution. Second, only the first term of e_k is non-zero, so we replace e_k and y by their first coordinates without changing the notation. Finally, we have:

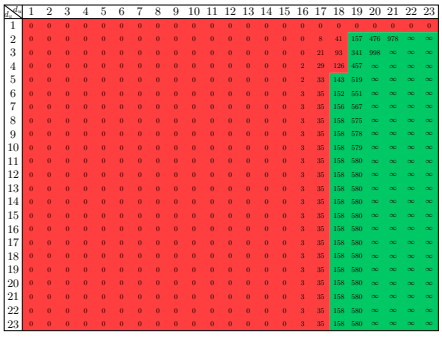
$$E = e_k \times y - s^{\mathrm{T}} \times (e_1 + e_u) + e_2 + e_v$$

- The coefficients of $e_k \in R_q$ and $s \in R_q^k$ have distribution $\mathcal{N}_{\mathbb{Z},0,\sigma}$.
- The coefficients of $y \in R_q$ have distribution CBD_{η_1}.
- The coefficients of $e_1 \in R_q^k$ and $e_2 \in R_q$ have distribution CBD_{η_2}.
- The coefficients of $e_u \in R_q^k$ and $e_v \in R_q$ are compression errors.

Therefore, the distribution of the coefficients of E is given by:

$$\mathcal{D} = n \cdot [\mathcal{N}_{\mathbb{Z},\sigma} \times \mathrm{CBD}_{\eta_1}] - (k \times n) \cdot [\mathcal{N}_{\mathbb{Z},\sigma} \times (\mathrm{CBD}_{\eta_2} + \mathcal{D}_u)] + \mathrm{CBD}_{\eta_2} + \mathcal{D}_v$$

Using that formula, we can use our estimator to compute the failure probability associated to any parameter set. Since the running time of our estimator on these parameters is reasonable, we give the estimated failure probability for all possible pairs, even if this is not required for choosing parameters (see Section 3.2 for a faster method). We present in Figure 4a the pairs (d_u, d_v) that achieve (in green) or not (in red) negligible failure probability. Then, we compute in Figure 4b the size of the ciphertext corresponding to each pair (d_u, d_v) of compression parameters. Finally, we can choose the best possible parameters, which minimize the size of the ciphertext while achieving negligible failure probability.



(a) $\lfloor -\log_2 (n \times \mathbb{P}(|E_0| > \lfloor q/4 \rfloor)) \rfloor$



(b) Size of the ciphertext (in bytes)

Fig. 4: Choosing compression parameters to reduce ciphertext size in [Cam25]

With our approach, we confirm that the original compression parameters from [Cam25] achieve negligible failure probability, even though the choice was only backed up with heuristics based on previous schemes. Moreover, our estimator gives a more accurate figure for the failure probability, which could only be upper bounded in previous IBE schemes. Finally, thanks to this precise estimation, we are able to fine-tune the ciphertext compression, which enables to decrease the size of the ciphertext by about 20%, as presented in Table 2.

Table 2: The improvement allowed by our estimator

	d_u	d_v	Ciphertext	$\mathbb{P}$(Failure)
[Cam25]	23	3	6 272 bytes	Unknown
This paper	19	2	5 120 bytes	2^{-157}

3.2 The scheme from Ducas, Lyubashevsky and Prest

In this section, we leverage our optimized estimator for the reference lattice-based IBE scheme. In 2014, Ducas, Lyubashevsky and Prest proposed an IBE scheme, which is still recognized as one of the most efficient lattice-based IBE schemes today [DLP14].

Compared to the schemes we presented before, [DLP14] is fully based on Ring-LWE, rather than Module-LWE, which basically means that $k = 1$. Moreover, the small errors y, e_1 and e_2 are sampled uniformly from $\{-1, 0, 1\}$, while (sk, e_k) is sampled from a discrete Gaussian distribution.

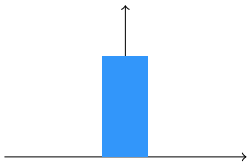


Fig. 5: The distribution of the small errors in [DLP14]

Another important difference with ML-KEM and the IBE based upon it is that there is no compression on u. However, in order to reduce as much as possible the size of the ciphertext, we enable this feature in this paper. Finally:

$$E = e_k \times y - s \times (e_1 + e_u) + e_2 + e_v$$

- The coefficients of $(e_k, s) \in R_q^2$ have distribution $\mathcal{N}_{\mathbb{Z},0,\sigma}$.
- The coefficients of $(y, e_1, e_2) \in R_q^3$ have distribution $\mathcal{U}_{\{-1,0,1\}}$.
- The coefficients of $e_u \in R_q$ and $e_v \in R_q$ are compression errors.

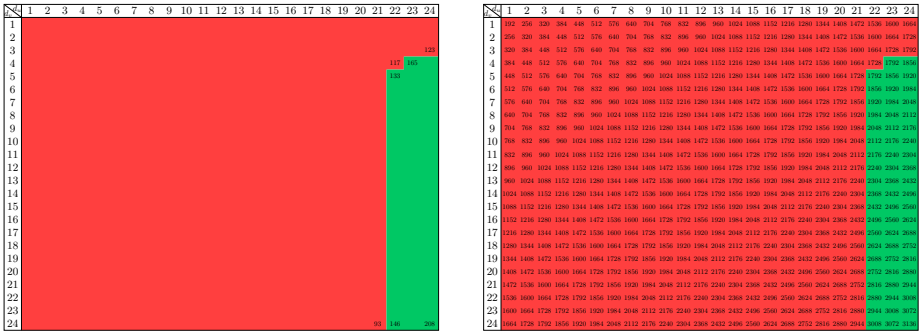
Therefore, the distribution of the coefficients of E is given by:

$$\mathcal{D} = n \cdot [\mathcal{N}_{\mathbb{Z},\sigma} \times \mathcal{U}_{\{-1,0,1\}}] - n \cdot [\mathcal{N}_{\mathbb{Z},\sigma} \times (\mathcal{U}_{\{-1,0,1\}} + \mathcal{D}_u)] + \mathcal{U}_{\{-1,0,1\}} + \mathcal{D}_v$$

As pointed out in [Cam25], the security levels announced in [DLP14] are probably outdated. However, our goal here is to illustrate the benefits of better estimating the failure probability, not to find a concrete instantiation with state-of-the-art security, which explains why we stick to the original parameters.

We could study the compression parameters as in Section 3.1, but we can reduce the number of estimations by noting that the failure probability is a decreasing function of the compression parameters, for the product order.

Here, our tool allows to gain on two important aspects. First, since the original paper used a theoretical upper bound, the failure probability was unknown. Our tool enables to compute the precise value (2^{-123}). Second, exploring various compression parameters, our tool allows to find the best possible parameters. More precisely, we found new compression parameters that allow to divide the failure probability by 2^{42} while keeping the exact same ciphertext size. Our results are presented in Figure 6 and Table 3.



(a) $\lfloor -\log_2 (n \times \mathbb{P}(|E_0| > \lfloor q/4 \rfloor)) \rfloor$ (b) Size of the ciphertext (in bytes)

Fig. 6: Choosing compression parameters to reduce ciphertext size in [DLP14]

Table 3: The improvement allowed by our estimator

	d_u	d_v	Ciphertext	$\mathbb{P}$(Failure)
[DLP14]	24	3	1 792 bytes	Unknown
This paper				2^{-123}
	23	4	1 792 bytes	2^{-165}

4 Future perspectives

In Section 3, we illustrated the benefits of our estimator in the context of lattice-based IBE. However, our tool is designed to be as generic as possible, and is able to compute the failure probability in most MLWE-based schemes. The only restriction is that all distributions sampled in the scheme must be symmetric, which holds for the vast majority of such schemes. We think our estimator could help in the context of Fully Homomorphic Encryption, to give concrete estimations of the failure probability.

When concrete computation was not possible, for example because of large parameters, we usually leverage approximations and upper bounds, with the following methodology. First, we assume that the coefficients of E have Gaussian distribution, since they are sums of many independent terms. Then, we compute the standard deviation of that discrete Gaussian distribution. Finally, we consider that "the discreteness is negligible", so that we can use a bound on the tail of a continuous Gaussian distribution. With our estimator, we could progressively increase the parameters (especially the modulus) to study if the Gaussian approximation really holds, and see how tight is the final bound.

5 Conclusion

In this paper, we designed an optimized Failure Probability Estimator, which enables to concretely compute the failure probability of many schemes based on Ring-LWE or Module-LWE. For several Identity-Based Encryption schemes, our estimator allows to decrease the sizes via a fine-tuned ciphertext compression.

To achieve that, we changed the way the operations are computed to enable parallelism, and we leveraged the shape of the distributions (symmetric and bounded) to improve the theoretical complexity of the algorithms.

Moreover, we implemented our estimator in low-level languages to better serve the performance objective. The resulting tool can run both on a CPU (with the C implementation) and on a GPU (thanks to the CUDA implementation), so that it can better serve each use-case. So that it can help better understand failure probability, and improve the parameters to decrease the sizes, we make our estimator available on a public GitHub repository [Cam26].

Our efficient estimator enabled to refine the analysis of two lattice-based Identity-Based Encryption schemes (including the reference [DLP14]). More precisely, more precise measurement of the failure probability has two separate advantages illustrated on these two schemes. First, it can allow to reduce the ciphertext size, illustrated by a gain of 20% for the first IBE scheme we studied. Second, it can allow to improve security, by reaching lower failure probability, illustrated by dividing it by 2^{42} for the exact same sizes for [DLP14].

References

BS20. Nina Bindel and John M. Schanck. Decryption failure is more likely after success. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 206–225. Springer, Cham, 2020. doi:10.1007/978-3-030-44223-1_12.

Cam25. Julien Cam. New post-quantum IBE leveraging maturity, efficiency and security of standard schemes. In Nadia El Mrabet, Loubna Ghammam, and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - AFRICACRYPT 2026*, pages 29–56, Cham, 2025. Springer Nature Switzerland. URL: https://eprint.iacr.org/2025/2143.

Cam26. Julien Cam. An efficient failure probability estimator for lwe-based encryption schemes, 2026. URL: https://github.com/JCamCrypto/Failure_Probability_Estimator.

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. URL: https://eprint.iacr.org/2019/1468.

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

DGJ+19. Jan-Pieter D'Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 565–598. Springer, Cham, April 2019. doi:10.1007/978-3-030-17259-6_19.

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. doi:10.1007/978-3-662-45608-8_2.

DRV20. Jan-Pieter D'Anvers, Mélissa Rossi, and Fernando Virdia. (One) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 3–33. Springer, Cham, May 2020. doi:10.1007/978-3-030-45727-3_1.

DVV18. Jan-Pieter D'Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. On the impact of decryption failures on the security of LWE/LWR based schemes. Cryptology ePrint Archive, Report 2018/1089, 2018. URL: https://eprint.iacr.org/2018/1089.

Nat24. 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. doi:10.6028/NIST.FIPS.203.

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. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

A Multiplying symmetric distributions

The computation performed in Algorithm 2 by the authors of Kyber can be written mathematically as follows:

$$\mathbb{P}_3(z) = \sum_{x\in\mathbb{Z}_q}\sum_{y\in\mathbb{Z}_q} \mathbb{1}(z = xy)\mathbb{P}_1(x)\mathbb{P}_2(y)$$

With $f(z)$ denoting $\mathbb{1}(z=0)\mathbb{P}\big([X=0]\cup[Y=0]\big)$, we have:

$$\mathbb{P}_3(z) = f(z) + \sum_{x\in\mathbb{Z}_q^*}\sum_{y\in\mathbb{Z}_q^*} \mathbb{1}(z = xy)\mathbb{P}_1(x)\mathbb{P}_2(y)$$

If q is even, $[\![1-q/2,-1]\!]\cup[\![1,q/2]\!]$ represents $\mathbb{Z}_q^*$. Therefore:

$$\begin{aligned}\mathbb{P}_3(z) = f(z) &+ \sum_{x=1}^{q/2-1}\sum_{y=1}^{q/2-1} 2\Big[\mathbb{1}(z=-xy)+\mathbb{1}(z=xy)\Big]\mathbb{P}_1(x)\mathbb{P}_2(y)\\ &+\sum_{x=1}^{q/2-1} 2\times\mathbb{1}(z=x[q/2])\Big[\mathbb{P}_1(x)\mathbb{P}_2(q/2)+\mathbb{P}_1(q/2)\mathbb{P}_2(x)\Big]\\ &+\mathbb{1}(z=[q/2]^2)\mathbb{P}_1(q/2)\mathbb{P}_2(q/2)\end{aligned}$$

If q is odd, $[\![-\lfloor q/2\rfloor,-1]\!]\cup[\![1,\lfloor q/2\rfloor]\!]$ represents $\mathbb{Z}_q^*$. Therefore:

$$\mathbb{P}_3(z) = f(z) + \sum_{x=1}^{\lfloor q/2\rfloor}\sum_{y=1}^{\lfloor q/2\rfloor} 2\Big[\mathbb{1}(z=-xy)+\mathbb{1}(z=xy)\Big]\mathbb{P}_1(x)\mathbb{P}_2(y)$$

So, for any q, we have:

$$\begin{aligned}\mathbb{P}_3(z) = \ &\mathbb{1}(z=0)\big[\mathbb{P}_1(0)+\mathbb{P}_2(0)-\mathbb{P}_1(0)\mathbb{P}_2(0)\big]\\ &+\sum_{x=1}^{\lceil q/2\rceil-1}\sum_{y=1}^{\lceil q/2\rceil-1} 2\Big[\mathbb{1}(z=-xy)+\mathbb{1}(z=xy)\Big]\mathbb{P}_1(x)\mathbb{P}_2(y)\\ &+\mathbb{1}(q\in2\mathbb{Z})\sum_{x=1}^{q/2-1} 2\times\mathbb{1}(z=x[q/2])\Big[\mathbb{P}_1(x)\mathbb{P}_2(q/2)+\mathbb{P}_1(q/2)\mathbb{P}_2(x)\Big]\\ &+\mathbb{1}(q\in2\mathbb{Z})\mathbb{1}(z=[q/2]^2)\mathbb{P}_1(q/2)\mathbb{P}_2(q/2)\end{aligned}$$