Cryptography and Communications
https://doi.org/10.1007/s12095-025-00837-z

RESEARCH

CCA-attacks on lattice-based encryption-decryption schemes

Alba Hernández Costoya[1] · **Alba Larraya Sancho**[1,2] · **Miguel Ángel Marco Buzunáriz**[3]

Received: 30 June 2025 / Accepted: 26 August 2025
© The Author(s) 2025

Abstract
This paper presents two distinct chosen-ciphertext attacks (CCA) against lattice-based encryption and decryption schemes, in particular based on the LWE problem, a class of post-quantum cryptographic algorithms. First we attack fully homomorphic encryption-decryption schemes (FHE) exploiting the additional information that the small modulus reduction offers. We compare this attack with the CPA^D attack presented in Checri (2024). Afterwards we present an attack against Kyber.CPAPKE, and another against a weakened version of Kyber.KEM where the cyphertext is not checked for correctness; and compare them to the previous Key Mismatch Attack in Qin (2021). Our work remarks the importance of protecting the decryption function in the different implementations of these cryptographic schemes, and the importance of CCA security in nowadays cryptosystems.

Keywords Learning with errors · CCA-attacks · Lattice-based cryptography · Fully homomorphic encryption

Mathematics Subject Classification (2010) 94a60

Alba Hernandez Costoya, Alba Larraya Sancho, and Miguel Ángel Marco Buzunáriz contributed equally to this work.

Funded by CEX2021-001142-S-20-9, Spanish Ministry of Science, Innovation and Universities and the State Research Agency. Funded by $\mathrm{E22}_{23\mathrm{R}}$:Álgebra y Geometría and PID2020-114750GB-C31.

✉ Alba Larraya Sancho
alarraya@bcamath.org

Alba Hernández Costoya
ahernandez148@ikasle.ehu.eus

Miguel Ángel Marco Buzunáriz
mmarco@unizar.es

[1] Universidad del País Vasco, Bilbao, Spain

[2] Basque Center for Applied Mathematics, Bilbao, Spain

[3] Universidad de Zaragoza, Zaragoza, Spain

Published online: 21 October 2025

1 Introduction

In the last years, there has been a big progress in the field of post-quantum cryptography, with lattice based cryptography being really promising. In 2016, NIST started studying different post-quantum cryptographic schemes, divided in five categories: lattice-based, code-based, multivariate-based, supersingular isogeny schemes and hash-based. In 2022, NIST selected four schemes out of the 69 initially received, three of which were lattice-based, CRYSTALS-Kyber, CRYSTALS-Dilithium and Falcon, and the last one was hash-based, SPHINCS$^+$. The first one of these schemes is used for general encryption, while the other three are chosen for digital signatures. These standards were published in August 2024. In March 2025 another algorithm was selected for standarization, HQC, which is code-based and used for general encryption. There are three additional algorithms which are being considered to be added to the standard.

Kyber is composed by a CPA-secure PKE, which is transformed into a CCA-secure KEM with the Fujisaki-Okamoto transform. The security of Kyber is based on the Module-LWE problem.

To exemplify the dangers of reusing the public keys, there have been previous works studying Key-Mismatch attacks against lattice-based KEMs, [2–4]. In those works, it is assumed that the attacker has access to an oracle that can say if the key encapsulation-decapsulation process was successful, and perform a series of queries in order to recover the private key. In this work, we present a CCA attack instead: we will assume that the attacker has access to a decryption oracle (that is, an oracle that implements the decryption function of the underlying PKE), and use it to recover the private key. This is a stronger assumption than in the key mismatch attack (we assume a decryption oracle, rather than a KEM oracle), but on the other hand, our attacks will require far less queries. After the attack to the decryption function, we have developed an attack to a CPA version of the KEM, that is, an oracle that performs the KEM operations except for the initial cyphertext verification in the Fujisaki-Okamoto transform.

Other kinds of attacks have also been studied for LWE-based schemes. In [1], CPA^D attack against LWE-based fully homomorphic encryption schemes is proposed. For this attack, the attacker is allowed to demand the oracle to perform encryption, decryption and evaluation of plaintexts and ciphertexts. The attack consists in making queries that allow to obtain information about the error term, such as the amplitude and the sign, and afterwards solving two linear systems.

The LWE problem was first studied by O. Regev in [5], where it is used as the base for a Public Key Cryptosystem on Integral Lattices. A general template for (Ring-)LWE based encryption schemes was afterwards proposed in [6], it conforms to the following structure:

Let $R_q = \mathbb{Z}/(q, x^n + 1)$, where q is a prime powerand n is an integer. Let k be a security parameter, also an integer.

Let $\mathrm{A} \in R_q^{k \times k}$, $\mathrm{s} \in R_q^k$ be a random secret key obtained with either a discrete gaussian distribution or with a binomial distribution of parameter η, and let e be an error vector obtained from the same distribution as s. Then the public key will be $(\mathrm{A}, \mathrm{b} = \mathrm{As} + \mathrm{e})$.

In order to encrypt a message m, given by a polynomial of degree n with 0 or 1 coefficients, we sample two polynomial vectors r, e_1 from the same distribution as s and a polynomial e_2, also from the same distribution as s, and compute:

$$\begin{array}{ll} \mathrm{u} = & \mathrm{A}\mathrm{r} + \mathrm{e}_1, \\ v = & \mathrm{b}\mathrm{r} + \left\lfloor \frac{q}{2} \right\rceil m + \mathrm{e}_2. \end{array} \tag{1}$$

The ciphertext will be $\mathrm{c} = (\mathrm{u}, v)$. In order to decipher and retrieve m we perform the following computation:

$$m_i = \begin{cases} 0 & \text{if } (v - \mathrm{u} \cdot \mathrm{s})_i \bmod q \in \left[-\frac{q}{4}, \frac{q}{4}\right], \\ 1 & \text{otherwise.} \end{cases} \tag{2}$$

where m_i and $(v - \mathrm{u} \cdot \mathrm{s})_i$ denote the *i*'th coefficient of the corresponding polynomials.

Note that given a decryption oracle for this general template, we could obtain the *j*-th component of the secret key by querying the oracle with u equal to minus the *j*-th vector of the canonical basis, and changing the value of the coefficients of v in order to see at which point $v + s$ changes from $\left[-\frac{q}{2}, -\frac{q}{4}\right) \cup \left(\frac{q}{4}, \frac{q}{2}\right)$ to $\left[-\frac{q}{4}, \frac{q}{4}\right]$.

However, in practical cases, this general scheme is not used as is, but some variants of it. Those variations prevent this trivial attack. In particular, CRYSTALS-Kyber introduces some extra compression and decompression functions [7], and [8] introduces an extra reduction modulo some additional small integer t coprime with q (in practice, $t = 2$), the scheme in this paper uses the particular case $n = 1$, which gives, $R_q = \mathbb{Z}/q\mathbb{Z}$, this is, it is based on integer LWE.

In this paper, we present CCA attacks on these two schemes. This attacks allows to recover the secret key by doing at most $k \log_2 q$ queries to the decryption oracle.

2 Description of the schemes

In this section we give a brief description of the schemes for which we will present attacks in the next section.

2.1 Fully homomorphic schemes

In [8], a fully homomorphic encryption scheme, based on the general LWE encryption scheme by O. Regev, is presented. This scheme allows to evaluate additive and multiplicative functions on ciphertexts in such a way that the decryption of the result coincides with the evaluation of the function on the original plaintext. We present now a brief sketch of this scheme.

Let $R_q = \mathbb{Z}/q\mathbb{Z}$, where q is a prime number of the order of 2^{10} or a power of 2. Let k be a security parameter, also an integer, and t a small integer, coprime with q.

Let $\mathrm{A} \in R_q^{k' \times k}$, with $k' > k \log q$. Then the fully homomorphic encryption scheme [8] is constructed as follows: let $\mathrm{s} \in R_q^k$ be a uniformly random secret key (that is, a vector whose k entries are integers modulo q), and let e be an error vector obtained from a noise distribution. Then the public key will be $(\mathrm{A}, \mathrm{b} = \mathrm{A}\mathrm{s} + t\mathrm{e})$.

Besides the public key, another key, called evaluation key, is generated. This key is used to perform the encrypted evaluation function, whose output can then be decrypted. Since it has no interest for the attack, we will not get into details about this encrypted evaluation.

In order to encrypt a single bit m, we sample a vector r uniformly, and compute:

$$\begin{aligned} \mathrm{u} &= \mathrm{Ar}, \\ v &= \mathrm{br} + m. \end{aligned} \tag{3}$$

The ciphertext will be $\mathrm{c} = (\mathrm{u}, v)$. In order to decipher and retrieve m we perform the following computation:

$$m = ((v - \mathrm{u} \cdot \mathrm{s}) \bmod q) \bmod t, \tag{4}$$

where the operation $\bmod q$ returns a representative of the class between $-\frac{q}{2}$ and $\frac{q}{2}$ (and that integer number is then reduced modulo t).

In the next section we will present an attack consisting on computing s by querying an oracle that retrieves m for given values of (u, v)[1].

In [1], the authors attack a fully homomorphic cryptosystem defined similarly, but they maintain the error vector when computing u, and a vector scalar when computing v. Said cryptosystem is based on Regev's LWE Integer cryptosystem in [5]. The decryption function can be written as

$$\mathrm{dec}(\mathrm{u}, v) = \left\lfloor \frac{2}{q}(v - s \cdot u) \right\rceil \bmod 2. \tag{5}$$

This allows them to use q a power of 2 and $t = 2$, not coprime, and still be able to decrypt correctly, since it takes all $(v - s \cdot u)$ that are in $\left[-\frac{q}{4}, \frac{q}{4}\right]$ to 0 and the rest to 1.

2.2 Kyber encryption scheme

Kyber is a key encapsulation mechanism, selected by NIST for the first Post-Quantum cryptographic standard. It has been proven to be Ind-CCA2 secure [7] when used strictly as KEM, but it is built on top of an encryption-decryption scheme for which we will show a CCA attack (that is, we will assume that an attacker has access to a full decryption oracle, rather than a KEM decapsulation oracle). The Kyber standard includes Encode and Decode functions to translate messages into vectors with entries in a given polynomial ring. For the purpose of this paper we will work directly with the polynomial representation, ignoring the Encode and Decode functions, since they don't affect our attacks.

Alice's secret key is a polynomial vector s in R_q^k chosen randomly from a binomial distribution with parameter η_1, and her public key is composed by a matrix $\mathrm{A} \in R_q^{k \times k}$ (sampled from a uniform distribution), and a vector $\mathrm{b} = \mathrm{As} + \mathrm{e}$, where e is an error vector taken from the binomial distribution of parameter η_2 (depending on the security level), where $R_q = \mathbb{Z}[x]/(q, x^n + 1)$. That is, the elements of R_q can be represented as polynomials of degree less than n, whose coefficients are integers modulo q.

To optimize the size of the ciphertext, Kyber uses compress and decompress functions that act on each coefficient of the polynomials, given by:

[1] And hence, the attack will also work for other schemes using the same decryption function.

$$\text{Compress}_q(a,d) = \left\lfloor \frac{2^d}{q} \cdot a \right\rceil \bmod 2^d, \tag{6}$$

$$\text{Decompress}_q(a,d) = \left\lfloor \frac{q}{2^d} \cdot a \right\rceil. \tag{7}$$

Where d is a parameter that determines how we are changing the interval, from $[\frac{-q}{2}, \frac{q}{2}]$ to $[-2^d, 2^d]$.

Then to encrypt a message m, which will be a polynomial with coefficients 0 or 1, take a two vectors r and e_1 sampled from binomial distributions with parameter η_1 and η_2 respectively, and a polynomial e_2 sampled from the same binomial as e_1, and compute:

$$\begin{aligned} \mathrm{u} &= \mathrm{Ar} + \mathrm{e}_1, \\ v &= \mathrm{br} + \text{Decompress}_q(m,1) + \mathrm{e}_2. \end{aligned} \tag{8}$$

The ciphertext consists of:

$$\begin{aligned} \mathrm{c}_1 &= \text{Compress}_q(\mathrm{u}, d_u), \\ c_2 &= \text{Compress}_q(v, d_v). \end{aligned} \tag{9}$$

In order to decrypt the received message (c_1, c_2), first we decompress:

$$\begin{aligned} \mathrm{u} &= \text{Decompress}_q(\mathrm{c}_1, d_u), \\ v &= \text{Decompress}_q(c_2, d_v), \end{aligned} \tag{10}$$

and then we compute $m = \text{Compress}_q\left(v - \mathrm{s}^T\mathrm{u}, 1\right)$.

All the parameters used for Kyber are specified in Table 1:

3 Description of the attacks

3.1 Attacks on the fully homomorphic scheme

Let q and t be as in section 2.1. Assume that a secret key s has been computed and that we have an oracle that computes the function $m(\mathrm{u}, v)$ as in equation (4). Our goal is to recover s by doing as few queries to m as possible.

The main idea for the first attack is bisecting the interval $I = \left[-\frac{q}{2}, \frac{q}{2}\right)$ in which we can find the components of the secret key (Table 2). Since the key is a vector with k components, we will need to repeat the process k times. For this process, we will denote s^j the $\mathrm{j}-th$ query to the oracle and s_j the $\mathrm{j}-th$ component of s.

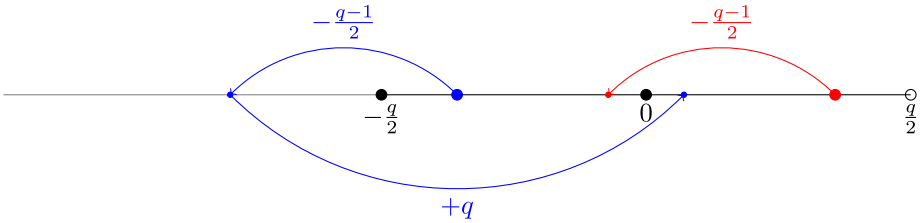
Table 1 Parameters for Kyber

	n	q	k	η_1	η_2	d_u	d_v
Kyber512	256	3329	2	3	2	10	4
Kyber768	256	3329	3	2	2	10	4
Kyber1024	256	3329	4	2	2	11	5

In order to obtain the *j*-th component of s, we will fix $\mathrm{u} = \mathrm{e}_j$ (being e_j the *j*-th element of the canonical basis) and for each query we will change *v*.

The first step is to compute $s^0 = m(-\mathrm{e}_j, 0) = s_j \bmod t$. After that, we will use different values of *v*, since that will allow us to compute

$$m(-\mathrm{e}_j, v) = (v + s_j) \bmod q \bmod t = \begin{cases} (v + s_j) \mod t & \text{if } v + s_j \in I, \\ (v + q + s_j) \mod t & \text{otherwise.} \end{cases}$$



We can then start the process of successively bisecting the interval to bound s_j. First, to know whether it is positive or negative, we choose $v = -\frac{q-1}{2}$, then, the negative half of the interval *I* will end out of said interval, and the oracle will output $s^1 = \left(v + q + s^0\right) \mod t$, and the non-negative half will stay in *I*, so the oracle will output $s^1 = \left(v + s^0\right) \mod t$. Since we have computed s^0, we can compare and decide which of the outputs we have obtained from the oracle. We will then establish the lower and upper bounds of s_j as $\left[-\frac{q}{2}, 0\right)$ or $\left[0, \frac{q}{2}\right)$.

On the following steps we take $v = -\mathrm{lb} - \frac{q}{2} - \mathrm{mp}$, where mp is the middlepoint, that is $\frac{\mathrm{lb}+\mathrm{ub}}{2}$, where lb will denote the lowerbound and ub the upperbound. If I_i is the interval in which we know s_j is, this allows us to translate I_i so that its new middle point is $-\frac{q}{2}$,

Table 2 Pseudocode for the bisection attack

Bisection attack:
Input: q, t, k,$\mathcal{O}_{q,t,n}^{CCA-Hom}$.
1:for $j = 1, \ldots, k$:
2: $s^0 \leftarrow \mathcal{O}_{q,t,n}^{CCA-Hom}(-\mathrm{e}_j, 0)$
3: $\mathrm{lb} \leftarrow -\lfloor -q/2 \rfloor$
4: $\mathrm{ub} \leftarrow \lfloor q/2 \rfloor$
5: for $i = 1, \ldots, \lceil \log_2 q \rceil$:
6: $\mathrm{mp} \leftarrow \lfloor (\mathrm{lb} + \mathrm{ub})/2 \rfloor$
7: $v \leftarrow \mathrm{lb} + \lfloor q/2 \rceil + \mathrm{mp}$
8: $s' \leftarrow \mathcal{O}_{q,t,n}^{CCA-Hom}(-\mathrm{e}_j, -v)$
9: **if** $s^0 = s'$:
10: $\mathrm{lb} \leftarrow \mathrm{lb} + v$
11: else:
12: $\mathrm{ub} \leftarrow \mathrm{ub} - v$
13: $\mathrm{s}[j] \leftarrow \mathrm{lb}$
14: returns.
Output:s.

effectively taking half of I_i out of I while living the other half inside, so the interval will be bisected and we can know in which half s_j is.

When dealing with a specific scheme, the chosen ciphertext (u, v) will be adapted to the specific case in order to obtain higher efficiency, as we see with the Kyber encryption scheme.

We only need $\lceil \log_2 q \rceil$ repetitions of this process to get an interval in which there is only one integer, s_j.

Definition 1 Let q, t, n be the scheme parameters, for q and t coprime. For $\mathrm{u} \in R_q^k$ and $v \in R_q$, we define the CCA-oracle for the fully homomorphic cryptosystem as an oracle performing the decipher operation:

$$\mathcal{O}_{q,t,n}^{CCA-Hom}(\mathrm{u}, v) = ((v - \mathrm{u} \cdot s) \bmod q) \bmod t. \tag{11}$$

In [1], Checri et al. present a CPA^D attack against fully homomorphic schemes that use a different decryption function from the one we presented, in particular, they take the decryption function (5). In this case, we can adapt our bisection attack and still recover the secret key. In section 5 we compare their attack to ours in term of queries. The oracle used for the attack on these kind of schemes will be:

Definition 2 Let q, t, n be the scheme parameters, for q and t not necessarily coprime. For $\mathrm{u} \in R_q^k$ and $v \in R_q$, we define the CCA-oracle for the fully homomorphic cryptosystem as an oracle performing the decipher operation:

$$\mathcal{O}_{q,t,n}^{CCA-Hom,2}(\mathrm{u}, v) = \left\lfloor \frac{2}{q}(v - \mathrm{s} \cdot \mathrm{u}) \right\rceil \bmod t. \tag{12}$$

The idea of the second attack will be to recover the j'th coefficient of s, by querying the oracle with entries of the form $(t^d \cdot \mathrm{e}_j, 0)$, where d will vary, and e_j is the j'th vector of the canonical basis.

Lemma 1 Fix j, and suppose that, for private keys s_1 and s_2, the oracle returns the same answers for the queries $\left\{ \left(t^d \cdot \mathrm{e}_j, 0\right) \mid d = 1, \ldots, \lceil log_t q \rceil \right\}$, then the j'th entries of s_1 and s_2 coincide.

Proof Consider the real interval $I = \left[-\frac{q}{2}, \frac{q}{2}\right)$. Since I has length q, for every $x \in \mathbb{R}$, there exists a unique $s_x \in \mathbb{Z}$ such that $x + s_x \cdot q \in I$.

For every $x \in I$, it is clear that $s_x = 0$. Now subdivide I in t equal intervals

$$I_l = \left[-\frac{q}{2} + l\frac{q}{t}, -\frac{q}{2} + (l+1)\frac{q}{t} \right), \quad \text{for} \quad l = 0, \ldots, t-1.$$

For each $x \in I_l$, we have that $tx + (\frac{t-1}{2} - l)q \in I$; that is, $s_{tx} = \frac{t-1}{2} - l$. As x ranges through I, s_{tx} will take t different consecutive values, so each interval I_l will have a different class mod

t. In other words, we have seen that if for two values $x, x' \in I$, the equality $s_{tx} \equiv s_{tx'} \mod t$ holds, then they must belong to the same interval $\left[-\frac{q}{2} + l\frac{q}{t}, -\frac{q}{2} + (l+1)\frac{q}{t}\right)$ of length $\frac{q}{t}$.

Now fix some positive integer n and consider an interval $I_{l,n} := \left[-\frac{q}{2} + l\frac{q}{t^n}, -\frac{q}{2} + (l+1)\frac{q}{t^n}\right)$ with $l \in \{0, \ldots, t^n - 1\}$ (that is, one of the resulting intervals when we subdivide it in t^n equal intervals). A similar computation shows that if two points $x, x' \in I_{l,n}$ satisfy $s_{t^{n+1}x} \equiv s_{t^{n+1}x'} \mod t$, then they must lie in the same $I_{l',n+1}$ for some integer l'. Proceeding by induction, we get that if $s_{t^i x} \equiv s_{t^i x'} \mod t$ for $i = 1, \ldots, n$, then x, x' must lie in the same segment of length $\frac{q}{t^n}$. □

The main idea for the attack consists in ruling out possibilities for each coefficient (Table 3). Let us see what happens for the coefficient s_j, and then we would just need to repeat the process for each of the k components of s.

When we start the attack, we have q possibilities for each component s_j (it could be any number in $\mathbb{Z}/q\mathbb{Z}$), so we keep all options in a list. Then, we send to the oracle $\mathrm{u} = \mathrm{e}_j$ and $v = 0$. This way, we get $s_j \mod t$, so we can reduce our list of possibilities by $1/t$, removing the elements that are not the same modulo t.

After that, we send to the oracle $\mathrm{u} = t \cdot \mathrm{e}_j$ and $v = 0$. This way, we check among the elements in the previous lists which ones still coincide, i.e. which are equal when multiplied by t and after doing the corresponding modulo q computations, taken modulo t. The elements that do not coincide are removed from the lists.

We repeat this process by sending $\mathrm{u} = t^d \cdot \mathrm{e}_j$ and $v = 0$ to the oracle, for $d = 0, 1, 2, \ldots$ When all lists are of length 1 we are finished, and the elements on the lists correspond to the coefficient s_j.

Now we just have to repeat the process for every $j = 1, \ldots, k$.

On the downside, this does not allow us to differentiate between $-x$ and x when $t = 2$. In this case, we will stop the algorithm when the list is of length at most 2, since in this case, the list will be of the form $\{0\}$ or $\{-c, c\}$. Then, we take an additional step in which $\mathrm{u} = \mathrm{e}_j$ and $v = \frac{q-1}{2}$, in this case, if $s_j < 0$, the oracle will give us $s_j - \frac{q-1}{2} + q$, which changes the parity and will therefore allow us to differentiate it from $-s_j$.

Table 3 Pseudocode for the elimination attack

Elimination attack:
Input: q, t, k,$\mathcal{O}_{q,t,n}^{CCA-Hom}$.
1: **for**$j = 1, \ldots, k$:
2: $L[j] = [-\lfloor q/2 \rfloor, .., \lfloor q/2 \rfloor - 1]$
3: $l = \mathrm{length}(L[j])$
4: $d \leftarrow 0$
5: while $l \neq 1$:
6: $\mathrm{s}' \leftarrow \mathcal{O}_{q,t,n}^{CCA-Hom}(-t^d e_j, 0)$
7: **for**x**in**$L[j]$
8: **if**$x \cdot t^d \bmod q \bmod t \neq \mathrm{s}'[j]$:
9: **remove**x from $L[j]$
10: $d \leftarrow d + 1$:
11: $l = \mathrm{length}(L[j])$
12: $\mathrm{s}[j] \leftarrow L[j]$
13: returns
Output:s.

Both of these attacks could be adapted to work with Module-LWE cryptosystems, by doing the attacks for all coefficients at the same time, since the oracle would output a polynomial in R_2, we would be able to recover the secret key without needing more queries than in the integer case. This is, the number of queries needed depends only on the number k of components of s, and not on the polynomial degree of said components.

3.2 Attack on Kyber encryption scheme

Our attack works thanks to the information we get from the $\mathrm{Compress}_q$ function. For $a \in R_q$, if $a \in \left(-\frac{q}{4}, \frac{q}{4}\right)$, then $\mathrm{Compress}_q(a, 1) = 0$, otherwise $\mathrm{Compress}_q(a, 1) = 1$.

First we need to note that

$$\frac{q}{2^{10}} \cdot 128 = \frac{q}{8} = \frac{q}{2^{11}} \cdot 256,$$

so we can always choose a value for c_1 that will be decompressed into a vector whose only nonzero entry is $u_i = \left\lfloor \frac{q}{8} \right\rceil$ (in particular, for this value $\mathrm{Decompress}_q(\mathrm{Compress}_q(u_i, d_u), d_u) = u_i$). From now on, when we say that we query the decryption oracle with values (u, v), we will mean that we query them with the values that decompress to said value (u, v). More precisely, the decompression of value u will be a vector $\mathrm{u} = \left\lfloor \frac{q}{8} \right\rceil \cdot \mathrm{e}_i$, and the decompression of value v will be a polynomial with all entries equal to the corresponding value v. Then the oracle will return a polynomial with coefficients equal to 0 and 1, where each coefficient will depend only on the corresponding coefficient of s_i. That is, we can think of recovering all the coefficients of s_i in the same way we would recover a single one.

Definition 3 Let q, n, d_u, d_v be the scheme parameters. Given a secret key $\mathrm{s} \in R_q^k$, we define the CCA-oracle for CRYSTALS-Kyber Encryption-Decryption scheme as a function that takes as input $\mathrm{u} \in R_q^k$ and $v \in R_q$, and returns the result of the decipher operation:

$$\mathcal{O}^{CCA-Kyb}_{q,d_u,d_v,n}(\mathrm{u}, v) = \mathrm{Compress}_q(\mathrm{Decompress}_q(v, d_v) - \mathrm{s} \cdot \mathrm{Decompress}_q(\mathrm{u}, d_u), 1). \quad (13)$$

The oracle's responses for each possible value of a given coefficient are shown in Table 4.

By looking at the table, we can construct the following decision tree for Kyber512:

The nodes in the tree represent the queries to the oracle, while the branches marked 0 or 1 are the responses of the oracle for a given coefficient. The end leaves are the different possible values for the coefficients in s. We can see that we can always determine the values of each coefficient in 3 queries (Figs. 1, and 2).

Analogously, we can build a decision tree for Kyber768 and Kyber1024:

Since $\mathrm{s} \in R_q^k$, we will obtain each of the components of s separately. Let s_j be the j-th component of s. Then the pairs we send the oracle will be of the form $(\mathrm{u} = (0, \ldots, u, \ldots, 0), v)$, where $u = \left\lfloor \frac{q}{8} \right\rceil$ is a constant polynomial in the j-th component and v is a polynomial such that each coefficient is given by the decision tree. Then, the oracle will give us a polynomial whose coefficients are 0 or 1 and we will use the decision tree to decide the value of each coefficient of v for the next query. That is, we treat the problem for polynomials, as a paral-

Table 4 Lists of oracle responses for each possible $s \in [-3, 3]$

s \ (u, v)	$(\lfloor \frac{q}{8} \rceil, 832)$	$(\lfloor \frac{q}{8} \rceil, 1456)$	$(\lfloor \frac{q}{8} \rceil, -1456)$	$(\lfloor \frac{q}{8} \rceil, -832)$
-3	1	0	0	0
-2	1	1	0	0
-1	1	1	1	0
0	0	1	1	0
1	0	1	1	1
2	0	0	1	1
3	0	0	0	1

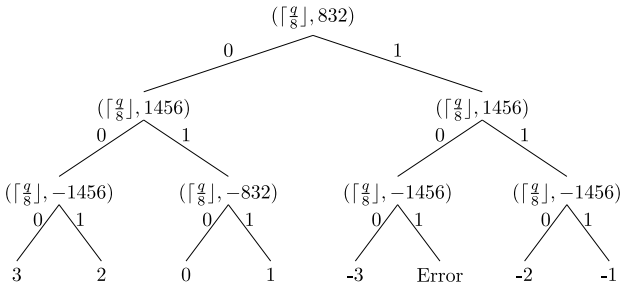


Fig. 1 Decision tree for Kyber512

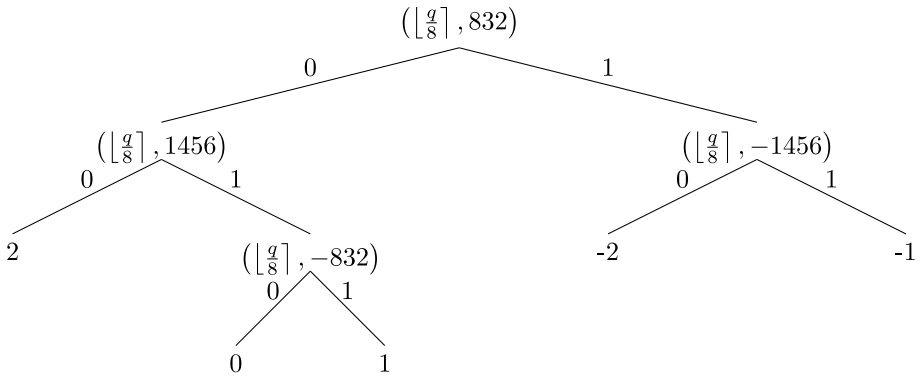


Fig. 2 Decision tree for Kyber768 and Kyber1024

lelized version of the problem for each coefficient. Then, the number of queries to get the secret s is at most $3 \cdot k$.

The complete attack for Kyber512 would be as shown in Table 5, where $p[i]$ will represent the i-th coefficient of a given polynomial p, and s^j will denote the j-th component of the secret key vector.

3.3 Attack on Kyber's CPA-KEM

The last attack we will be presenting is on Kyber's CPA-KEM. Therefore, in this case the oracle will perform a CPA version of the KEM.Dec operation in [7]. Let *H*, *G* and *KDF* be the Hash functions in Kyber's specifications, then the oracle will output a key computed as:

Definition 4 Let *sk* be the secret key, *pk* the public key and $h = H(pk)$. We denote Dec_{sk} the CPA decryption function in Kyber, which depends on d_u, d_v, and G_1 the first component of the output of the hash function G. Then the oracle is

$$\mathcal{O}^{CCA-KEM}_{q,n,Dec_{sk}}(\mathrm{u}, v) = KDF\left(G_1\left(Dec_{sk}(\mathrm{u}, v)||h\right)||H(\mathrm{u}, v)\right). \tag{14}$$

In order to perform the attack, we will fix $\mathrm{u}_j = \left\lfloor \frac{q}{32} \right\rceil \mathrm{e}_j$ the vector with a constant polynomial in the *j*-th component of **u**. Each component of the secret key is a polynomial with

Table 5 Attack on Kyber512

Kyber512 attack:
Input: q, k, $\mathcal{O}^{CCA-Kyb}_{q,d_u,d_v,n}$.
1: for $j = 1, \ldots, k$:
2: $s_0 = \mathcal{O}^{CCA-Kyb}_{q,d_u,d_v,n}(\mathrm{Compress}_q((0, \ldots, q/8, \ldots, 0)), \mathrm{Compress}_q(832(1 + x + \cdots + x^n)))$
3: $s_1 = \mathcal{O}^{CCA-Kyb}_{q,d_u,d_v,n}(\mathrm{Compress}_q((0, \ldots, q/8, \ldots, 0)), \mathrm{Compress}_q(1456(1 + x + \cdots + x^n)))$
4: $v = 0$
5: for $i = 0, \ldots, 255$:
6: if $s_0[i] == 0$ and $s_1[i] == 1$:
7: $v = v + \mathrm{Compress}_q(-832)x^i$
8: else:
9: $v = v + \mathrm{Compress}_q(-1516)x^i$
10: $s_2 = \mathcal{O}^{CCA-Kyb}_{q,d_u,d_v,n}(\mathrm{Compress}_q((0, \ldots, q/8, \ldots, 0)), v))$
11: $s^j = 0$
12: for $i = 0, \ldots, 255$:
13: if $s_0[i] == 0$ and $s_1[i] == 0$ and $s_2[i] == 0$:
14: $s^j = s^j + 3x^i$
15: if $s_0[i] == 0$ and $s_1[i] == 0$ and $s_2[i] == 1$:
16: $s^j = s^j + 2x^i$
17: if $s_0[i] == 0$ and $s_1[i] == 1$ and $s_2[i] == 0$:
18: $s^j = s^j + 0x^i$
19: if $s_0[i] == 0$ and $s_1[i] == 1$ and $s_2[i] == 1$:
20: $s^j = s^j + x^i$
21: if $s_0[i] == 1$ and $s_1[i] == 0$ and $s_2[i] == 0$:
22: $s^j = s^j - 3x^i$
23: if $s_0[i] == 1$ and $s_1[i] == 1$ and $s_2[i] == 0$:
24: $s^j = s^j - 2x^i$
25: if $s_0[i] == 1$ and $s_1[i] == 1$ and $s_2[i] == 1$:
26: $s^j = s^j - x^i$
Output: $s = (s^1, \ldots, s^k)$

256 components, so we will try to recover each polynomial by computing groups of b coefficients at a time.

For a given component s_j of s, in order to retrieve b coefficients, we note that since u is small enough, when $v = 0$ the decryption function would give 0 for all coefficients of s_j. Then, we will be able to isolate the coefficients we're interested in by sending the corresponding coefficients of v different from 0, and 0 for the coefficients we are not working with.

We can perform the attack in a non-adaptive way, in which we will need 6 queries for each group of coefficients. In this case, for each group of coefficients, we will send v the polynomial with a value a in the coefficients we are interested in and 0 in the rest of coefficients. This value a is given by the six different values in Table 6. Then we will compute the oracle's operation taking as $Dec_{sk}(\mathrm{u}, v)$ all the different possibilities of 1's and 0's in the group of coefficients for all the (u, v) we have passed to the oracle, and get one of said coefficients for each query, comparing with Table 6 until we find the value of each coefficient.

On the other hand, we can also perform the attack in an adaptive way. In this case, we can construct the binary trees as in the previous section from Table 6 in such a way that we would only need 3 queries in order to get the value of a given coefficient of s. After each query we will go over all combinations of 0's and 1's in the group of coefficients and perform the oracle operations until we get the value 1 or 0 for each coefficient, and then follow the binary tree to get the coefficients for the next polynomial v (Fig. 3).

By going through all the combinations of 0's and 1's and computing the oracle's operation, we are finding the value $m' = Dec_{sk}(\mathrm{u}, v)$ that results from the decryption function, which allows us to build a Decryption oracle from a KEM oracle for a small group of coefficients. Then we can follow binary tree adaptive attack for Kyber already explained. The binary tree for Kyber512 in this case would be:

Notice the tradeoff between the number of queries to the oracle and the number of hash operations needed: the number of queries decreases with b, whereas the number of hash operations grows exponentially.

4 Efficiency

We will measure the efficiency of the attacks according to the number of calls to the oracle needed to obtain the secret key s.

Table 6 Lists of oracle responses for each possible $s \in [-3, 3]$

s \ (u, v)	$\left(\lfloor \frac{q}{32} \rceil, 624\right)$	$\left(\lfloor \frac{q}{32} \rceil, 832\right)$	$\left(\lfloor \frac{q}{32} \rceil, 1040\right)$	$\left(\lfloor \frac{q}{32} \rceil, -1040\right)$	$\left(\lfloor \frac{q}{32} \rceil, -832\right)$	$\left(\lfloor \frac{q}{32} \rceil, -624\right)$
-3	1	1	1	0	0	0
-2	0	1	1	0	0	0
-1	0	1	1	1	0	0
0	0	0	1	1	0	0
1	0	0	1	1	1	0
2	0	0	0	1	1	0
3	0	0	0	1	1	1

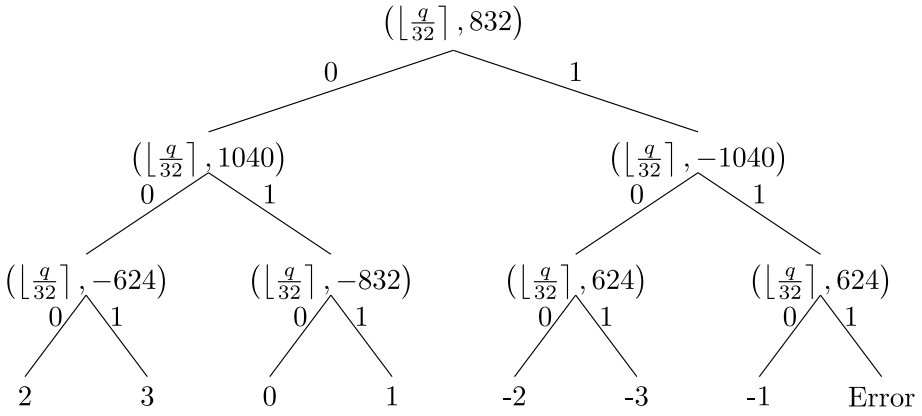


Fig. 3 Decision tree for Kyber512

For the fully homomorphic scheme, the coefficients of s can be on the whole interval $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$, so we will need at most $\lceil \log_t q \rceil$ queries to the oracle to recover each component of the secret key vector.

Notice that since the possible values for the secret key are q^k, and each query to the oracle can return t possible different values, it is not possible to distinguish all possible private keys with less than $\left\lceil \frac{k \log q}{\log t} \right\rceil$ queries. Our attack requires $k \left\lceil \frac{\log q}{\log t} \right\rceil$ which is at most $k-1$ higher than the theoretical optimum. In particular, for $k = 1$, it is the theoretical optimum.

For Kyber's encryption scheme, we have seen that we can obtain each of the components of the secret key in at most 3 queries to the oracle, therefore, in order to recover the full secret we will need $3k$ queries to the oracle. That is, for Kyber512 we need a total of 6 queries, for Kyber768 we need 9 and for Kyber1024 we need 12 queries. We can compute the theoretical optimum by counting the possible values of the private key and the oracle answers as before for each of the security levels. The results are $\lceil 2 \log_2 7 \rceil = 6$, $\lceil 3 \log_2 5 \rceil = 7$ and $\lceil 4 \log_2 5 \rceil = 10$ for Kyber512, Kyber768 and Kyber1024 respectively. Notice how our attack is optimal for Kyber512, and only 2 queries worse than the theoretical optimum in the other two cases.

Lastly, for the attack on Kyber's KEM, as it also follows the trees we built for Kyber's decryption scheme, we will only need 3 queries to the oracle to determine each group of b coefficients. Therefore, we will only need $3 \cdot k \cdot \frac{256}{b}$ queries. For example, when $b = 16$, we will need 96, 144 and 192 queries for Kyber512, Kyber768 and Kyber1024 respectively. The bigger we take b, the smaller the number of queries needed will be, on the other hand, when b grows, the number of combinations of 0's and 1's grows exponentially (Table 7), so finding the combination that gives as the same key as the oracle takes a lot more time and the computational cost of performing the hash functions on the combinations grows fast. For example, if we tried to compute the whole secret key in one query, we would have a combinations set of $2^{256 \cdot k}$ elements to go through.

Table 7 Number of Hash function operations and queries for different values of b

$b =$	1		2		4		8		16	
	Q	H	Q	H	Q	H	Q	H	Q	H
Kyber512	1536	3072	768	3072	384	6144	192	49152	96	6291456
Kyber768	2304	4608	1152	4608	576	9216	288	73728	144	9437184
Kyber1024	3072	6144	1536	6144	768	12288	384	98304	192	12582912

5 Comparison with other attacks

As we have previously mentioned, there has been previous work attacking CPA-secure lattice based KEMs by means of Key Mismatch attacks and CPA^D attacks. The former use an oracle that simulates Alice's KEM. This oracles output 1 or 0 depending on wether the obtained key coincides with the one the attacker obtains or not, and Key Mismatch attacks use this information to recover Alice's secret key. On the other hand, we propose CCA-attacks that call an oracle that simulate the decryption function that Alice uses as part of the decapsulation process, instead of the complete decapsulation function (that is, it requires access to a stronger oracle); and later we propose a CCA-attack on a CPA version of CRYSTALS-Kyber.

In [2], Qin et al. calculate the average number of queries to the oracle needed to obtain the secret key s for different NIST candidates. In particular, for Kyber512, Kyber768 and Kyber1024 the average number of queries is 1312.06, 1774.08 and 2365.44 respectively; meanwhile, applying our CCA-attack, with the appropriate choice of ciphertext, the number of queries needed are 6, 9 and 12 respectively, a number considerably smaller than for the Key Mismatch attacks (Table 8).

We have implemented our attack using SageMath [9], and computed 1000 secret keys, that we were able to recover correctly in the number of theoretical queries. This is, our attack reduces the number of queries by 99.75%.

As we have already commented, in [1] the CPA^D attacker has access to an oracle that performs encryption, evaluation and decryption of outputs of previous functions and tries to find the secret key of a fully homomorphic encryption scheme. The decryption function involves a computation modulo 2, so we can set $t = 2$. The parameters used are $q = 2^{240}$ and $n = 8192$ or 16384 in order to obtain a proportion of ciphertexts for which the noise amplitude is approximately 1, and the total number of queries is of the order of 2^{23} and 2^{24} respectively. Meanwhile, applying our CCA-attack with the same q, since our attack works in $k \left\lceil \frac{\log q}{\log t} \right\rceil$ queries, we would be able to obtain the same secret key in $240 \cdot k$ queries (Table 9). In the case of [1], s is taken in $\mathbb{Z}_q^n$, so in this setting we would take $k = n$. After implementing our attack on SageMath [9], and computing 1000 secret keys, we were able to recover each one on an average of $240n$ queries , this is, we reduce the number of decryption queries by approximately 69.81%.

Table 8 Comparative between Key Mismatch and CCA attack for Kyber

	Key Mismatch theoretical queries	Key Mismatch experiment queries	CCA-attack theoretical queries	CCA-attack experiments queries
Kyber512	1312	1311	6	6
Kyber768	1774	1777	9	9
Kyber1024	2365	2368	12	12

Table 9 Comparative between CPA^D and CCA attack for FHE

		CPA^D			CCA-attack		
n	q	Encryption queries	Evaluation queries	Decryption queries	Encryption queries	Evaluation queries	Decryption queries
8192	2^{240}	14593	6512218	6512218	0	0	1966080
16384	2^{240}	29016	12924174	12924174	0	0	3932160

6 Conclusions

We present a CCA attack against schemes based on the LWE problem. In particular, against the fully homomorphic encryption schemes presented in [8] and [1], and the CRYSTALS-Kyber key encapsulation scheme [7]. The attacks recover the secret key using a limited (close to optimal) number of queries to the decryption function.

This showcases the importance of not exposing the decryption function to a potential adversary in the used implementations. In particular, implementations of CRYSTALS-Kyber in devices where an attacker might gain access (such as smartcards or Yubikeys) should make sure that their interface does not expose the PKE decryption function, since it would make them vulnerable to exfiltration of the secret keys. Similarly, the key decapsulation function should follow tightly the specificaction, and perform all the corresponding checks.

Author Contributions A.L and M.M wrote the main manuscript text. A.H and A.L wrote the code that was used for experimentation, M.M reviewed and improved said code. All authors reviewed the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Checri, M., Sirdey, R., Boudguiga, A., Bultel, J.-P.: On the practical cpa d security of "exact" and threshold fhe schemes and libraries. In: Annual International Cryptology Conference, pp. 3–33. Springer (2024)
2. Qin, Y., Cheng, C., Zhang, X., Pan, Y., Hu, L., Ding, J.: A systematic approach and analysis of key mismatch attacks on lattice-based nist candidate kems. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV 27, pp. 92–121. Springer (2021)

3. Ding, J., Cheng, C., Qin, Y.: A simple key reuse attack on lwe and ring lwe encryption schemes as key encapsulation mechanisms (kems). Cryptology ePrint Archive (2019)
4. Qin, Y., Cheng, C., Ding, J.: An efficient key mismatch attack on the nist second round candidate kyber. Cryptology ePrint Archive (2019)
5. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM (JACM) **56**(6), 1–40 (2009)
6. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29, pp. 1–23. Springer (2010)
7. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehle, D.: Crystals-kyber–algorithm specifications and supporting documentation. NIST Technical Report (2019)
8. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM J. Comput. **43**(2), 831–871 (2014)
9. Alba Hernandez Costoya, M.A.M.d.B. Alba Larraya Sancho: CCA-Kyber-attack. Github (2025)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.