La Transformada Teórica de Números para Kyber

Néstor Antuñano-Cabrera
Universidad de La Laguna
Tenerife, España
alu0101440460@ull.edu.es

Édgar Pérez-Ramos
Universidad de La Laguna
Tenerife, España
alu0101207667@ull.edu.es

Candelaria Hernández-Goya
Universidad de La Laguna
Tenerife, España
mchgoya@ull.edu.es

Pino Caballero-Gil
Universidad de La Laguna
Tenerife, España
pcaballe@ull.edu.es

***Resumen*—La Transformada Teórica de Números es un método eficiente para la multiplicación de dos polinomios de grado alto, ampliamente usado para sistemas criptográficos basados en retículos como Kyber y Dilithium. Este documento se centra en el caso concreto de Kyber. Incluye una revisión de los conceptos básicos del álgebra de anillos y posteriormente una explicación sobre la convolución de polinomios usando la Transformada Teórica de Números. Además se introducen algunos algoritmos básicos como el radix-2 basado en los algoritmos de Cooley-Tukey y Gentleman-Sande. Finalmente se describe una implementación en Python de la Transformada Teórica de Números generalizada y la correspondiente convolución de polinomios en Kyber.**

***Index Terms*—Criptografía post-cuántica, CRYSTALS-Kyber, NTT**

Tipo de contribución: *Investigación en desarrollo*

I. Introducción

La criptografía de clave pública actual basada principalmente en la dificultad de factorizar primos grandes y la resolución de logaritmos discretos, posee vulnerabilidades frente a ataques cuánticos. Por ello, la criptografía post-cuántica es una alternativa fundamental para el futuro próximo.

En diciembre de 2016, el Instituto Nacional de Estándares y Tecnología (*National Institute of Standards and Technology*, NIST) inició un proceso para estandarizar algoritmos criptográficos resistentes a la computación cuántica, publicando una solicitud de propuestas de algoritmos de criptografía post-cuántica. Esta iniciativa fue un paso importante en el esfuerzo por desarrollar sistemas criptográficos que sean seguros frente a ordenadores cuánticos y clásicos, y que puedan interoperar con protocolos y redes de comunicaciones existentes.

El objetivo de la iniciativa era recibir propuestas de algoritmos que pudieran servir como estándares para firmas digitales y mecanismos de encapsulación de claves (*Key Encapsulation Mechanism*, KEM), que son fundamentales para la seguridad en la era post-cuántica. Para la fecha límite, en noviembre de 2017, el NIST había recibido 69 algoritmos elegibles. Posteriormente en agosto de 2023, NIST dio el siguiente paso para la estandarización de algoritmos criptográficos resistentes a ataques desarrollados con ordenadores cuánticos, publicando borradores de estándares para tres de los cuatro algoritmos seleccionados en 2022 [1]:

- FIPS 203: CRYSTALS-Kyber. Es un KEM basado en el problema de aprendizaje con errores (*Learning With Errors*, LWE) sobre anillos [2].
- FIPS 204: CRYSTALS-Dilithium [3].
- FIPS 205: SPHINCS+ [4].

El NIST invitó a la comunidad criptográfica mundial a proporcionar comentarios sobre estos borradores hasta noviembre de 2023, con el fin de recibir retroalimentación y asegurarse de que los estándares sean completos y no tengan omisiones antes de su finalización.

Por otra parte, la Transformada Teórica de Números (*Number Theoretic Transform*, NTT) es esencial para el desarrollo tanto de CRYSTALS-Kyber, como de CRYSTALS-Dilithium porque permite realizar multiplicaciones de polinomios de manera eficiente, una operación clave en criptosistemas basados en retículos como Kyber. La NTT es una variante de la Transformada Rápida de Fourier (*Fast Fourier Transform*, FFT) adaptada para trabajar en un anillo de números enteros módulo un número primo [5], [6].

La eficiencia de CRYSTALS-Kyber depende en gran medida de la rapidez con la que se pueden realizar estas multiplicaciones polinómicas, y la NTT permite hacerlo de manera rápida y eficiente. Por tanto, la NTT es una herramienta crucial para optimizar el rendimiento de Kyber, especialmente en lo que respecta a la generación de claves, el cifrado y el descifrado, que son procesos computacionalmente intensivos.

En resumen, sin la NTT, los algoritmos como CRYSTALS-Kyber no serían prácticos para su uso en aplicaciones reales debido a las demandas del cálculo computacional que implican. La NTT permite que Kyber sea un esquema de cifrado viable y seguro para la criptografía en la era post-cuántica.

Este trabajo se estructura como sigue. La sección II incluye algunos preliminares esenciales de la Teoría de Anillos, la Transformada Teórica de Números y el Teorema Chino del Resto. La sección III se dedica a los algoritmos Butterfly, mientras que la sección IV trata sobre la convolución en Kyber. En la sección V se aportan datos sobre la implementación de la NTT. Finalmente la sección VI cierra el trabajo con algunas conclusiones y trabajos futuros.

II. Preliminares

A continuación se define la notación utilizada y se incluye una breve introducción a la multiplicación polinomial y a la convolución.

II-A. Teoría de Anillos

Sea $\mathbb{Z}$ el anillo de los enteros, y sean $n \in \mathbb{Z}$ positivo y q un número primo, se considera $\mathbb{Z}_q = \{[0]_q, [1]_q, ..., [q-1]_q\}$ el anillo de los enteros módulo q.

Se denota a lo largo de todo el documento $\mathbb{Z}_q[x]/I$ al anillo cociente sobre el anillo de polinomios $\mathbb{Z}_q[x]$, donde q es un número primo e $I = (x^n + 1)$. Véase en [7] que cualquier $a(x) \in \mathbb{Z}_q[x]/I$, tiene grado menor o igual a $n-1$. Debe tenerse en cuenta que al hacer modulo $x^n + 1$, resulta que $x^n = -1$ por lo que los polinomios del anillo no tendrán grado igual a n, sino menor.

Definición 2.1: (k-ésima raíz primitiva de la unidad [7]). Sea R un anillo conmutativo y unitario, es decir un anillo conmutativo con identidad multiplicativa 1, entonces ψ es una k-ésima raíz de la unidad en R si y solo si

$$\psi^k = 1, \quad \psi^i \neq 1, \forall i \in \{0, 1, .., k-1\}. \tag{1}$$

En Kyber se utilizan los parámetros $q = 3329$, $n = 256$ siendo el anillo $\mathbb{Z}_q[x]/I = \mathbb{Z}_{3329}[x]/(x^{256}+1)$.

El anillo $\mathbb{Z}_{3329}$ en concreto, tiene una particularidad que lo hace ligeramente distinto a los demás y es que para el uso de algoritmos en la multiplicación de polinomios, como se ve más adelante, es necesaria la existencia de una $2n$-ésima raíz de la unidad. En CRYSTALS-Kyber, $2n = 512$, y se comprueba en la implementación desarrollada que no existe dicha $2n$-ésima raíz primitiva de la unidad.

Definición 2.2: (Negative Wrapped Convolution [8]). Sean $q \in \mathbb{N}$ un número primo e $I = (x^n + 1)$ el ideal generado por $x^n+1 \in \mathbb{Z}_q[x]$, y $a(x), b(x) \in \mathbb{Z}_q[x]/I$, supóngase que ambos polinomios $a(x), b(x)$ tienen un grado exacto de $n-1$, si fueran de grado menor se rellenan los coeficientes con ceros.

Siendo $a = (a_0, ..., a_{n-1})$, $b = (b_0, ..., b_{n-1}) \in \mathbb{Z}_q^n$ los vectores de coeficientes de $a(x), b(x)$ respectivamente.

Se define la *Negative Wrapped Convolution* (NWC) de $a(x)$ y $b(x)$ como la multiplicación de estos polinomios

$$c(x) = \sum_{k=0}^{n-1} c_k x^k \tag{2}$$

siendo:

$$c_k = \sum_{i=0}^{k} a_i b_{k-i} + \sum_{i=k+1}^{n-1} a_i b_{k+n-i} \pmod q, \tag{3}$$

$\forall k \in \{0, 1, .., n-1\}$

Estos coeficientes c_k no son más que los coeficientes de la multiplicación de polinomios en $\mathbb{Z}_q[x]$ reducido módulo $I = (x^n + 1)$.

II-B. Transformada Teórica de Números

La NTT es un caso especial de la transformada discreta de Fourier sobre un cuerpo finito. Hay varios tipo de NTT, pues también se pueden dar convoluciones concretas para $\mathbb{Z}_q[x]$, y $\mathbb{Z}_q[x]/(x^n - 1)$. En el caso de $\mathbb{Z}_q[x]/(x^n + 1)$ se tiene la convolución definida anteriormente, y la NTT en Kyber está basada en ella.

Definición 2.3: (Transformada Teórica de Números basada en la NWC [8]). Sea $q \in \mathbb{N}$ un número primo tal que $q \equiv 1 \pmod{2n}$ tal que la $2n$-ésima raíz de la unidad ψ_{2n} existe en $\mathbb{Z}_q$. Se denota $w_n = \psi_{2n}^2$ (la n-ésima raíz de la unidad en $\mathbb{Z}_q$).

Se define entonces la Transformada Teórica de Números basada en la NWC de $a = (a_0, ..., a_{n-1}) \in \mathbb{Z}_q^n$ siendo a el vector de coeficientes de $a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1} \in \mathbb{Z}_q[x]/(x^n + 1)$ como:

$$NTT_\psi(a) = (\hat{a}_0, \hat{a}_1, ..., \hat{a}_{n-1}) \tag{4}$$

Donde:

$$\hat{a}_j = \sum_{i=0}^{n-1} a_i \psi_{2n}^i w_n^{ij} \pmod q \tag{5}$$

Realmente la NTT_ψ se puede definir en $\mathbb{Z}_q^n$ como la aplicación:

$$\begin{array}{rccc} NTT_\psi: & \mathbb{Z}_q^n & \longrightarrow & \mathbb{Z}_q^n \\ & (a_0, a_1, ..., a_{n-1}) & \longmapsto & (\hat{a}_0, \hat{a}_1, ..., \hat{a}_{n-1}) \end{array} \tag{6}$$

con $\hat{a}_j$ definidos como *Ec.* (5).

Véase que al ser $w_n = \psi_{2n}^2$, se tiene que la expresión *Ec.* (5) es equivalente a:

$$\hat{a}_j = \psi_{2n}^{2j+1} \sum_{i=0}^{n-1} a_i \psi_{2n}^i \pmod q$$

Además en este documento se denota de igual manera $NTT(a) := NTT_\psi(a)$ pues no se está considerando otro tipo de convoluciones.

Definición 2.4: (Transformada Teórica de Números Inversa basada en la NWC [8]). Partiendo de las mismas hipótesis de la definición de NTT_ψ se define la Transformada Teórica de Números Inversa (*Inverse Number Theoretic Transform*, INTT) basada en la NWC de $\hat{a} = (\hat{a}_0, \hat{a}_1, .., \hat{a}_{n-1})$ siendo $\hat{a}$ el vector de coeficientes de $\hat{a}(x) = \hat{a}_0 + \hat{a}_1 x + \hat{a}_2 x^2 + \cdots + \hat{a}_{n-1}x^{n-1} \in \mathbb{Z}_q[x]/(x^n + 1)$ como:

$$INTT_\psi(\hat{a}) = (a_0, a_1, ..., a_{n-1}) \tag{7}$$

donde:

$$a_i = n^{-1} \psi_{2n}^{-i} \sum_{j=0}^{n-1} \hat{a}_j w_n^{-ij} \pmod q \tag{8}$$

Proposición 2.1: Sea $a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1} \in \mathbb{Z}_q[x]/(x^n + 1)$ y $a = (a_0, ..., a_{n-1}) \in \mathbb{Z}_q^n$ su vector de coeficientes adecuado, entonces:

$$INNT_\psi(NTT_\psi(a)) = a \tag{9}$$

Esta proposición ya indica que la NTT_ψ es una correspondencia biyectiva.

Proposición 2.2: (Propiedad de la NWC [8]). Sea $c(x) \in \mathbb{Z}_q[x]/(x^n+1)$ la convolución de dos polinomios $a(x), b(x) \in \mathbb{Z}_q[x]/(x^n + 1)$, denotando por c, a, b los vectores de coeficientes de los respectivos polinomios, y $\circ$ al producto punto a punto de dos vectores, entonces:

$$NTT_\psi(c) = NTT_\psi(a) \circ NTT_\psi(b) \tag{10}$$

Como consecuencia directa de las proposiciones anteriores, se tiene que:

$$c = INNT_\psi(NTT_\psi(a) \circ NTT_\psi(b)) \tag{11}$$

Por tanto, se puede obtener la multiplicación de dos polinomios en $\mathbb{Z}_q[x]/(x^n + 1)$ usando NTT_ψ y $INTT_\psi$ siempre que se asuman las hipótesis.

En Kyber, como se ha mencionado anteriormente, no existe dicha 256-raíz de la unidad, y por tanto, lo anterior por ahora no es aplicable.

En la definición y teorema siguientes se introduce una justificación para el procedimiento que se emplea en Kyber.

II-C. Teorema Chino del Resto

Definición 2.5: (Ideales coprimos [9]). Sean $I, J \subseteq R$ ideales, con R anillo conmutativo y unitario, entonces I, J son ideales coprimos si y solo si $\exists i \in I,\ j \in J$ tal que $i + j = 1$.

Teorema 2.1: (Teorema Chino del Resto Generalizado [9]). Sea R un anillo conmutativo y unitario, e $I_1, I_2, .., I_k \subseteq R$ ideales coprimos entre sí, considerando I como la intersección de todos los I_j, los siguientes anillos son isomorfos:

$$R/I \cong R/I_1 \times R/I_2 \times \cdots \times R/I_k \tag{12}$$

Considérese en $R = \mathbb{Z}_q[x]$ el ideal $H = (x^n + 1)$ con $n \in \mathbb{N}$ una potencia de 2 y ψ_{2n} la $2n$-ésima raíz de la unidad, se cumple por definición que $\psi_{2n}^{2n} = 1$ y al ser q primo, entonces $\mathbb{Z}_q$ es un cuerpo, luego se puede asegurar que $\psi_{2n}^{n} = -1$ por tanto se tiene que el polinomio generador x^n+1 cumple que $x^n+1 = x^n-\psi_{2n}^n = [x^{\frac{n}{2}}-\psi_{2n}^{\frac{n}{2}}][x^n+\psi_{2n}^{\frac{n}{2}}]$.

Así, si se define $I_1 = (x^{\frac{n}{2}} - \psi_{2n}^{\frac{n}{2}})$ y $I_2 = (x^n + \psi_{2n}^{\frac{n}{2}})$ se tiene que $I = I_1 \cap I_2 = H$.

Además se observa que si:

- $a(x) = (-2^{-1}\psi_{2n}^{-\frac{n}{2}})(x^n - \psi_{2n}^{\frac{n}{2}}) \in I_1$
- $b(x) = (2^{-1}\psi_{2n}^{-\frac{n}{2}})(x^n + \psi_{2n}^{\frac{n}{2}}) \in I_2$

se da que:

$$a(x) + b(x) = -2^{-1}\psi_{2n}^{-\frac{n}{2}}x^n + 2^{-1}\psi_{2n}^{\frac{n}{2}}x^n + 2^{-1} + 2^{-1} = 2^{-1} + 2^{-1} = 2 \cdot 2^{-1} = 1$$

Luego, se reúnen las hipótesis del teorema y se concluye que:

$$\mathbb{Z}_q[x]/(x^n + 1) \cong \mathbb{Z}_q[x]/(x^{\frac{n}{2}} - \psi_{2n}^{\frac{n}{2}}) \times \mathbb{Z}_q[x]/(x^{\frac{n}{2}} + \psi_{2n}^{\frac{n}{2}})$$

Es posible aplicar esto recursivamente (ver Fig. 1) pero dado que las raíces de x^n+1 son ψ_{2n}^{2j+1}, con $j \in \{0, 1, ...n-1\}$, de manera análoga teniendo la descomposición de:

$$x^n + 1 = \prod_{j=0}^{n-1}(x^n - \psi_{2n}^{2j+1}) \tag{13}$$

y aplicando el Teorema Chino del Resto:

$$\mathbb{Z}_q[x]/(x^n + 1) \cong \prod_{j=0}^{n-1}\mathbb{Z}_q[x]/(x - \psi_{2n}^{2j+1}) \tag{14}$$

Definición 2.6: (Bit-reverso [8]). Sea $n \in \mathbb{N}$ una potencia de 2 y $b \in \mathbb{Z}$, con $b \geq 0$, se define el bit-reverso de b respecto de n como:

$$br_n(b) = br_n(b_{log_2(n-1)}2^{log_2(n-1)} + \cdots + b_1 2 + b_0) = b_0 2^{log_2(n-1)} + \cdots + b_{log_2(n-2)}2 + b_{log_2(n-1)} \tag{15}$$

Donde b_i representa el i-ésimo bit en la forma binaria de b. Como se ve es simplemente revertir el orden de los bits y devolver la representación entera.
Ahora como el bit reverso (br) de los enteros $\{0, ..., n-1\}$ forman una permutación de este conjunto, la expresión que se tenía es equivalente a:

$$\mathbb{Z}_q[x]/(x^n + 1) \cong \prod_{j=0}^{n-1}\mathbb{Z}_q[x]/(x - \psi_{2n}^{2br(j)+1}) \tag{16}$$

La razón detrás de esta reversión de bits en los índices de exponenciación es que se da un mejor acceso en memoria a la hora de implementar.

Ahora con *Ec. (17)* se sabe que si existe una $2n$-ésima raíz de la unidad, entonces es posible recurrir a una multiplicación punto a punto en la NTT, pues al ser reducido con un polinomio de grado 1 solo quedan constantes.

En Kyber no se puede recurrir a una multiplicación punto a punto de términos constantes, ya que solo existe hasta la n-ésima raíz de la unidad. Luego, se puede aplicar el Teorema Chino del Resto hasta polinomios de grado 2, es decir:

Considerando w_n la n-ésima raíz de la unidad con los parámetros de Kyber, se cumple que:

$$\mathbb{Z}_q[x]/(x^n + 1) \cong \prod_{j=0}^{\frac{n}{2}-1}\mathbb{Z}_q[x]/(x^2 - w_n^{2br(j)+1}) \tag{17}$$

II-D. Complejidad de la NTT

La complejidad de aplicar NTT/INTT de manera directa es de $O(n^2)$, la cuál es la misma complejidad que se da en la multiplicación de polinomios directa con anillos de la forma $K[x]$ con K cuerpo. Por tanto, la eficiencia de esta transformada se encuentra en los algoritmos que rebajan el orden. Se observa en las siguientes subsecciones, y llegan a conseguir un orden de $O(n\log(n))$.

II-E. Ventajas de la NTT

La NTT presenta ciertas propiedades que favorecen a los esquemas de cifrado:

- La NTT es una correspondencia lineal y biyectiva, lo cuál resultará útil para la implementación de Kyber.
- Debido a que la NTT es una correspondencia biyectiva, conserva aleatoriedad de un vector de coeficientes. Con esto se puede generar un vector aleatorio y verlo como un vector ya transformado de la NTT. Esto es posible ya que es sobreyectiva por lo que tendrá un vector que lo tenga como imagen, y además no hay problemas con la aleatoriedad pues la conserva ya que, al ser biyectiva, si la imagen no fuera aleatoria se podría conocer el elemento de entrada, por lo que no sería aleatorio.
- En cuanto al acceso de memoria en una implementación, como la transformada lleva n puntos en n puntos, es posible guardar $\hat{a}$ donde originalmente se encontraba a en memoria.
- Como se ve en el algoritmo de Cooley-Tukey (ver Fig. 2), se pueden calcular con cierta recursividad los coeficientes $\hat{a}_j$, haciendo que se ahorre en términos de coste computacional en la transformación.

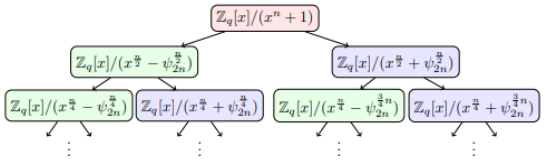

Figura 1. Teorema Chino del Resto en $\mathbb{Z}_q[x]/(x^n + 1)$

III. Algoritmos-*Butterfly*: Cooley-Tukey y Gentleman-Sande

En esta sección se desarrollan los algoritmos que mejoran en gran medida el rendimiento de la NTT, explicando el procedimiento por el cuál se llegan a tener los esquemas *Butterfly* [10].

III-A. Base teórica de los algoritmos Butterfly.

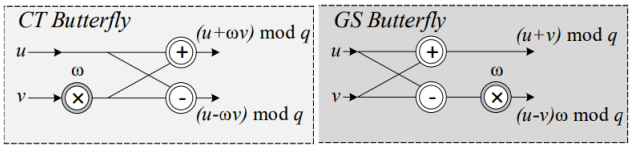

Figura 2. Algoritmos Butterfly

De la definición de NTT_ψ (2.3), los coeficientes transformados Ec. (5) :

$$\hat{a}_j = \sum_{i=0}^{n-1} a_i \psi_{2n}^i w_n^{ij} \pmod q$$

Se divide el sumatorio en los índices pares e impares por i, recordar que n es potencia de 2 luego divisible por 2, queda:

$$\hat{a}_j = \sum_{i=0}^{\frac{n}{2}-1} a_{2i} \psi_{2n}^{2i} w_n^{2ij} + \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} \psi_{2n}^{2i+1} w_n^{(2i+1)j} \pmod q$$

De aquí en adelante las operaciones asumen la aplicación de módulo q.

Usando simplemente la propiedad de potencias, y sacando el factor ψ_{2n} y w_n^j, pues j no depende de la variable i en el sumando, se tiene que $\hat{a}_j$ es:

$$\sum_{i=0}^{\frac{n}{2}-1} a_{2i} (\psi_{2n}^i)^2 (w^{ij})_n^2 + \psi_{2n} w_n^j \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} (\psi_{2n}^i)^2 (w_n^{ij})^2$$

Ahora, la idea es sustituir $j + \frac{n}{2}$ en lo anterior y considerar las propiedades de estas raíces de la unidad que se vieron en las secciones II-B, II-C, estas son: $\psi_{2n}^n = -1$ y $w_n = \psi_{2n}^2$. Resulta que $\hat{a}_{j+\frac{n}{2}}$ es:

$$\sum_{i=0}^{\frac{n}{2}-1} a_{2i} (\psi_{2n}^i)^2 (w^{ij})_n^2 - \psi_{2n} w_n^j \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} (\psi_{2n}^i)^2 (w_n^{ij})^2$$

Se define:

$$\hat{a}'_j := \sum_{i=0}^{\frac{n}{2}-1} a_{2i} (\psi_{2n}^i)^2 (w^{ij})_n^2 \pmod q$$

$$\hat{a}''_j := \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} (\psi_{2n}^i)^2 \pmod q$$

Luego, teniendo en cuenta que $\psi_{2n} w_n^j = \psi_{2n}^{2j+1}$ y sustituyendo por $\hat{a}'_j$, $\hat{a}''_j$:

$$\hat{a}_j = \hat{a}'_j + (\psi_{2n}^{2j+1}) \hat{a}''_j \pmod q$$

$$\hat{a}_{j+\frac{n}{2}} = \hat{a}'_j - (\psi_{2n}^{2j+1}) \hat{a}''_j \pmod q$$

Donde además estos $\hat{a}'_j$, $\hat{a}''_j$ corresponden al cálculo de una NTT sobre $\frac{n}{2}$ puntos (de la propia definición) y así recursivamente se obtienen los coeficientes.

Véase que lo que se está haciendo es separar en cada etapa k en la que se aplica esta recursividad con una distancia de $n/(k+1)$ en los índices de $\hat{a}_j$ tal y como está descrito en Fig. 3 [11]. Por ejemplo en la primera etapa, se está a distancia $\frac{n}{2}$: $\hat{a}_j$ y $\hat{a}_{j+\frac{n}{2}}$.

Esta es una de las bases del algoritmo Butterfly, el cuál se denota así por que las operaciones en cada subetapa al representarlo esquemáticamente se 'asemejan' a la figura de una mariposa, como se muestra en Fig. 2.

Para el caso de la INTT se procede manera análoga, de la expresión de la $INTT_\psi$:

$$a_i = n^{-1} \psi_{2n}^{-i} \sum_{j=0}^{n-1} \hat{a}_j w_n^{-ij} \pmod q$$

Desarrollando y definiendo los siguientes elementos:

$$\hat{b}'_j := \hat{a}_j + \hat{a}_{j+\frac{n}{2}} \pmod q$$

$$\hat{b}''_j := (\hat{a}_j + \hat{a}_{j+\frac{n}{2}}) \psi^{-(2j+1)} \pmod q$$

Nos queda en la sustitución la expresión:

$$a_{2i} = (\psi_{2n}^2)^{-i} \sum_{j=0}^{\frac{n}{2}-1} \hat{b}'_j (w^2)_n^{-ij} \pmod q$$

$$a_{2i+1} = (\psi_{2n}^2)^{-i} \sum_{j=0}^{\frac{n}{2}-1} \hat{b}''_j (w^2)_n^{-ij} \pmod q$$

Por tanto, es posible aplicar ahora un $INTT_\psi$ basada en $\frac{n}{2}$ puntos. De nuevo es posible hacer una recurrencia análoga a la que hecha con la NTT_ψ.

III-B. Cooley-Tukey

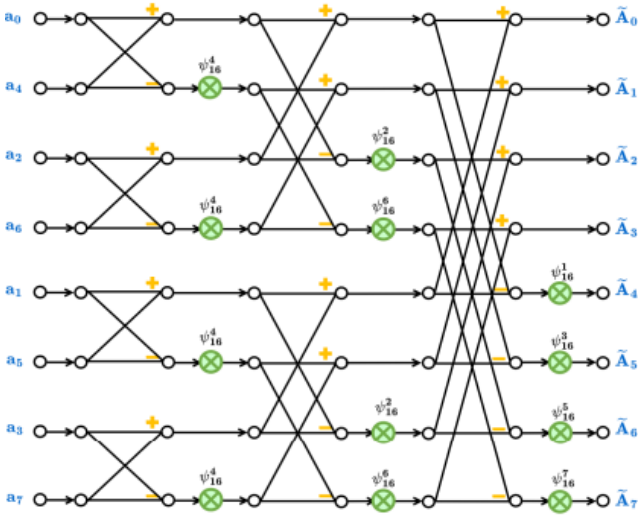

Figura 3. Butterfly-Cooley-Tukey para un $n = 8$

Una vez introducidas las bases, se describe el algoritmo de Cooley-Tukey [12] en el pseudocódigo incluido como

Algoritmo 1. La notación usada es la siguiente: m indica la etapa actual de la NTT_ψ, mientras que k se refiere a la distancia a la que están los índices en dicha etapa. A su vez, el cálculo de índices allí descrito se usa para indexar cada uno de los elementos, sumar la distancia k y realizar la operación mariposa.

Véase que la salida de este algoritmo está automáticamente generada en bit-reverse, tal y cómo se refleja en Fig. 3. La lista de entrada, cuyos elementos corresponden a los coeficientes de un polinomio en $\mathbb{Z}_q[x]/(x^8+1)$, tienen la correspondencia de índices en bit reverse (sobre 3-bits pues $3 = \log_2(8)$) : $[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7] \rightarrow [a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7]$ por lo que está en reversión.

Esto no es un problema ni requiere modificaciones, ya que el conjunto de datos de entrada $INTT_\psi$ está organizado de acuerdo con el método de bit-reversión y luego se aplica el proceso de bit-reversión nuevamente en la salida de manera predeterminada. Por lo tanto, al realizar convoluciones de polinomios que requieran la aplicación simultánea de NTT_ψ e $INTT_\psi$, se obtiene el resultado correcto.

Algoritmo 1 NTT_ψ basada en el algoritmo Cooley-Tukey Butterfly

Entrada: Vector $a = (a_0, a_1, .., a_{n-1})$ de $\mathbb{Z}_q$.
Salida: $a \leftarrow NTT_\psi(a)$
$m \leftarrow 1$
$k \leftarrow \frac{n}{2}$
$\psi \leftarrow RaizPrimitiva(2n, q)$ ▷ Previamente creada
while $m < n$ **do**
  for $i = 0$ to $i = m - 1$ **do**
    $j_1 \leftarrow 2 \cdot i \cdot k$
    $j_2 \leftarrow j_1 + k - 1$
    $S \leftarrow \psi^{bitrev(m+i,n)}$
    for $j = j_1$ to $j = j_2$ **do**
      $u \leftarrow a[j]$
      $v \leftarrow a[j + t]$
      $a[j] \leftarrow u + v \cdot S \pmod q$
      $a[j + t] \leftarrow u - v \cdot S \pmod q$
    end for
  end for
  $m \leftarrow m \cdot 2$
  $k \leftarrow k/2$
end while
return a

III-C. Gentleman-Sande

Siguiendo el pseudocódigo del Algoritmo 2, y utilizando un razonamiento similar al explicado anteriormente, se observa que al aplicar la recursividad inversa, los índices m y k, así como la exponenciación de ψ^{-1}, se comportan de manera opuesta a la utilizada en el algoritmo de Cooley-Tukey. Se concluye con la correspondiente operación mariposa descrita en la sección III-A.

Es importante notar que dado que $\psi^{2n} = 1$, entonces $\psi^{2n-1} \cdot \psi = 1$. Asi, se puede deducir que $\psi^{-1} = \psi^{2n-1}$. Por lo tanto, se calcula la inversa de ψ elevando a la potencia $2n - 1$, como se indica en el Algoritmo 2.

Algoritmo 2 $INTT_\psi$ basada en el algoritmo Gentleman-Sande Butterfly

Entrada: Vector $a = (a_0, a_1, .., a_{n-1})$ de $\mathbb{Z}_q$.
Salida: $a \leftarrow INTT_\psi(a)$
$m \leftarrow \frac{n}{2}$
$k \leftarrow 1$
$\psi \leftarrow RaizPrimitiva(2n, q)$ ▷ Previamente creada
$\psi^{-1} \leftarrow \psi^{2n-1}$
$n^{-1} \leftarrow n^{q-1}$
while $m \geq n$ **do**
  for $i = 0$ to $i = m - 1$ **do**
    $j_1 \leftarrow 2 \cdot i \cdot k$
    $j_2 \leftarrow j_1 + k - 1$
    $S \leftarrow \psi^{bitrev(m+i,n)}$
    for $j = j_1$ to $j = j_2$ **do**
      $u \leftarrow a[j]$
      $v \leftarrow a[j + t]$
      $a[j] \leftarrow (u + v) \pmod q$
      $a[j + t] \leftarrow (u - v) \cdot S \pmod q$
    end for
  end for
  $m \leftarrow m/2$
  $k \leftarrow k \cdot 2$
end while
$a \leftarrow a \cdot n^{-1}$ ▷ Se multiplica cada componente por n^{-1}
return a

III-D. Convolución de polinomios, con una $2n$-ésima raíz de la unidad en $\mathbb{Z}_q$

Una vez se cuenta con estos algoritmos, al estar diseñados en base a la existencia de una $2n$-ésima raíz de la unidad en $\mathbb{Z}_q$, la convolución en este caso consiste en aplicar la ecuación siguiente:

$$c = INNT_\psi(NTT_\psi(a) \circ NTT_\psi(b)) \tag{18}$$

Para ello es conveniente dar un criterio que asegurare la existencia de dicha raíz primitiva de la unidad.

Proposición 3.1: (Criterio de existencia de una $2n$-ésima raíz de la unidad en $\mathbb{Z}_q$ [13]). Sea q un número primo tal que $2n$ divide a $q - 1$, entonces existe una $2n$-ésima raíz de la unidad en $\mathbb{Z}_q$.
Por lo que, en caso de cumplir este criterio, bastaría con implementar los algoritmos correspondientes. Se incluye un caso para ilustrar la implementación realizada en Python con los siguientes datos: número primo $q = 257 = 2^8 + 1$ y $n = 16$. Dado que $257 - 1 = 2^8$ es divisible por $2n = 32$, se tiene la existencia de la raíz de la unidad.

Realmente es necesario otro requerimiento para aplicar la NTT_ψ, y es que se necesita que en todo momento se pueda dividir n por 2 hasta llegar 1, luego el valor de n debe ser una potencia de 2.

IV. Convolución en Kyber

En Kyber se cuenta con los parámetros $q = 3329$ y $n = 256$, donde no existe una 512-ésima raíz de la unidad en $\mathbb{Z}_{3329}$. Por tanto, no se pueden aplicar directamente los

algoritmos descritos, se requiere una pequeña modificación inicial.

Tal y como se describió en la sección II-C *Teorema Chino del Resto* se necesita una multiplicación punto a punto en $\mathbb{Z}_q[x]/(x^2 - w_n^{2br(i)+1})$ por lo que se tiene una multiplicación de polinomios de grado 1, y posteriormente una reducción modular con un polinomio de grado 2.

La estrategia a seguir es separar un polinomio dado en el anillo de Kyber con $n = 256$, en dos polinomios nuevos, uno con sus coeficientes pares y otro con sus coeficientes impares. Ahora, cada polinomio tiene grado $n = 128$, por lo que si existe una 256-raíz de la unidad en $\mathbb{Z}_{3329}$ para cada uno. Esto permite aplicar la NTT a cada uno de estos polinomios. El resultado obtenido en estas dos transformaciones tiene un sentido algebraico derivado de lo estudiado para Kyber en la sección II-C. Esto es, se parte de:

$$a(x) = a_0 + \cdots + a_{255}x^{255}$$
$$b(x) = b_0 + \cdots + b_{255}x^{255}$$

- Se separan los polinomios en su parte par e impar:

$$a_{par}(x) = a_0 + a_2x \cdots + a_{254}x^{128}$$
$$a_{impar}(x) = a_1 + a_3x + \cdots + a_{255}x^{128}$$
$$b_{par}(x) = b_0 + b_2x \cdots + b_{254}x^{128}$$
$$b_{impar}(x) = b_1 + b_3x + \cdots + b_{255}x^{128}$$

- Se aplica NTT a los coeficientes asociados, denotando la NTT_ψ y $INTT_\psi$ de un polinomio como la transformada de sus coeficientes asociados:

$$NTT_\psi(a_{par}(x)) = (\hat{a}_0, \hat{a}_2, .., \hat{a}_{254})$$
$$NTT_\psi(a_{impar}(x)) = (\hat{a}_1, \hat{a}_3, ..., \hat{a}_{255})$$
$$NTT_\psi(b_{par}(x)) = (\hat{b}_0, \hat{b}_2, ..., \hat{b}_{254})$$
$$NTT_\psi(b_{impar}(x)) = (\hat{b}_1, \hat{b}_3, ..., \hat{b}_{255})$$

- Se usa la expresión algebraica que tiene estas transformada en conjunto sobre $\mathbb{Z}_q[x]/(x^2 - w_n^{2br(i)+1})$:

$$NTT_{Kyber}(a(x)) = (\hat{a}_0 + \hat{a}_1x, \hat{a}_2 + \hat{a}_3x, .., \hat{a}_{254} + \hat{a}_{255}x)$$
$$NTT_{Kyber}(b(x)) = (\hat{b}_0 + \hat{b}_1x, \hat{b}_2 + \hat{b}_3x, .., \hat{b}_{254} + \hat{b}_{255}x)$$

- Se aplica en este dominio la multiplicación punto a punto en $\mathbb{Z}_q[x]/(x^2 - w_n^{2br(i)+1})$, es decir:

$$NTT_{Kyber}(a(x)) \circ NTT_{Kyber}(b(x)) =$$
$$(\hat{c}_0 + \hat{c}_1x, \hat{c}_2 + \hat{c}_3x, .., \hat{c}_{254} + \hat{c}_{255}x)$$

  Donde estos c_k se pueden calcular sin tener que multiplicar y posteriormente hacer reducción de módulo a cada uno, pues ya se sabe la forma que tiene ese cociente cuando es una multiplicación de polinomios de grado 1. Esta es:

$$c_{2i} = a_{2i} \cdot b_{2i} + a_{2i+1} \cdot b_{2i+1}w^{2br(i)+1}$$
$$c_{2i+1} = a_{2i} \cdot b_{2i+1} + a_{2i+1} \cdot b_{2i}$$

  con $i \in \{0, 1, 2..., 255\}$.
- Por último, se consideran estos coeficientes c_k obtenidos de la multiplicación punto a punto y se aplica la $INTT_\psi$ a los coeficientes pares e impares, tal y como se precedió con NTT_ψ. Juntando en un solo polinomio los coeficientes obtenidos que corresponden a la parte par e impar, se obtiene la convolución de polinomios. Esto es, denotando $INTT_{Kyber}$ a la forma descrita de aplicar $INTT_\psi$ se obtiene:

$$a(x) \cdot b(x) = INTT_{Kyber}(c(x))$$

V. Implementación de la NTT_ψ, $INTT_\psi$ y la convolución en Kyber

En esta sección se incluye la implementación realizada en Google Colab desarrollada en Python, de la NTT_ψ, $INTT_\psi$, usando Cooley-Tukey, Gentleman-Sande respectivamente tal y como se describió en los Algoritmos 1 y 2, así como la convolución en Kyber, se puede consultar en [15].

Además como medida de eficiencia se muestran los tiempos de ejecución de dicha implementación y una ya creada en el módulo $SymPy$, una biblioteca de Python para matemáticas simbólicas. También se incluye una comparación de tiempos de ejecución de la multiplicación de polinomios de Kyber usando una multiplicación directa desde el mismo módulo de $SymPy$, que ya cuenta con algoritmos eficientes para multiplicar polinomios de alto grado, y un tipo de convolución para un anillo de polinomios.

Cabe resaltar que la implementación de NTT e INTT de $SymPy$ hace uso de un algoritmo similar pero para un caso distinto de convolución que no es aplicable para $\mathbb{Z}_q[x]/(x^n + 1)$ pero es lo más cercano que dispone Python con librerías reconocidas.

Se ha escogido la plataforma de Google Colab y Python con el fin de obtener una verificación de estos algoritmos y facilitar el aprendizaje sobre la NTT. Python es un lenguaje de alto nivel sencillo de manejar, y Google Colab proporciona un entorno apto para introducir de manera didáctica la implementación realizada.

Para ello se crean las funciones necesarias para la implementación, y posteriormente las funciones NTT, INTT y KyberConvolution. Además, se han implementado funciones más básicas como por ejemplo, el *bit-reverse*. Seguidamente se describen las principales funciones.

V-A. *Función de la n-ésima raíz de la unidad*

Para implementar una función en Python que calcule la n-ésima raíz de la unidad en $\mathbb{Z}_q$ se ha planteado una búsqueda iterativa que devuelve el elemento si cumple las dos condiciones necesarias para que sea n-ésima raíz de la unidad. Véase que no solo se calcula una n-ésima raíz de la unidad si no que es la primera de ellas.

En Fig. 4 y Fig. 5 se muestran los resultados de aplicar esta función en el cuerpo de Kyber y en $\mathbb{Z}_{17}$, ilustrando así que es una función general. Además, en Fig. 6 se muestra con esta misma función que no existe una 512-raíz de la unidad en $\mathbb{Z}_{3329}$.

```
# Ejemplo de la función primera_raiz_primitiva para el
#número q primo de kyber 3329=13*2^8+1
PrimeraRaizPrimitiva(256,3329)

17
```

Figura 4. Primera-Raiz-Primitiva para $Kyber$ y $n = 256$

```
# Ejemplo de la función primera_raiz_primitiva para el
#número q primo de 17 = (2**4)+1
PrimeraRaizPrimitiva(2,17)

16
```

Figura 5. Primera-Raiz-Primitiva para $q = 17$ y $n = 2$

```
# Ejemplo de la función primera_raiz_primitiva para el
#número q primo de 17 = (2**4)+1
PrimeraRaizPrimitiva(512,3329)

Error: Se busca un n tal que n divida a q-1.
```

Figura 6. Primera-Raiz-Primitiva para Kyber y $n = 512$

```
#Ejemplo de NTT para un
# q = 257 = 2**8+1 el cúal es primo.
NTT([6,3,4,6,2,16,7,8,6,3,4,6,2,16,7,8],257)

[172, 172, 66, 225, 157, 182, 54, 45, 54, 30, 210, 87, 73, 45, 64, 2]
```

Figura 7. NTT aplicada a $q = 257$ y $n = 16$

V-B. *Funciones NTT y INTT*

```
# Veamos que nuestra INTT es precisamente la inversa.
INTT(NTT([6,3,4,6,2,16,7,8],17),17)

[6, 3, 4, 6, 2, 16, 7, 8]
```

Figura 8. INTT aplicada a un elemento del dominio de NTT

```
# Ejemplo de multiplicación en Kyber usando convolution.

# Con los siguientes polinomios haremos el proceso de multiplicacion vía ntt.
p=[x for x in range(256)]
g=[(6*y+3) for y in range(256)]
p_mult_g= KyberConvolution(p,g)
print(p_mult_g)

[3150, 1934, 2272, 847, 1000, 2743, 2759, 1060, 987, 2552, 2438, 657, 550, 2129,
```

Figura 9. Kyber Convolution aplicada a polinomios de $\mathbb{Z}_{3329}[x]/(x^{256}+1)$

```
# Verificamos el resultado usando multiplicación directa con sympy.
q=3329
x= Symbol('x')

p=[x for x in range(256)]
g=[(6*y+3) for y in range(256)]
n=len(p)

p_reorder= list(reversed(p)) # Esto porque sympy toma al revés los coeficientes.
g_reorder= list(reversed(g))

p = Poly(p_reorder, x,domain=GF(q))
g = Poly(g_reorder, x,domain=GF(q))

c=(p*g)%(x**(n)+1)

coef_an_a0 = c.all_coeffs()
coef_a0_an = list(reversed(coef_an_a0))
polinomio_mult_modq = [x%q for x in coef_a0_an] # Aplicamos reducción modular.

print(polinomio_mult_modq==p_mult_g) # Aquí comparamos.

True
```

Figura 10. Verificación de la multiplicación vía NTT.

A continuación se muestran los resultados obtenidos en la implementación presentada para las funciones principales, mostrando en Fig. 7 un caso de aplicación de la NTT_ψ y en Fig. 8 una comprobación de que la $INTT_\psi$ calcula el inverso de la transformada de manera correcta.

Esta implementación se consigue usando como punto de partida los Algoritmos 1 y 2, aplicados sobre una lista de entrada que tomaran los coeficientes de su respectivo polinomio de menor a mayor, siendo el output de la misma naturaleza.

V-C. *Función Kyber-Convolution*

En esta función se implementa el proceso descrito en la sección IV.

Se muestra en Fig. 9 un ejemplo definiendo dos polinomios de gran tamaño con $n = 256$ en $\mathbb{Z}_{3329}[x]/(x^{256}+1)$, creados a partir de expresiones simples utilizando un bucle, donde se verifica el resultado utilizando multiplicación directa por $SymPy$ en Fig. 10.

V-D. *Comparativa en tiempos de ejecución*

Se muestra la comparativa de tiempos de ejecución para las diferentes NTT de la implementación desarrollada y la de $SymPy$. Se incluye también comparación de la multiplicación de polinomios vía NTT y multiplicación directa.

Se debe recordar que realmente la NTT de $SymPy$ no es aplicable para la convolución en Kyber y es por ello, por lo que se incluyen los tiempos de la multiplicación directa de $SymPy$. Para la medición de los tiempos de ejecución se ha usado el paquete *time* de Python.

Se destacan los resultados realizados con los parámetros $q = 257$, $n = 16$ para la NTT, INTT, INTT $\circ$ NTT y parámetros $q = 3329$, $n = 256$ para la multiplicación en la I. Realmente tardan menos de 1 segundo pero al usar el módulo de time se ralentizan esos pasos, de todas formas interesa la comparativa por lo que no supone un inconveniente.

Tabla I
COMPARATIVA PARA LA IMPLEMENTACIÓN DE LA NTT

Código a comparar	En $SymPy$ (s)	Implementación propia (s)
NTT	1,0076167583465576	1,006410837173462
INTT	1,007662057876587	1,007622480392456
INTT $\circ$ NTT	1,009049892425537	1,006197214126587
Multiplicación	2,2660510540008545	1,2369062900543213

Se puede observar que existe una gran diferencia entre la multiplicación directa y la convolución, lo cuál es de esperar pues $SymPy$ además de hacer multiplicación directa calcula la reducción módulo $x^{256}+1$, una operación costosa. Además, debido a que la versión NTT de $SymPy$ es para un tipo de convolución que no necesita exponenciación de la raíz primitiva en los algoritmos Butterfly, tendría que ser menos costosa por lo que si están con valores de ejecución próximos entre sí significa que la implementación presentada no es una mala aproximación para lograr ilustrar la Transformada Teórica de Números.

VI. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se aporta un detallado desarrollo de la multiplicación polinomial sobre anillos cocientes mediante el uso de la Transformada Teórica de Números, elemento de algunos esquemas de cifrado resistentes a la computación

cuántica que se suele pasar por alto debido a su complejidad. Sin embargo, este elemento es crucial a la hora de lograr una implementación eficiente de los algoritmos que involucran criptografía post-cuántica basada en retículos. Por ello, se incluye en este documento un análisis que parte de la base teórica necesaria y una descripción de los algoritmos implicados (Cooley-Tukey y Gentleman-Sande), haciendo pruebas de implementación en Python y verificando la eficiencia de esta comparando tiempos de ejecución.

Una de las líneas de trabajo futuro es el estudio e implementación del esquema de cifrado CRYSTALS-Kyber al completo, haciendo uso de la eficiente multiplicación de polinomios que se ha presentado en este documento.

Agradecimientos

Este trabajo ha sido posible gracias a las Cátedras de Ciberseguridad de la Universidad de La Laguna patrocinadas por Binter y por INCIBE. Además forma parte de los proyectos: PID2022-138933OB-I00 financiado por MCIN/AEI/10.13039/501100011033/FEDER, UE, y SCITALA C064/23 ULL-INCIBE financiado con fondos del Plan de Recuperación, Transformación y Resiliencia, con financiación de la UE (Next Generation).

Referencias

[1] NIST: “NIST: an official website of the US government”, en: https://www.nist.gov/news-events/news/2023/08/nist-standardize-encryption-algorithms-can-resist-attack-quantum-computers.

[2] CRYSTALS Kyber: “Crystals Kyber resource”, en https://pq-crystals.org/kyber/index.shtml.

[3] CRYSTALS Dilithium: “Crystals Dilithium resource”, en https://pq-crystals.org/dilithium/index.shtml.

[4] SPHINCS+: “Stateless Hash-Based Digital Signature Standard”, en https://doi.org/10.6028/NIST.FIPS.205.ipd.

[5] Hung Nguyen, Linh Tran. “Design of Polynomial NTT and INTT Accelerator for Post-Quantum Cryptography CRYSTALS-Kyber”, en https://link.springer.com/article/10.1007/s13369-022-06928-w.

[6] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé. “Algorithm Specifications And Supporting Documentation”, en https://pq-crystals.org/kyber/resources.shtml.

[7] Lauritzen N. “Concrete Abstract Algebra: From Numbers to Gröbner Bases”. Cambridge University Press, 2003.

[8] Zhichuang Liang, Yunlei Zhao. “Number Theoretic Transform and Its Applications in Lattice-based Cryptosystems: A Survey”, en https://doi.org/10.48550/arXiv.2211.13546.

[9] Kenneth Ireland, Michael Rosen. “A Classical Introduction to Modern Number Theory”. Springer-Verlag, 1990.

[10] Mojtaba Bisheh Niasar, Reza Azarderakhsh, Mehran Mozaffari Kermani. “High-Speed NTT-based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography”, en https://eprint.iacr.org/2021/563.

[11] Sin-Wei Chiu, Keshab K. Parhi. “Long Polynomial Modular Multiplication using Low-Complexity Number Theoretic Transform”, en https://arxiv.org/abs/2306.12519.

[12] Longa, P., Naehrig, M. (2016). “Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. In: Foresti, S., Persiano, G. (eds) Cryptology and Network Security. CANS 2016”. Lecture Notes in Computer Science(), vol 10052. Springer, Cham. en: https://doi.org/10.1007/978-3-319-48965-0_8

[13] Tom Apostol. “Introducción A La Teoría Analítica De Los Números”. Springer-Verlag, 1984.

[14] SymPy: “SymPy”, en https://www.sympy.org/en/index.html. Consultado en: 24 de marzo de 2024.

[15] Antuñano Cabrera, N.: “Implementación NTT”https://colab.research.google.com/drive/1oUBPXxWyBsXkIiBWhCK6LchLmpE_Gqvd?usp=sharing. Consultado en: 24 de marzo de 2024.