Lattice-Based Cryptography: Development and Analysis of a New Variant of the Crystals-Kyber Algorithm

Mauricio Sebastian Cisneros Laule
smlaulet@gmail.com
https://orcid.org/0009-0007-4056-4467
Universidad de Lima, Peru

Javier Enrique Olazabal Silva
javierenriqueos@gmail.com
https://orcid.org/0009-0003-2728-8614
Universidad de Lima, Peru

Hernan Nina Hanco
hninaha@ulima.edu.pe
https://orcid.org/0000-0003-0230-5812
Universidad de Lima, Peru

Received: September 8th, 2024 / Accepted: October 12th, 2024
doi: https://doi.org/10.26439/interfases2024.n020.7383

ABSTRACT. The imminent arrival of quantum computing has accelerated the need for cryptographic systems resistant to quantum attacks. Such attacks exploit the vulnerability in private and public key encryption systems, where the public key is derived from the private key, which could be refactored from the public key. To address this issue, the National Institute of Standards and Technology (NIST) launched a global competition in 2016 to create quantum-resistant algorithms. CRYSTALS-Kyber, a lattice-based algorithm focused on the learning with errors (LWE) problem, was selected for standardization. This work introduces RKyber, a variant that instead targets the learning with rounding (LWR) problem, simplifying computations by using deterministic errors rather than random noise. Both algorithms were executed 1000 times, showing that RKyber offers improved speed at the cost of some security.

KEYWORDS: post-quantum / lattice-based / quantum computing / Kyber / quantum cryptanalysis

M. S. Cisneros, J. E. Olazabal, H.Nina

CRIPTOGRAFÍA BASADA EN RETÍCULAS: DESARROLLO Y ANÁLISIS DE UNA NUEVA VARIANTE DEL ALGORITMO CRYSTALS-KYBER

RESUMEN. La inminente llegada de la computación cuántica ha hecho necesario el desarrollo de sistemas criptográficos resistentes a los ataques cuánticos. Los ataques cuánticos explotan la debilidad de la encriptación de llave pública y privada, la cual radica en que la llave pública es derivada desde la llave privada y esta última podría ser factorizada a partir de la llave pública. En respuesta, el NIST inició un concurso mundial en 2016 para crear algoritmos resistentes a la computación cuántica. CRYSTALS-Kyber, un algoritmo basado en celosía que aborda el problema de Aprendizaje con Errores fue seleccionado para su estandarización. Este trabajo introduce una variante, RKyber, que en su lugar aborda el problema de Aprendizaje con Redondeo, simplificando los cálculos mediante el uso de errores deterministas en lugar de ruido aleatorio. Ambos algoritmos se ejecutaron 1000 veces, demostrando que RKyber es más rápido, aunque sacrifica algo de seguridad.

PALABRAS CLAVE: poscuántico / basado en retículas / computación cuántica / Kyber / criptoanálisis cuántico

INTRODUCTION

This paper aims to research and demonstrate a modification to the fundamental problem solved by the Kyber algorithm. Presenting this research is important, as it explores the ramifications derived from modifying the system's core problem from learning with errors (LWE) to learning with rounding (LWR). Quantum computing is widely recognized as a threat to most cryptosystems in use today and LWR is known to be computationally comparable to LWE, meaning that breaking the encryption of one should be as challenging as decrypting the other. However, no research has specifically addressed this modification within the newly standardized CRYSTALS-Kyber algorithm. This knowledge gap has therefore been selected as a focal point for this study.

This paper provides the necessary background to understand the topic, as well as the algorithmic demonstration and brute-force tests conducted on the modified algorithm named RKyber (short for "Rounded Kyber"). The first section covers the theoretical framework, establishing the essential background. This is followed by a description of the methodology and experimental design. The final sections discuss the experimental results, provide a brief analysis, and present the final conclusions of the research.

BACKGROUND

2.1 Quantum Computing

Quantum computing is a paradigm that has seen recent advancements at the hardware level, though research in this field has been ongoing for about 50 years. It began with Stephen Wiesner's concept of conjugate coding (Mor & Renner, 2014), a system in which multiple messages were transmitted, but reading one would destroy the others. In the following years, Benioff (1980) described the Turing machine using Schrödinger's equation and created a model for a quantum computer in 1982 (Mor & Renner, 2014). Based on this model, mathematician Peter Shor and computer scientist Lov Grover each developed a famous algorithm: Shor's algorithm, which uses quantum Fourier transform (QFT) to break many current encryptions (Shor, 1997), and Grover's algorithm, which significantly accelerates database searches (Grover, 1996).

The basis of this paradigm is the qubit, which primarily focuses on superposition and error correction. Superposition allows a single qubit to represent two values at the same time. As shown in Figure 1, qubits are represented on "Bloch spheres," with two opposing poles corresponding to "1" or "0" and a vector pointing to an arbitrary position on the sphere. Depending on the vector's direction, there is a higher or lower probability of receiving a "1" or "0" when measuring the qubit. This probability distribution must follow the linear equation presented in Equation (1):

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \qquad (1)$$

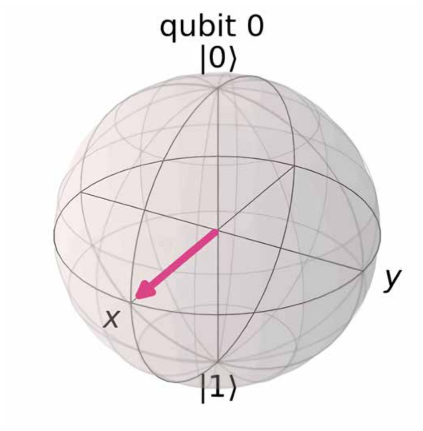
Where ψ is the probability column vector and must meet the limits of Equation (2):

$$|\alpha|^2 + |\beta|^2 = 1 \qquad (2)$$

Where α represents the probability of a 0 measurement and β a 1 measurement.

Figure 1

Representation of a Qubit with a Bloch Sphere



The goal of this technology is to enhance processes currently considered efficient, surpassing the limits known in classical computing. Aaronson and Chen's (2017) complexity theory discuss the concept of quantum supremacy, which is expected to be achieved soon, especially in the field of cryptography. In response to this, algorithms designed to resist the impending threat of quantum computing have been developed. One of these resistant algorithms or "post-quantum" cryptographic algorithms is the CRYSTALS-Kyber algorithm, which has successfully passed numerous tests and is on the path toward standardization by the National Institute of Standards and Technology (NIST) (Avanzi et al., 2021). This algorithm addresses the LWE problem, which is based on the difficulty of solving linear equations in a finite field with noise (Peikert, 2016; Lyubashevsky et al., 2010).

2.2 Lattice-Based Identity

The lattice-based family of algorithms focuses on addressing specific problems that define their particularities, namely the closest vector problem (CVP), shortest vector problem (SVP), and shortest independent vector problem (SIVP). All solutions to these problems are considered during the generation of vector sets used to create points within the finite field containing all lattices (Peikert, 2016). Lattices are defined as a discrete subgroup L of $\mathbb{R}^{\mathrm{n}}$, where L contains the linear combination of vectors, as seen in Equation (3):

$$av_1 + bv_2 + \cdots + kv_n \quad \text{with} \quad a, b, \cdots, k \in Z \;\; \wedge \;\; v_1, v_2, \cdots, v_n \in L \qquad (3)$$

The base of the lattice is usually represented in the form shown in Equation (4):

$$\sum_{i=1}^{r} a_i \, v_i \qquad a_i \in \mathrm{Z} \tag{4}$$

Where r refers to the rank or dimension of the lattice. If r equals the dimension n, it is considered a full-rank lattice (Micciancio & Goldwasser, 2002). This definition is represented with a matrix V that generalizes the dimensional vectors, as shown in Equation (5):

$$V \epsilon R^{nxr} \,,\; L = \{Vx \,,\, x \epsilon Z^{r}\} \tag{5}$$

In this context, Vx represents the linear combination of vectors generating points collectively known as a lattice. These combinations are the solutions provided by the algorithms for the presented problems. From a cryptographic perspective, these solutions are used to encrypt and send messages. During this process, two sets of vectors of the same rank are generated, and their linear combination creates the same points in a given space. One set is designated as the public key and the other as the private key. The private key will have a combination that is easier to calculate to reach a point i, and it should only be accessible to the user (Micciancio & Goldwasser, 2002).

This public–private key cryptographic method is used in message exchange. When user A sends a message to user B, user A employs user B's public key to encrypt the message, allowing user B to decrypt it with their own private key. In lattice-based cryptography, the message is assigned by user A employing user B's public key to a point in the subgroup L_B (Lyubashevsky et al., 2010).

2.3 CRYSTALS-Kyber – Key Generation

The Kyber algorithm utilizes vector principles. A relatively simple base vector is used as the private key s and a derived public key t. Public key generation requires a matrix of random polynomials A of the same dimension as the private key and an error vector e (Avanzi et al., 2021). This error vector is the proposed solution to the LWE problem. They are defined as shown in Equation (6):

$$s = \left(a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0 \,;\, b_n x^n + b_{n-1}x^{n-1} + b_0 \;;\, k_n x^n + k_{n-1}x^{n-1} + \cdots + k_0\right) \tag{6}$$

Where the coefficients a, b, ..., k belong to the integers in dimension n ($\mathbb{Z}^n$). Likewise, the matrix A is defined as shown in Equation (7):

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, a_n m = bx^n + cx^{n-1} + \cdots + zx^0, b, c, \cdots, z \epsilon \mathbb{Z}^n \tag{7}$$

The error vector *e* is defined in the form shown in Equation (8):

$$e = (ax^n + b^{n-1} + \ldots + k),\ \mathrm{a,b,\ldots,k} \in Z^n \qquad (8)$$

This results in the derivation of the public key vector t, defined in Equation (9):

$$t = As + e \qquad (9)$$

The key generation algorithm works as shown in Algorithm 1, where coefficients a are restricted to values between -3 and 3, and the parameter q is the commonly used modulo 3329 (Avanzi et al., 2021).

```
Algorithm 1: Key Generation
1.  Start
2.      Generate key s
3.          $\mathrm{S} = a_i x^n + a_{i-1}x^{n-1} + \cdots + a_1 x^0 where a_i \in [-3.3]$
4.      Assign value to q
5.          $\mathrm{q} = 3329$
6.      Generate matrix A
7.          $\mathrm{A} = [a11(X)\ a21(X)\ \ldots ann(X)]$
8.      Generate public key t
9.          $\mathrm{T} = \mathrm{As} + \mathrm{e}$
10. Return (s, t, A)
11. End
```

2.4 CRYSTALS-Kyber – Encryption

The encryption method of the Kyber algorithm works as follows. A random vector of polynomials *r* is generated, matching the dimension of the previous keys, along with two error vectors, e_1 and e_2. The polynomials in these vectors are relatively small, similar to the vector *s* (Avanzi et al., 2021). For encryption, the message is transformed into its binary representation, where each bit *n* of the message is used as a coefficient, as shown in Equation (10):

$$m_b = n_1 x^k + \ldots + n_i\ ,\ \ \mathrm{n} \in \{1;0\} \qquad (10)$$

The polynomial presented in Equation (11) is multiplied by the nearest integer to the quotient, resulting in the polynomial m. This polynomial m is then encrypted using the public key (*A, t*), resulting in the values (*u, v*).

$$m = m_b \cdot \frac{q}{2} \qquad (11)$$

Where the polynomial u is defined as shown in Equation (12):

$$u = A^T r + e_1 \qquad (12)$$

And the polynomial v is defined in Equation (13):

$$v = t^T r + e_2 + m \tag{13}$$

The encryption process is demonstrated in Algorithm 2. This algorithm first asks whether the input value is a string or an integer. A constant n, which in this case is the dimension used for encryption, is assigned a value of 256 (Avanzi et al., 2021).

```
Algorithm 2: Encryption
1.  Start
2.     Input value x to encrypt
3.     If x is integer:
4.        Transform it to binary value $(x)_{10} = (x)_2$
5.        With $(x)_2$ : $m_b = \mathrm{P}(x) = b_n x^n + b_{n-1}\,\mathrm{x}^{n-1} + \cdots + b_0 x^0$  where $b_n \in \{0.1\}$
6.        $m = m_b * \frac{q}{2}$
7.     Else if x is string:
8.        Transform to binary value $(x) = (x)_2$
9.        If elements in $(x)_2 > n$:
10.          Create blocks where the elements of $(x)_2 <= n$ where $n = 256$ and add to list L
11.    Generate random vector (r)
12.    Calculate values u, v :
13.       $\mathrm{u} = A^T \mathrm{r} + e_1$
14.       $\mathrm{v} = t^T \mathrm{r} + e_2 + \mathrm{m}$
15.    Return (u, v)
16. End
```

2.5 CRYSTALS-Kyber – Decryption

The decryption method uses the private key s and the resulting polynomials (*u, v*). The remainder will be called m_n, which will be noisy due to the error vectors generated in the previous steps that modify the result. The resulting coefficients in m_b are compared with the value of $q/2$, and if a coefficient is closer to this value, it will be replaced with it. If it is closer to 0 or q, it will be replaced with 0. The result is then divided by $q/2$, and the coefficients become the original bits of the sent message (Avanzi et al., 2021). See Equation (14):

$$m_n = v - s^T u \tag{14}$$

Expanding the result, m_n is presented in Equation (15):

$$m_n = e^T \mathrm{r} + e_2 + \mathrm{m} + s^T e_1 \tag{15}$$

Finally, the coefficients n_k make up the sent message, represented by its bits, as shown in Equation (16):

$$m_b = \frac{1}{\frac{q}{2}}\left(n_1 x^n + n_2 x^{n-1} + \ . \ . \ . \ + n_k x^0\right) \ , \ n_k \epsilon \left\{0\,;\ \frac{q}{2}\right\} \tag{16}$$

The decryption process is demonstrated in Algorithm 3 where the conditional executes for each coefficient *i* on the noisy value m_n. Finally, it returns the *res* which would be the original encrypted message (Avanzi et al., 2021).

Algorithm 3: Decryption

```
1.  Start
2.      Input values to decrypt (u,v)
3.      Calculate noisy value
4.          $m_n = v - s^t u$
5.      For each coefficient (i) of $m_n$ :
6.          If $\left|i - \frac{q}{2}\right| < |i - q|$ o $\left|i - \frac{q}{2}\right| < |0 - i|$ :
7.              Then $i = \frac{q}{2}$
8.          Else:
9.              Then $i = 0$
10.     Calculate original value:
11.         $res = m_b * \frac{1}{\frac{q}{2}}$
12.         Return res
13. End
```

The CRYSTALS-Kyber algorithm addresses the LWE problem from the lattice-based algorithm family. Additionally, there is a main extension to this problem known as LWR (Alwen et al., 2013). The main difference between the LWR and LWE cryptosystems is shown in Equation (9). The lack of the error vector and the addition of the parameter p results in a key as shown in Equation (17):

$$t = \left[\frac{p}{q} As\right] \qquad (17)$$

The cryptosystem built around the LWR problem assumes that the complexity of reducing the quotient product of the parameters p and q, matrix A, and vector s is sufficient to withstand quantum attacks. Therefore, the error vector can be removed without compromising security (Alwen et al., 2013).

2.6 LWE, LWR, and Reverse Engineering Methodology

The LWE problem enhances encryption security by adding pseudo-random noise to polynomial rings, making it challenging to solve without the private key. In contrast, LWR simplifies the process by removing random noise and using deterministic rounding errors, which makes the algorithm more efficient while maintaining security (Lyubashevsky et al., 2010; Regev, 2005). If the ratio between p and q is large enough, LWR provides computational improvements without significantly compromising the security.

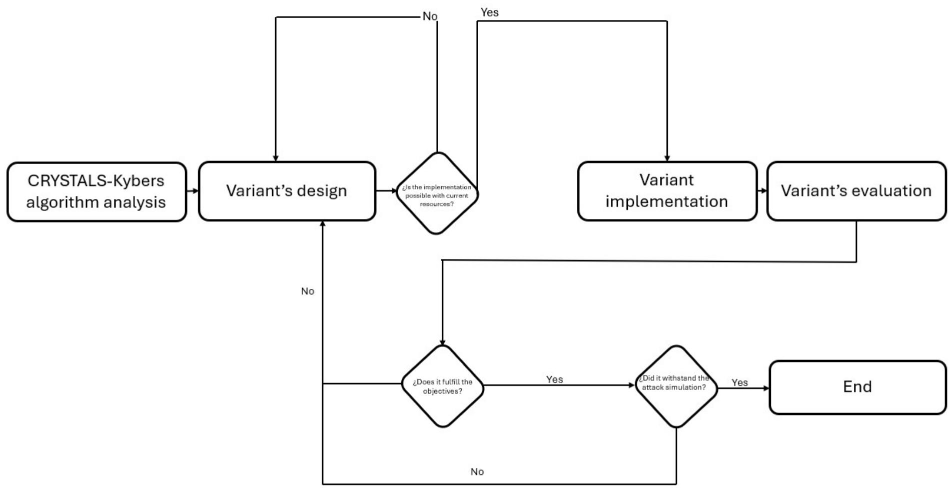
In 2024, researchers from NTT DATA demonstrated a reverse engineering attack on algorithms like Kyber. Their method reduces the number of polynomial calculations by exploiting known public key parameters and minimizing the possible values of the error vector (NTT DATA Perú, 2024). Using regression and projection techniques, this approach estimates private key values based on the time taken to break the encryption. Although not a direct attack on the encryption itself, this method narrows down potential key values by limiting the error vectors to a few discrete options, thereby speeding up brute-force attacks.

3. METHODOLOGY

The methodology follows the structured process shown in Figure 2. First, the CRYSTALS-Kyber algorithm was analyzed to identify areas suitable for modification. After validating the resources, a prototype of the variant was designed and implemented. The final step involved executing brute-force attacks to evaluate the security of both the original algorithm and its variant.

Figure 2

Methodology Followed in the Development of the RKyber Variant


CRYSTALS-Kybers algorithm analysis
Variant's design
¿Is the implementation possible with current resources?
No
Yes
Variant implementation
Variant's evaluation
No
¿Does it fulfill the objectives?
Yes
¿Did it withstand the attack simulation?
Yes
End
No

3.1 CRYSTALS-Kyber Algorithm Analysis

To start, an analysis of the original CRYSTALS-Kyber algorithm was conducted to identify specific areas requiring modification in order to achieve the proposed variant design. The algorithm presented in the previous section consists of multiple steps to accomplish its task of protecting messages. For simplification, these steps are divided into three methods. The first method involves generating public and private keys, the second handles encryption, and the third manages decryption (Avanzi et al., 2021).

The authors of the algorithm propose a security solution that addresses the LWE problem (Avanzi et al., 2021). This involves adding noise or errors to a point or lattice within the discrete group L. In this case, the researchers added random error vectors e, e_1, and e_2 across the different methods. These error vectors add security by slightly obfuscating the encrypted message (Avanzi et al., 2021; Lyubashevsky et al., 2010), but they also increase the computational cost and, consequently, the execution time. It should be noted that these vectors create small and controlled errors, meaning that if the algorithm's steps are followed, the message will never be lost during execution.

3.2 Variant Design and Implementation

The proposed variant involves taking the base algorithm and addressing another problem from the lattice-based family. While the CRYSTALS-Kyber algorithm solves the LWE problem (Wei et al., 2023), this variant focuses on addressing its extension, LWR (Alwen et al., 2013). The LWR problem centers around the idea that the security provided by polynomial vectors in key creation is sufficient, and that the additional computational cost introduced by polynomial error vectors can be reduced by replacing them with deterministic errors generated by the parameters p and q (Alwen et al., 2013). As a result, the following equations were modified to remove the error vectors. Equation (9) takes the form presented in Equation (18):

$$\mathrm{t} = \left[\frac{p}{q}As\right] \tag{18}$$

The polynomial u is modified as shown in Equation (19):

$$\mathrm{u} = \left[\frac{p}{q}A^T r\right] \tag{19}$$

The polynomial v is modified as shown in Equation (20):

$$\mathrm{v} = \left[\frac{p}{q}t^T r\right] + \mathrm{m}\left[\frac{q}{2}\right] \tag{20}$$

The noisy result m_n takes the form shown in Equation (21):

$$m_n = v - us \tag{21}$$

Finally, the result must be rescaled to obtain the original value, as presented in Equation (22):

$$m_b = \left[\frac{2\,m_n}{q}\right] mod\,2 \tag{22}$$

Python was selected for implementing the variant due to its suitability for polynomial and matrix calculations, which simplified the mathematical operations required by these equations (Gonzalez, 2021). Additionally, the tests to verify the effectiveness of the algorithm were also conducted in the same programming language.

The three modified algorithms are presented in Algorithm 4, Algorithm 5, and Algorithm 6. In Algorithm 4, key generation is shown, where the keys s and t must be generated along with matrix A. These functions receive global parameters q and n, which represent the space and maximum dimension, respectively. During this first phase, the value of p is generated, which —together with q— defines the rounded space for the system.

```
Algorithm 4: Key Generation
1.  Start
2.     Generate key s
3.        $S = a_i x^n + a_{i-1}x^{n-1} + \cdots + a_1 x^0$ where $a_i \epsilon[-3.3]$
4.     Generate space value p
5.        $q = 3329$
6.        $P = k$ while $k < q$
7.     Generate matrix A
8.        $A = [a_{11}(X) a_{21}(X) \dots a_{nn}(X)]$
9.     Generate public key t
10.       $T = \left[\frac{p}{q} * (s * A)\right]$
11. Return (s, t, A)
12. End
```

In Algorithm 5, the encryption process is shown. This involves handling both text strings and numbers. The initial transformation step requires converting the input message into its binary form. In the case of numbers, this is relatively straightforward. However, for text, the number generated must be checked to ensure it does not exceed the established dimension *n*. If it does, it is divided into blocks with elements smaller than or equal to dimension *n*. Subsequently, the operations are carried out, generating both u and v along with a pseudo-random vector *r*. The encryption process concludes by returning the encrypted pair (*u,v*).

```
Algorithm 5: Encryption
1.  Start
2.     Input value x to encrypt
3.     If x is integer:
4.        Transform it to binary value $(x)_{10} = (x)_2$
5.        With $(x)_2$ : $m_b = P(x) = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_0 x^0$ where $b_n \epsilon \{0.1\}$
6.        $M = m_b * \frac{q}{2}$
7.     Else if x is string:
8.        Transform to binary value $(x) = (x)_2$
9.        If elements in $(x)_2 > n$:
10.          Create blocks where the elements of $(x)_2 <= n$ where $n = 256$ and add to list L
11.    Generate random vector (r)
12.    Calculate values u, v :
13.       $u = \left[\frac{p}{q} A^T r\right]$
14.       $v = \left[\frac{p}{q} t^T r\right] + m\left[\frac{q}{2}\right]$
15.    Return (u, v)
16. End
```

Finally, in Algorithm 6, the decryption process is shown. Here, the established keys are considered as global variables, while the encrypted pair (*u,v*) is given as an argument. The noisy result m_n is calculated. For each coefficient *i*, the distance is compared with *q/2* , *q*, and 0. If *i* is closer to *q/2*, it will be assigned that value; otherwise, if its distance to *q* or 0 is smaller, it will be assigned 0. The result is then scaled back to retrieve the original value.

Algorithm 6: Decryption

```
1.   Start
2.      Input values to decrypt (u,v)
3.      Calculate noisy value
4.         $m_n = \mathrm{v} - \mathrm{u} * s$
5.      For each coefficient (i) of $m_n$ :
6.         If $\left|i - \frac{q}{2}\right| < |i - q|$ o $\left|i - \frac{q}{2}\right. < |0 - i|$ :
7.               Then $\mathrm{i} = \frac{q}{2}$
8.         Else:
9.               Then $\mathrm{i} = 0$
10.     Calculate original value:
11.        $res = m_b * \frac{1}{\frac{q}{2}}$
12.        Return res
13.  End
```

3.3 Experimental Design

The first step in conducting the experiment was to implement the algorithm based on the formulated equations. Matrix and vector operations were managed using the NumPy library. Once the implementation was completed, each stage of the algorithm was tested to verify that it used the necessary modified and added parameters to consider the change toward LWR (Alwen et al., 2013). The second step involved implementing a brute-force attack method using reverse engineering as presented by NTT DATA (NTT DATA Perú, 2024).

The original Kyber algorithm and its RKyber variant were refactored and implemented respectively in Python. The time in seconds required to generate both public and private keys was recorded for dimensions (*n*) ranging from 2 to 13. Using this data, the exponential function was applied to project values for larger dimensions, extending up to 256, which is the official value for the original versions (Gonzalez, 2021). The execution times for each method of both algorithms were compared. Finally, it is worth noting that the implementation of the original algorithm followed the official NIST documentation for the competition (Avanzi et al., 2021), while the modification was based on the work by Alwen et al. (2013). The algorithm was executed on a machine with the following specifications: Core i5-11300H processor, 24GB RAM, Windows 11 operating system, and GeForce RTX 3050 GPU.

3.4 Variant Evaluation

Once the implementation of the variant was completed, we focused on comparing the execution times for each part of the algorithm: key generation, encryption, and decryption. This evaluation involved executing each part of the algorithm 1000 times for both the original Kyber algorithm and its variant. The primary objectives of the variant were to improve computational resource efficiency compared to the original algorithm. The decryption process necessarily required equations resulting in u and v. It was essential to perform all calculations related to key generation and encryption. The public key takes the form presented in Equation (9), which can be expressed as shown in Equation (23):

$$\begin{bmatrix} a_{11}(X) & a_{12}(X) \\ a_{21}(X) & a_{22}(X) \end{bmatrix} \begin{bmatrix} s_1(X) \\ s_2(X) \end{bmatrix} + \begin{bmatrix} e_1(X) \\ e_2(X) \end{bmatrix} \tag{23}$$

Where the elements of the matrices are generated based on *X*, with each element of matrix *A* being a negacyclic matrix (Gonzalez, 2021). This expression applies to Kyber512, where the coefficients of $s_i(X)$ are only restricted to {-3, -2, -1, 0, 1, 2, 3}. This range generates 7^{512} possibilities for the private key. In the case of Kyber768, it involves a 3x3 matrix ***A***, and for Kyber1024, a 4x4 matrix ***A*** (Gonzalez, 2021). Considering that the error vector is removed in the RKyber variant, and this vector dimension increases with each Kyber version, the estimated security loss is proportional to the number of polynomials removed. For RKyber512, the public key is calculated as presented in Equation (24):

$$\left\lfloor \frac{p}{q} \begin{bmatrix} a_{11}(x) & a_{12}(x) \\ a_{21}(x) & a_{22}(x) \end{bmatrix} \begin{bmatrix} s_1(x) \\ s_2(x) \end{bmatrix} \right\rfloor \tag{24}$$

Given the numerous private key possibilities, the corresponding public key becomes unfathomably large. In a reduced example, where values are divided into two polynomials for the key ***s***, the number of possibilities was calculated to be 7^6. In this same example, the remaining polynomials would need to be assigned to matrix ***A***, requiring 7^{500} calculations. This results in a total of 7^{512} calculations. Based on this estimate, the projected security loss for RKyber512 is 1.17 %. For RKyber768 and RKyber1024, the estimated loss is 0.7 % and 0.5 %, respectively, assuming that the security loss is directly proportional to the number of removed polynomial operations.

The time required to break the encryption was estimated using reverse engineering and regression analysis. Following NTT researchers (NTT DATA Perú, 2024), this reverse engineering approach minimizes all the possible variables in the key generation method. The number of possible values was reduced from 7^6 to 3^6 by using only the values {-1, 0, 1}. In this method, matrix *A* is known, so only the private key **s**, divided into $\mathbf{s_1}$ and $\mathbf{s_2}$, needs to be addressed. Moreover, the parameters ***p*** and ***q*** are known, as they are defined at the start of the execution. The results are derived as presented in Equation (25):

$$\left\lfloor \frac{p}{q}\left(A_{1,1}\, s_1 + A_{1,2} s_2\right) \right\rceil = t_1$$
$$\left\lfloor \frac{p}{q}\left(A_{2,1} s_1 + A_{2,2} s_2\right) \right\rceil = t_2 \tag{25}$$

In the original algorithm, error vectors were included in the key generation process, resulting in 3^6 possibilities (Avanzi et al., 2021). However, since this variant removes the error vector, the number of possibilities is further reduced to 3^4.

4 RESULTS

4.1 Analysis Results

Starting with the original CRYSTALS-Kyber algorithm, the first method involves generating the matrix *A*, the private key ***s***, the error vector ***e***, and calculating the public key ***t***. Since generating matrix ***A*** is the most complex operation in terms of time and space, it dictates the complexity of the key generation method, with a time complexity of $O(n^2)$ and a space complexity of $O(n^2 \log q)$. Here, *q* refers to the space in which the polynomial coefficients are generated. The second method, encryption, generates vector *r* and the pair of encrypted message values *u* and *v*. The space complexity for this method is $O(n \log q)$, while the time complexity is $O(n^2)$, again generated in space *q*. Finally, the decryption method runs in $O(\log q)$ space and $O(n)$ time, as reversing the message is relatively simpler to store and execute.

In the RKyber variant, no error vector ***e*** is generated. Instead, it works with an additional parameter ***p***, which creates a "deterministic error" in place of the pseudo-random error seen in LWE. The first key generation method runs with $O(n^2 \log q)$ space and $O(n^2)$ time. However, in this case, the public key is generated within space ***p*** rather than *q*, as seen in Equation (18). The second encryption method has the same time and space complexity as its LWE counterpart, with $O(n \log q)$ for space and $O(n^2)$ for time. The main difference lies in how the vectors *u* and ***v*** are generated in space *p* rather than *q*. Finally, the decryption method runs in $O(\log p)$ space and $O(n)$ time.

4.2 Experimental Results

4.2.1 Execution Time Tests

The results from the experiment consist of the execution times for each part of both algorithms, averaged over 1 000 runs per stage. The Kyber algorithm has three versions in terms of security: Kyber512 (lower security, faster), Kyber768, and Kyber1024 (higher security, more time-consuming). The following tables show the execution times for both the original algorithm and the RKyber variant across Kyber512, Kyber768, and Kyber1024.

Table 2

Execution Times in Milliseconds for RKyber512 and Kyber512

RKyber 512				Kyber512		
Time per Stage (in ms)	Keygen	Encryption	Decryption	Keygen	Encryption	Decryption
Maximum	12.002	27.526	46.002	17.010	22.002	55.999
Minimum	3.997	7.945	14.225	4.032	7.998	14.612
Median	5.430	9.420	16.181	5.611	9.723	16.658
Standard Deviation	0.804	1.068	1.887	0.881	1.197	2.039
Total	5.43	9.42	16.18	5.6	9.72	16.65

Table 3

Execution Times in Milliseconds for RKyber768 and Kyber768

RKyber768				Kyber768		
Time per Stage (in ms)	Keygen	Encryption	Decryption	Keygen	Encryption	Decryption
Maximum	35.675	51.999	72.510	45.556	41.000	88.998
Minimum	7.600	12.618	21.409	7.996	12.998	21.868
Median	9.251	14.586	24.153	9.619	15.051	24.957
Standard Deviation	1.531	2.558	3.598	2.014	1.874	3.598
Total	9.25	14.58	24.15	9.61	15.05	25.12

Table 4

Execution Times in Milliseconds for RKyber1024 and Kyber1024

RKyber1024				Kyber1024		
Time per Stage (in ms)	Keygen	Encryption	Decryption	Keygen	Encryption	Decryption
Maximum	43.998	83.237	113.001	70.714	94.000	125.999
Minimum	12.533	19.023	31.160	11.997	18.204	29.708
Median	14.759	21.884	35.271	14.665	21.739	34.774
Standard Deviation	1.872	3.695	5.064	4.166	5.574	7.521
Total	14.664	21.73	34.77	14.75	21.88	36.27

These results show that RKyber achieves slightly faster execution times than the original Kyber algorithm. As the algorithm's complexity increases with the security level (from Kyber512 to Kyber1024), the differences in execution times become more apparent.

To model these results, parameters *n* (degree of polynomials), *k* (number of polynomials), and ***N*** (possible range of coefficient values) were used (NTT DATA Perú, 2024).

4.2.2 BRUTE-FORCE ATTACK TESTS

This section presents brute-force security tests conducted for each algorithm. These tests involved incrementally increasing the polynomial dimensions from the lowest to the highest possible values to project the time required to break these algorithms using brute force. The results are shown in Table 5 and Figures 3 and 4.

Table 5

Brute-Force Attack Times for RKyber512 and Kyber512

n	Kyber	RKyber
2	0.000529162	0.000525681
3	0.001454904	0.001440572
4	0.004000182	0.003947727
5	0.010998292	0.010818309
6	0.030239226	0.029646378
7	0.083141166	0.081242618
8	0.228592276	0.222636403
9	0.628502473	0.610110424
10	1.728034584	1.671940092
11	4.751140451	4.581766782
12	13.06301146	12.55582478
13	35.91606484	34.40784821

Using the results from the attack executions, the exponential function that fits the attack times can be determined. This allows for the estimation of approximate values for N = 256, which is the standard value used in these algorithms.

Figure 3

Brute-Force Attack on Kyber512

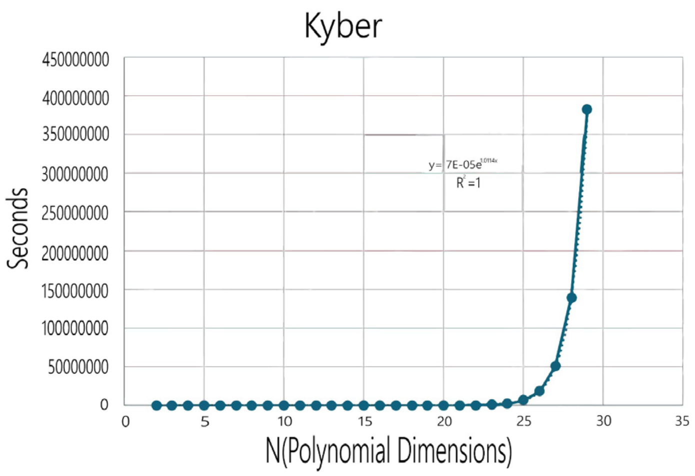
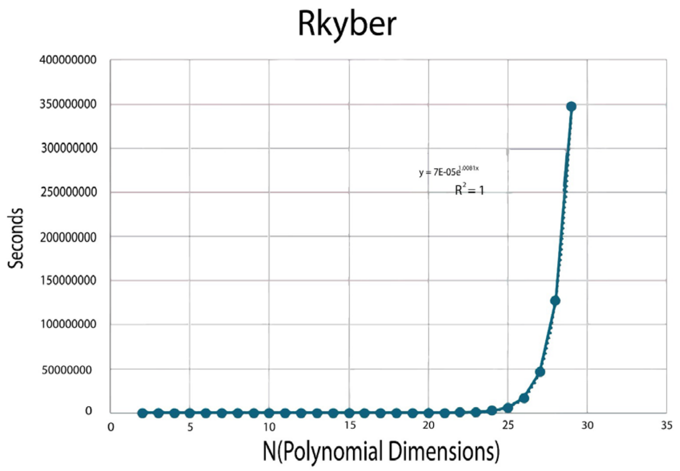


Figure 4

Brute-Force Attack on RKyber512



These results demonstrate that RKyber is slightly easier to break using brute-force attacks, but the difference is negligible. For example, with N = 30, the estimated execution time to break RKyber is approximately 12 years. With N = 256, the attack would take an incalculable amount of time using current technology, meaning that the variant remains secure. In summary, RKyber offers computational advantages in execution time without significantly compromising security, making it a viable alternative to the original algorithm in environments where speed is critical and extreme security is not required.

5. DISCUSSION

As seen in the results from Tables 2, 3, and 4, the execution times for the RKyber variant are shorter than those for the original Kyber algorithm. This improvement occurs because, in RKyber, errors are no longer generated randomly; instead, they are converted into deterministic values, simplifying the algorithm's execution and computation. At first glance, the time difference may not seem significant. However, it is important to consider that these are cumulative results over 1000 executions, where a difference of just 1 ms per execution can add up substantially. On the other hand, the trade-off for this improvement in execution time is a reduction in security. The original Kyber algorithm introduced random noise, including an additional security layer by making it more difficult for attackers to reverse-engineer the encryption. In RKyber, this noise has been removed, making it theoretically more vulnerable. However, it should be emphasized that RKyber remains resistant to quantum attacks, as the underlying lattice is unchanged, meaning an attacker would still need to find the correct polynomial to decrypt the message. So far, no practical attack has fully compromised Kyber's security. The only known attack, KyberSlash (Bernstein, 2024), assumes direct access to the processor where the system is running, making it incredibly difficult to execute. Dr. Daniel Bernstein has tracked this attack and reported that the original implementation has already been patched to mitigate this threat.

In terms of complexity, the biggest difference between the two algorithms lies in the space used in the generations. The LWE-based version of Kyber primarily uses q space, whereas the RKyber variant uses p space (Alwen et al., 2013). However, the overall complexity remains the same, as p is always smaller than q, making this difference negligible in terms of asymptotic bounds.

Originally, this study aimed to simulate an attack using Shor's quantum algorithm, since it efficiently breaks current public-key schemes. However, it became clear that such attacks are not feasible, at least at the time of writing this report. The reason is the way Shor's algorithm inputs and outputs data. In this case, the input should be a number, and the output should be its prime factors. Given that the fundamental bases of lattices are polynomials stored in vectors, factoring them in this way is not possible. A more appropriate solution would involve using Grover's algorithm to optimize coefficient generation, though this was not tested in this research. Using Grover's algorithm along with the reverse-engineering method could be a potential solution.

To estimate the time needed to break the algorithm, a statistical analysis was conducted based on the time (in seconds) recorded during key generation across multiple experiments. A simple statistical model was built using the input parameters n, k, and N (Gonzalez, 2021). This model allowed for the projection of breaking times for different configurations.

6. CONCLUSIONS

This paper introduces a new perspective on the CRYSTALS-Kyber algorithm by modifying the core problem it addresses and extending it to the LWR problem. The resulting variant, named RKyber, significantly improves execution times at the cost of a slight reduction in security. The primary objective of this work was to transition from LWE to LWR and test the modified algorithm. The change inherently compromises security by removing the pseudo-random obfuscation method used in LWE encryption systems, resulting in a trade-off between efficiency and security. However, the reduction in security does not render the algorithm unusable.

Many systems do not require extremely high levels of security, and as discussed in Section 5, the security loss is minimal. The time needed to break RKyber using brute-force attacks remains prohibitive with current technology, ensuring that the variant is still resistant to quantum attacks. Furthermore, RKyber's improvements in execution time could be advantageous in environments where resource efficiency is a priority. The transition from LWE to LWR supports the hypothesis that both pseudo-random and deterministic errors are computationally congruent. Alwen et al. (2013) suggest that the decryption step, where the error is handled, is the most computationally demanding part of the system. By removing the need to deal with LWE's pseudo-random error, the algorithm becomes more efficient without significantly compromising security. This holds true for all versions of Kyber (512, 768, 1024), where the only significant changes are the parameters involved in the key generation, encryption, and decryption processes.

REFERENCES

Aaronson, S., & Chen, L. (2017). Complexity-theoretic foundations of quantum supremacy experiments. *Proceedings of the 32nd Computational Complexity Conference (CCC'17)*, Dagstuhl, Germany, Article 22, 1–67. https://doi.org/10.48550/arXiv.1612.05903

Alwen, J., Krenn, S., Pietrzak, K., & Wichs, D. (2013). Learning with rounding, revisited. In R. Canetti, & J. A. Garay (Eds.), *Lecture Notes in Computer Science: Vol. 8042. Advances in Cryptology – CRYPTO 2013* (pp. 57–74). Springer. https://doi.org/10.1007/978-3-642-40041-4_4

Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., & Stehlé, D. (2021). *CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.01)*. https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf

Benioff, P. (1980). The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, *22*, 563–591. https://doi.org/10.1007/bf01011339

Bernstein, D. J. (2024, June 28). *KyberSlash: division timings depending on secrets in Kyber* software. FAQ. https://kyberslash.cr.yp.to/faq.html

Gonzalez, R. (2021, September 14). *Kyber - How does it work?* Approachable Cryptography. https://cryptopedia.dev/posts/kyber/

Grover, L.K. (1996). *A fast quantum mechanical algorithm for database search.* Bell Labs. https://doi.org/10.1145/237814.237866

Lyubashevsky, V., Peikert, C., & Regev, O. (2010). On ideal lattices and learning with errors over rings. In H. Gilbert (Ed.), *Lecture Notes in Computer Science: Vol. 6110. Advances in Cryptology – EUROCRYPT 2010* (pp. 1–23). Springer. https://doi.org/10.1007/978-3-642-13190-5_1

Micciancio, D., & Goldwasser, S. (2002). *Complexity of lattice problems: A cryptographic perspective*. Springer. https://ci.nii.ac.jp/ncid/BB14507293

Mor, T., & Renner, R. (2014). Preface. *Natural Computing*, *13*(4), 447–452. https://doi.org/10.1007/s11047-014-9464-3

NTT DATA Perú. (2024, April 17). Algoritmos post-cuánticos: criptografía y ciberseguridad en la era cuántica. NTT DATA Perú. https://pe.nttdata.com/documents/paper_crystals_kyber_ntt_data.pdf

Peikert, C. (2016). A decade of lattice cryptography. *Foundations and Trends in Theoretical* Computer Science, 10 (4), 283–424. https://doi.org/10.1561/0400000074

Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC '05),* New York, NY, USA, 84–93. https://doi.org/10.1145/1060590.1060603

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, *26*(5), 1484–1509. https://doi.org/10.1137/s0097539795293172

Wei, Y., Bi, L., Lu, X., & Wang, K. (2023). Security estimation of LWE via BKW algorithms. Cybersecurity, 6, Article 24. https://doi.org/10.1186/s42400-023-00158-9