PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE HÁBITAT, INFRAESTRUCTURA Y CREATIVIDAD

CARRERA DE SISTEMAS

PROYECTO DE TITULACIÓN

Tema: ANÁLISIS COMPARATIVO DE ALGORITMOS POST-CUÁNTICOS CRYSTALS-KYBER Y CRYSTALS-DILITHIUM FRENTE A ALGORITMOS CLÁSICOS RSA Y ECC PARA AUTENTICACIÓN Y TRANSFERENCIA SEGURA DE ARCHIVOS EN UN ENTORNO DE LABORATORIO DE COMUNICACIÓN SEGURA

AUTOR:

ABEL ANTONIO VÁSQUEZ VELARDE

TUTOR:

Ing. JORGE ALARCÓN

QUITO DM, ENERO 2026

DEDICATORIA

Este trabajo de titulación es dedicado a:

A mi madre, Isabel, por el incalculable amor con el que trabaja para darme la educación académica que no pudo tener.

A mi padre, Antonio, por las innumerables horas de velada para que tenga las oportunidades que no puedo tener.

A mi hermana, Milena, por ser el más puro ejemplo de excelencia académica.

AGRADECIMIENTO

Un título universitario lleva un solo nombre impreso, pero pesa por las historias de todos los que lo sostienen.

A mi madre, Isabel. Gracias por ser el ejemplo de equilibrio entre firmeza y ternura en mi vida. Porque las veces en las que aguantabas las lágrimas no fueron en vano, pues forjaron parte de la persona que soy hoy en día. Me diste la vida y razones para vivirla.

A mi padre, Antonio. Gracias por enseñarme la resiliencia no con palabras, sino con vida. En los momentos donde sentí que no podía más, recordaba tus batallas y encontraba la fuerza que me heredaste.

A mi hermana, Milena. Porque la excelencia se contagia y tú fuiste el espejo donde quise reflejarme. Gracias por marcar el estándar y demostrarme que se puede llegar lejos con integridad.

A mi primo, Jonathan. Porque desde pequeño fuiste quien me enseñó con su propia vida el significado de superación. Gracias por abrir el camino y demostrarme que, no importa cuán humilde sea nuestro origen, la grandeza de nuestro destino depende solo de nosotros.

A mis amigos. Ustedes son, sin duda, el tesoro más valioso e inesperado que me dejó esta carrera. Gracias por hacerme reír cuando solo quería llorar y por recordarme que el éxito no sabe a nada si no se comparte.

Y finalmente, porque las palabras humanas a veces se quedan cortas para tanto sentimiento, tomo prestada la despedida de un viejo hobbit: "No conozco a la mitad de ustedes ni la mitad de lo que querría, y lo que quiero a menos de la mitad de ustedes es la mitad de lo que se merecen".

ABSTRACT

El presente trabajo de titulación aborda la amenaza crítica que la computación cuántica representa para la seguridad de los criptosistemas asimétricos actuales vulnerando estándares fundamentales como RSA y ECC mediante la aplicación teórica del algoritmo de Shor por lo que en alineación con el actual proceso de estandarización liderado por el NIST esta investigación desarrolla un análisis comparativo de rendimiento técnico entre los nuevos algoritmos de Criptografía Post-Cuántica seleccionados específicamente CRYSTALS-Kyber y CRYSTALS-Dilithium contrastándolos frente a los algoritmos clásicos vigentes en escenarios de establecimiento de claves y firma digital.

La evaluación experimental se ejecutó dentro de un entorno de laboratorio controlado sobre sistema operativo Linux empleando scripts automatizados desarrollados en Python junto con la librería liboqs para cuantificar métricas críticas como el costo computacional, el consumo de memoria y la sobrecarga del canal en los niveles de seguridad NIST 1, 3 y 5 arrojando resultados empíricos que evidencian la obsolescencia práctica de RSA para la generación de claves efímeras al registrar tiempos miles de veces más lentos que sus competidores mientras que en el escenario KEM el algoritmo CRYSTALS-Kyber demostró una superioridad notable en el rendimiento del servidor durante la desencapsulación comparado con ECC aunque asumiendo el costo de un incremento en el tamaño de las claves.

Por su parte en el escenario DSA el análisis de CRYSTALS-Dilithium reveló una reingeniería del perfil de rendimiento al ofrecer una verificación más veloz que ECDSA en los niveles altos de seguridad a cambio de introducir una sobrecarga significativa en el ancho de banda con firmas digitales hasta treinta y cuatro veces más grandes, permitiendo concluir finalmente que la migración hacia estándares PQC resulta computacionalmente viable y beneficiosa para la eficiencia del servidor siempre y cuando se implementen estrategias para mitigar el impacto de la latencia de red derivada del mayor volumen de las primitivas criptográficas.

Palabras clave: Criptografía Post-Cuántica, PQC, CRYSTALS-Kyber, CRYSTALS-Dilithium, Rendimiento, RSA, ECC, Ciberseguridad

Tabla de contenido

Índice de ilustraciones

Índice de figuras

Contenido de tablas

CAPÍTULO I: INTRODUCCIÓN

1.1 Justificación

Desde la concepción teórica de la computación cuántica por Paul Benioff en 1980, hasta el desarrollo del procesador Osprey de 433 cúbits por IBM en 2022, la evolución de esta tecnología ha planteado un desafío a la seguridad digital. La viabilidad del algoritmo de Shor demuestra que los sistemas criptográficos tradicionales, como RSA y ECC, resultarán vulnerables ante ataques cuánticos a gran escala. Esta situación representa una amenaza crítica para la integridad de los datos en los sistemas actuales. La pérdida de confianza en la capacidad de cifrado implicaría que la información sensible sea susceptible de descifrado retroactivo. Este fenómeno, denominado "cosechar ahora y descifrar después" (Harvest Now, Decrypt Later), sugiere que actores malintencionados podrían estar capturando tráfico de red en la actualidad con el fin de procesarlo en el futuro.

En respuesta a esta vulnerabilidad, el NIST ha evaluado diversos algoritmos candidatos para establecer estándares de criptografía post-cuántica (PQC). Como resultado de este proceso, se han seleccionado los algoritmos CRYSTALS-Kyber, para el cifrado y establecimiento de claves, y CRYSTALS-Dilithium, para firmas digitales.

El presente trabajo de titulación tiene como propósito analizar y comparar el rendimiento de estos nuevos estándares frente a RSA y ECC en un entorno de laboratorio controlado. Para la evaluación, se emplean herramientas especializadas como liboqs (Open Quantum Safe), PQClean y OpenSSL con soporte PQC. El estudio se enfoca en la cuantificación de métricas técnicas, tales como el tiempo de ejecución, el consumo de memoria y la eficiencia en la transferencia de archivos. La investigación resulta de utilidad para organizaciones que busquen anticipar la transición post-cuántica, al proveer información técnica sobre la viabilidad operativa de estos esquemas. De este modo, se busca contribuir a una migración fluida hacia los estándares seleccionados por el NIST sin comprometer el rendimiento de los sistemas de información.

1.2 Planteamiento del problema

La computación se encuentra en el umbral de una revolución con implicaciones profundas para la seguridad global. La historia de la computación cuántica se remonta a las primeras décadas de la física teórica, pero su materialización como una herramienta de ingeniería comenzó a tomar forma en 1980, cuando Paul Benioff describió el primer modelo teórico de una computadora cuántica. Esta idea fue complementada por Richard Feynman en 1982, quien sugirió que tales máquinas serían ideales para simular sistemas cuánticos; una tarea intratable para las computadoras clásicas.

Sin embargo, el panorama cambió cuando en 1994 el matemático Peter Shor desarrolló un algoritmo cuántico capaz de factorizar números enteros grandes y calcular logaritmos discretos en tiempo polinomial. La magnitud de este descubrimiento radica en que estos dos problemas matemáticos —tanto la dificultad de la factorización (en la que se basa el algoritmo RSA) como la dificultad de calcular logaritmos discretos (en la que se basa la criptografía de curva elíptica, ECC)— son los pilares que sostienen la infraestructura de clave pública (PKI) moderna. Todo, desde las transacciones bancarias en línea (protocolos TLS/SSL) y las redes privadas virtuales (VPN), hasta las firmas digitales que garantizan la autenticidad del software, depende de que estos problemas sean extremadamente difíciles de resolver para las computadoras clásicas.

Durante décadas, la amenaza de Shor fue teórica. No obstante, la última década ha sido testigo de una aceleración en el desarrollo de hardware cuántico. Empresas y laboratorios como Google, IonQ e IBM han entrado en una carrera por la supremacía cuántica. Un ejemplo es la presentación del procesador "Osprey" de IBM en 2022, con 433 cúbits. Aunque estos procesadores aún son "ruidosos" (NISQ - *Noisy Intermediate-Scale Quantum*) y no son capaces de ejecutar el algoritmo de Shor a una escala que rompa cifrados reales, la trayectoria es constante: la interrogante ya no es si una computadora cuántica criptográficamente relevante (CRQC) será construida, sino cuándo ocurrirá y su disponibilidad en el mercado.

Esta evolución ha convertido la migración hacia una criptografía resistente a los cuánticos (PQC) en una prioridad de seguridad nacional y corporativa. Los algoritmos tradicionales como RSA y ECC se volverán obsoletos. Esta vulnerabilidad representa una amenaza crítica, pero el riesgo no comienza el día que se complete una CRQC. El problema más apremiante se conoce como "Cosechar Ahora, Descifrar Después" (*Harvest Now, Decrypt Later* - HNDL). Actores maliciosos, con el respaldo de recursos significativos de almacenamiento, interceptan y archivan volúmenes masivos de datos cifrados en la actualidad. Estos datos, protegidos por los estándares vigentes, son incomprensibles por ahora; sin embargo, una vez que una computadora cuántica funcional esté disponible, esta biblioteca de datos podrá ser descifrada retroactivamente. Esto implica que cualquier información con un período de confidencialidad largo —secretos de estado, propiedad intelectual o registros médicos— que se esté transmitiendo hoy, ya se encuentra comprometida. La amenaza cuántica es, por tanto, un riesgo activo que compromete la confidencialidad a largo plazo.

En respuesta a esta amenaza, el Instituto Nacional de Estándares y Tecnología (NIST) inició un proceso de estandarización global en 2016. El objetivo fue evaluar y seleccionar algoritmos seguros tanto contra ataques clásicos como cuánticos. En julio de

2022, el NIST anunció los primeros cuatro algoritmos para la estandarización. Entre ellos, CRYSTALS-Kyber (mecanismo de encapsulamiento de clave, KEM) fue seleccionado como el estándar principal para el cifrado e intercambio de claves, y CRYSTALS-Dilithium como el estándar principal para las firmas digitales. Ambos se basan en la dificultad de problemas matemáticos en retículos estructurados.

No obstante, esta estandarización introduce un nuevo conjunto de problemas de ingeniería. Los algoritmos PQC presentan perfiles de rendimiento distintos a los de RSA y ECC. Específicamente, tienden a tener tamaños de clave pública, clave privada y firmas significativamente más grandes. Por ejemplo, una clave pública de ECC (NIST P-256) puede ser de 64 bytes; en contraste, la de CRYSTALS-Kyber-512 es de 800 bytes, y una firma de CRYSTALS-Dilithium-2 supera los 2400 bytes. Este incremento tiene implicaciones directas en:

- **Latencia de Red:** El mayor tamaño de clave y firma incrementa los datos a transmitir durante el establecimiento de una conexión segura (*handshake* de TLS).
- **Consumo Computacional (CPU):** Las operaciones en retículos son diferentes a las de RSA/ECC, y resulta incierto cómo se traducirá esto en el consumo de CPU.
- **Uso de Memoria (RAM):** Los nuevos algoritmos pueden requerir más memoria para almacenar claves o estados intermedios.
- **Dispositivos de Recursos Limitados (IoT):** Esta sobrecarga es crítica para dispositivos que operan con poder de cómputo y ancho de banda limitados.

Por lo tanto, emerge un vacío de conocimiento: existe una brecha entre la selección teórica de algoritmos (NIST) y la comprensión práctica de su impacto en el rendimiento. Las organizaciones no pueden realizar una migración sin datos empíricos, ya que una transición mal planificada podría degradar el rendimiento de los servicios. El problema central que aborda esta investigación es la ausencia de un *benchmark* de rendimiento comparativo entre los esquemas PQC (CRYSTALS-Kyber, CRYSTALS-Dilithium) y los esquemas clásicos (RSA, ECC) en un entorno de laboratorio controlado. Aunque herramientas como liboqs, PQClean y OpenSSL integran estos algoritmos, se requieren estudios sistemáticos que midan el impacto de su adopción en métricas clave.

En resumen, este trabajo aborda la siguiente pregunta central: **¿Cuál es la sobrecarga de rendimiento y la viabilidad práctica de reemplazar RSA y ECC por CRYSTALS-Kyber y CRYSTALS-Dilithium en un sistema de comunicación segura, medido en términos de tiempo de ejecución, uso de memoria y eficiencia de transferencia?**

1.3 Objetivos

1.3.1 Objetivo general

Evaluar el rendimiento técnico de los algoritmos criptográficos post-cuánticos CRYSTALS-Kyber y CRYSTALS-Dilithium frente a los estándares clásicos RSA y ECC, mediante pruebas experimentales en un entorno de comunicación segura, para determinar su viabilidad operativa en procesos de autenticación y transferencia de archivos.

1.3.2 Objetivos específicos

- Analizar los fundamentos matemáticos y el funcionamiento de los algoritmos CRYSTALS-Kyber, CRYSTALS-Dilithium, RSA y ECC para identificar las diferencias teóricas en el tamaño de sus primitivas y su resistencia ante el algoritmo de Shor.
- Medir el costo computacional en nanosegundos, el pico de consumo de memoria RAM y la sobrecarga del canal en bytes para cada algoritmo en los niveles de seguridad 1, 3 y 5 del NIST.
- Implementar en un entorno de laboratorio los algoritmos CRYSTALS-Kyber, CRYSTALS-Dilithium, RSA y ECC utilizando herramientas de software reconocidas para establecer una base experimental de comparación.
- Contrastar los resultados obtenidos mediante un análisis estadístico comparativo para establecer el balance técnico (trade-off) entre la seguridad post-cuántica y la eficiencia operativa de los sistemas.

1.4 Alcance

Este proyecto de titulación establece un alcance estrictamente experimental y comparativo, enfocado en la evaluación de la viabilidad de los estándares de Criptografía Post-Cuántica (PQC) seleccionados por el Instituto Nacional de Estándares y Tecnología (NIST) frente a los algoritmos clásicos de clave pública. El objetivo primordial es trascender la evaluación teórica para proveer datos de rendimiento aplicados que justifiquen la inminente transición criptográfica.

1.4.1 Foco Algorítmico y Niveles de Seguridad

El análisis se centrará en un conjunto específico de algoritmos de clave pública, abarcando tanto los estándares PQC seleccionados por el NIST como sus contrapartes clásicas. Para realizar un análisis empírico del costo computacional y evaluar cómo escala el rendimiento, la comparación no se limitará a un solo punto de datos, sino que se evaluará en los tres principales niveles de seguridad definidos por el NIST.

Los algoritmos específicos seleccionados para cada nivel son:

- Nivel de Seguridad 1 (Equivalente a 128-bit simétrico):
  - PQC-KEM: CRYSTALS-Kyber-512
  - PQC-DSA: CRYSTALS-Dilithium-2
  - Clásico-KEM: RSA-3072 y ECDH con curva P-256.
  - Clásico-DSA: ECDSA con curva P-256.
- Nivel de Seguridad 3 (Equivalente a 192-bit simétrico):
  - PQC-KEM: CRYSTALS-Kyber-768
  - PQC-DSA: CRYSTALS-Dilithium-3
  - Clásico-KEM: RSA-7680 y ECDH con curva P-384.
  - Clásico-DSA: ECDSA con curva P-384.

Nota: RSA se omite desde este nivel, ya que su clave equivalente (7680 bits) es computacionalmente inviable para la mayoría de las comparaciones prácticas.

- Nivel de Seguridad 5 (Equivalente a 256-bit simétrico):
  - PQC-KEM: CRYSTALS-Kyber-1024
  - PQC-DSA: CRYSTALS-Dilithium-5
  - Clásico-KEM: ECDH (con curva P-521)
  - Clásico-DSA: ECDSA (con curva P-521)

1.4.2 Entorno de Implementación y Herramientas

Los algoritmos serán implementados en un entorno de laboratorio controlado, diseñado para asemejarse a una estructura de tipo cliente-servidor, sin validar un sistema productivo a escala real u otro tipo de infraestructura adicional. Este entorno se desplegará utilizando el software de virtualización VirtualBox (v. 7.1.8). Consistirá en dos máquinas virtuales (VM) que representan a los actores de la comunicación usados generalmente como ejemplos, Alice y Bob, conectadas a través de una red virtual interna aislada para

simular un canal de comunicación dedicado y permitir mediciones de red precisas. Para reflejar un caso de uso asimétrico real, las máquinas virtuales tendrán roles y configuraciones distintas:

- VM Cliente (Alice): Se implementará sobre Xubuntu 22.04 LTS. Esta distribución simula una estación de trabajo de usuario, proveyendo un entorno gráfico ligero (XFCE) que permite un desarrollo más ágil sin impactar significativamente en las métricas de rendimiento. Esta máquina virtual contará con 3vCPUs y 4 GB de RAM.
- VM Servidor (Bob): Se implementará sobre Ubuntu Server 22.04 LTS. Esta configuración simula un servidor de producción controlado exclusivamente desde la terminal, o sea sin entorno gráfico, optimizado para un rendimiento puro y un consumo mínimo de recursos base. Asimismo, esta máquina virtual contará con 3vCPUs y 4 GB de RAM.

Para establecer una base experimental sólida se utilizarán librerías ampliamente utilizadas y reconocidas en la comunidad de criptografía post:

- Open Quantum Safe (OQS) y la librería liboqs: Se aprovecharán sus implementaciones estandarizadas y sus rutinas de benchmarking para KEMs y DSAs resistentes a la cuántica.
- PQClean: Se utilizará para la obtención de implementaciones limpias de los algoritmos.
- OpenSSL con soporte PQC: Se empleará para simular la integración de los algoritmos en protocolos de comunicación seguros como TLS 1.3, esenciales para los escenarios de autenticación y transferencia.

El alcance de la implementación se limita a la capa de software, utilizando arquitecturas de CPU de propósito general, sin incluir la evaluación o diseño de aceleradores de hardware especializados o sistemas empotrados de bajo consumo, aunque los resultados podrán servir de referencia para dichos entornos.

CAPÍTULO II: MARCO TEÓRICO

2.1 Antecedentes

Los estándares actuales que soportan la comunicación global se basan en la robustez de los algoritmos criptográficos asimétricos Durante décadas, la fortaleza de estos sistemas ha dependido de la inviabilidad computacional para resolver ciertos problemas matemáticos utilizando ordenadores clásicos.

No obstante, la inminente llegada de la computación cuántica representa una amenaza crítica. Este campo, cuyo modelo teórico inicial fue diseñado por Paul Benioff en 1980 describiéndolo como un sistema físico (Benioff, 1980), aprovecha los principios de la mecánica cuántica para resolver problemas complejos de manera que superan las capacidades de las máquinas clásicas. El principal motivo de preocupación es el algoritmo de Shor, desarrollado en 1994 por Peter Shor. Este algoritmo cuántico tiene la capacidad de resolver eficientemente el Problema de Factorización de Enteros (IFP), que es la base de RSA, y el Problema de Logaritmo Discreto de Curva Elíptica (ECDLP), que es la base de ECC. La importancia de estas amenazas radica en el hecho de que el algoritmo de Shor proporciona una herramienta para resolver estos problemas exponencialmente más rápido que cualquier método clásico, por ende, el problema desaparece cuando se utiliza una computadora cuántica. Durante décadas, esta amenaza fue puramente teórica. Sin embargo, el desarrollo de hardware cuántico se ha acelerado exponencialmente. Un ejemplo de este avance es la presentación del procesador "Osprey" de IBM en 2022, un dispositivo que alcanzó los 433 cúbits (bits cuánticos) (IBM, 2022). Aunque los procesadores actuales aún no pueden romper cifrados reales, la trayectoria es clara.

La consecuencia de esta vulnerabilidad es que, si un ordenador cuántico a gran escala se materializa, podría descifrar la mayor parte de las comunicaciones y los datos sensibles protegidos por RSA y ECC. Esta situación genera una urgencia denominada "Store Now, Decrypt Later" (Almacenar ahora, descifrar después) o "Harvest Now, Decrypt Later”. Bajo este escenario, los atacantes pueden interceptar y almacenar hoy las comunicaciones cifradas, y esperar a que el ordenador cuántico esté disponible para descifrarlas a gran escala, lo que provocaría un colapso del marco de seguridad existente.

2.2 Marco teórico

La computación cuántica representa uno de los más grandes desafíos para los algoritmos criptográficos tradicionales cuya seguridad está fundamentada en la incapacidad de los computadores clásicos de resolver el IFP que consiste en la dificultad computacional de encontrar factores primos de un numero entero grande N. Sin embargo, el algoritmo de Shor permite resolverlos de manera exponencial dejando a los algoritmos de encriptación actuales incapaces de asegurar la integridad de los datos encriptados. Ante este escenario, la criptografía post-cuántica (PQC, por sus siglas en inglés) surge como un campo de investigación orientado a desarrollar algoritmos resistentes a ataques cuánticos, garantizando tanto la integridad como la confidencialidad de los datos, dos de los tres pilares de la ciberseguridad.

Para hallar una solución a este problema desde el 2016 el Instituto Nacional de Estándares y Tecnología (NIST) ha impulsado un proceso de estandarización en el que destacan algoritmos basados en retículas, como CRYSTALS-Kyber para el intercambio de claves que asegura la confidencialidad de los datos y CRYSTALS-Dilithium para la firma digital que asegura la integridad de los datos, debido a su solidez matemática y eficiencia práctica. La aplicación de PQC es de alta relevancia en la mensajería y transferencia de archivos, ya que estos sistemas son la base de la comunicación segura. La implementación de CRYSTALS-Kyber y CRYSTALS-Dilithium en estos entornos de laboratorio controlados permite evaluar su viabilidad práctica más allá de la teoría. Por lo tanto, un análisis comparativo de su rendimiento midiendo tiempo de ejecución, uso de recursos y tamaño de claves es un paso necesario para determinar la factibilidad de su adopción y facilitar la transición a estos nuevos estándares.

2.2.1 Criptografía Post-Cuántica (PQC) y su distinción con QKD

Es fundamental establecer la distinción entre criptografía post-cuántica y la criptografía cuántica que se suele identificar como el proyecto de Distribución Quántica de Llaves (QKD, por sus siglas en inglés) debido a que tienen diferencias tanto conceptuales como tecnológicas.

La diferencia se encuentra en la solución que ofrece: la criptografía cuántica es una solución de hardware, distribuye claves con tecnología cuántica, basándose en el principio de incertidumbre de Heisenberg, mientras que PQC es una solución de software

que permite a los ordenadores clásicos poder establecer una conexión segura con otro asegurando la dificultad criptográfica de las primitivas de comunicación.

Esta es la distinción clave:

- PQC: es una actualización de software a las comunicaciones actuales cuyas conexiones pueden ser vulneradas por computadores cuánticos mediante el algoritmo de Shor, implementando problemas intratables para computadores cuánticos.
- QKD: es una solución a largo plazo para la distribución de llaves, aún se necesita de infraestructura como enlaces ópticos directos y solo aborda la solución de la distribución de llaves, no de firma digital.

Este proyecto es un análisis empírico de PQC porque abordan directamente las primitivas criptográficas que sostienen los sistemas de comunicación actuales. PQC es una respuesta ante la llegada de la computación cuántica y con ello el problema de recolectar ahora, desencripta después con la modernización de los sistemas existentes de comunicación y transferencia segura de archivos.

2.2.2 Amenazas y vulnerabilidades residuales

Aunque los algoritmos de criptografía post-cuántica están diseñados para mitigar el algoritmo de Shor, la implementación de estos algoritmos introduce nuevas vulnerabilidades y expone a los sistemas que los acogen a ellos.

Para fortalecer la evaluación de seguridad se requiere un análisis detallado de los siguientes escenarios:

2.2.2.1 Ataque de canal lateral y vulnerabilidades residuales

Los algoritmos CRYSTALS-Kyber y CRYSTALS-Dilithium por su naturaleza matemática introducen complejidades operacionales que pueden ser explotadas por atacantes:

- Ataques de canal lateral (Side-Channel Attack): este tipo de ataque mide las emisiones físicas colaterales de un dispositivo durante el cómputo criptográfico como: tiempo de ejecución, consumo de energía o la radiación electromagnética. La vulnerabilidad reside en que la implementación de los algoritmos PQC utilizan operaciones que dependen de la información, lo que puede filtrar información sobre la clave secreta

ya que la complejidad de las multiplicaciones de los polinomios puede variar en función del texto a cifrar.

- Ataques de implementación: estos tipos de ataques explotan las fallas lógicas o en la implementación del sistema operativo, como, por ejemplo, el manejo de redondeo de decimales en los algoritmos de retículas podría filtrar bits de información sobre la clave secreta.

2.2.2.2 Escenarios de ataques específicos

La robustez de la evaluación de seguridad debe incluir la consideración de escenarios que van más allá ataques con el algoritmo de Shor, algunos pueden ser:

- Ataques de fallos (Fault Injection): escenarios donde un atacante induce fallos deliberados en el hardware del servidor como picos de voltaje durante la ejecución de una primitiva criptográfica, lo que puede desencadenar en que fallen de una forma controlada, lo que permite al atacante reconstruir la clave secreta.
- Ataques de Oráculo (Oracle Attack): el atacante utiliza una repuesta de error como una especie de pista para intentar adivinar la clave, por ejemplo, el servidor responde más rápido cuando la clave encapsulada es casi correcta, el atacante puede usar esa diferencia de tiempo para adivinar bits de la clave.

2.2.3 Retos prácticos de integración y viabilidad operacional

La transición de algoritmos clásicos hacia los algoritmos post-cuánticos conlleva desafíos de ingeniería más allá del costo computacional de la CPU. La viabilidad de PQC depende íntimamente de su implementación en infraestructura existente debido a que está ligada a un problema actual y urgente discutido previamente como es la amenaza de cosechar ahora, desencriptar después.

2.2.3.1 Impacto en protocolos y sobrecarga del canal

El principal desafío de implementación es el tamaño de las primitivas criptográficas, debido a la naturaleza de los algoritmos post-cuánticos, estas son considerablemente más grandes. Además, la arquitectura de los protocolos TLS/SSL ampliamente utilizados fueron diseñados para manejar primitivas más pequeñas, como las generadas por los algoritmos criptográficos clásicos, por ejemplo, ECC.

- Latencia y ancho de banda: Los algoritmos PQC analizados al tener signaturas considerablemente más grandes, ocupan un ancho de banda mayor por la cantidad de datos que deben transmitirse durante el establecimiento de la conexión (handshake), esto puede conllevar una mayor latencia, especialmente en conexiones con alta pérdida de paquetes.
- Buffer: La inserción de claves PQC en el handshake de protocolos existentes puede llegar a exceder los límites establecidos de los segmentos de red y conlleva a una reconfiguración de la capa de red.

2.2.3.2 Desafíos de compatibilidad y consumo energético

La integración de PQC con los dispositivos IOT (Internet de las Cosas) es otro de los grandes retos de implementación debido a que los dispositivos cuentan con recursos limitados, tales como: poca memoria RAM, bajo ancho de banda y un limitado poder de cómputo hace que no puedan manejar una huella de memoria más grande cuando se ejecutan las operaciones de generación de llave o firma digital. Además, el consumo energético asociado a las operaciones que conllevan los algoritmos puede reducir la vida útil de la batería integrada en el dispositivo.

Estos retos demuestran que la migración de algoritmos clásicos a PQC no es un simple cambio de algoritmo, sino una serie de decisiones complejas de ingeniería que requieren una discusión detallada entre eficiencia operativa de los sistemas y la seguridad futura.

2.3 Marco conceptual

2.3.1 Computación cuántica

La computación cuántica es un campo emergente de la informática y la ingeniería que aprovecha las cualidades únicas de la mecánica cuántica para resolver problemas más allá de la capacidad incluso de los ordenadores clásicos más potentes. El campo de la computación cuántica incluye una serie de disciplinas, como el hardware cuántico y los algoritmos cuánticos. Mientras siga en desarrollo, la tecnología cuántica pronto podrá resolver problemas complejos que los superordenadores clásicos no pueden resolver (o no pueden resolver lo suficientemente rápido). (Schneider & Smalley, 2025)

2.3.2 Cúbits

Según (Schneiider & Smalley, 2024) un cúbit, o bit cuántico, es la unidad básica de información utilizada para codificar datos en la computación cuántica y puede entenderse mejor como el equivalente cuántico del bit tradicional utilizado por las computadoras clásicas para codificar información en un sistema binario. El término "cúbit" se atribuye al físico teórico estadounidense Benjamin Schumacher. En general, los cúbits se crean, aunque no exclusivamente, manipulando y midiendo partículas cuánticas (los bloques más pequeños conocidos del universo físico), como fotones, electrones, iones atrapados, circuitos superconductores y átomos. Gracias a las propiedades únicas de la mecánica cuántica, las computadoras cuánticas utilizan cúbits para almacenar más datos que los bits tradicionales, mejorar enormemente los sistemas criptográficos y realizar cálculos muy avanzados que tardarían miles de años (o serían imposibles) incluso para las supercomputadoras clásicas. Impulsadas por cúbits, es posible que, a corto plazo, las computadoras cuánticas puedan resultar fundamentales para resolver muchos de los mayores desafíos de la humanidad, como el cáncer y otras investigaciones médicas, el cambio climático, el aprendizaje automático y la inteligencia artificial (IA).

2.3.3 Criptografía

La criptografía es la práctica de desarrollar y utilizar algoritmos codificados para proteger y ocultar la información transmitida para que solo puedan ser leídas por aquellos con permiso y capacidad de descifrarla. Dicho de otro modo, la criptografía oculta las comunicaciones para que las partes no autorizadas no puedan acceder a ellas. (IBM, 2025)

2.3.4 Criptografía cuántica

La criptografía cuántica es un conjunto de métodos que utilizan las peculiares, pero bien entendidas, reglas de la mecánica cuántica para cifrar, transmitir y decodificar información de forma segura. La criptografía cuántica emplea dispositivos cuánticos, como sensores capaces de registrar partículas individuales de luz (fotones), para proteger los datos de ataques adversarios. Aunque técnicamente compleja, la criptografía cuántica promete ventajas sobre los sistemas criptográficos clásicos no cuánticos. Por ejemplo, el enfoque cuántico tiene el potencial de detectar y frustrar mejor a los intrusos que intentan interceptar datos. (NIST, 2025)

Además, la criptografía cuántica tiene varios tipos tales como: basada en posición, independiente del dispositivo, simétrica y asimétrica de tal manera que cada una tiene sus ventajas y desventajas, sin embargo, la más utilizada hoy en día es la criptografía cuántica de distribución de claves cuánticas que según (Schneider & Smalley, 2023).

2.3.5 KEM

Un mecanismo de encapsulación de claves (KEM, por sus siglas en inglés) según (NIST, 2025) es un conjunto de algoritmos que, bajo ciertas condiciones, permite a dos partes establecer una clave secreta compartida a través de un canal público. Una clave secreta compartida, establecida de forma segura mediante un KEM, puede utilizarse con algoritmos criptográficos de clave simétrica para realizar tareas básicas en comunicaciones seguras, como el cifrado y la autenticación. Esta norma específica un mecanismo de encapsulación de claves denominado ML-KEM. La seguridad de ML-KEM está relacionada con la dificultad computacional del problema de Aprendizaje de Módulos con Errores. Actualmente, se considera que ML-KEM es seguro, incluso frente a adversarios que poseen una computadora cuántica. Esta norma específica tres conjuntos de parámetros para ML-KEM. En orden creciente de seguridad y decreciente de rendimiento, estos son ML-KEM-512, ML-KEM-768 y ML-KEM-1024.

2.3.6 DSS

El estándar de firma digital (DSS, por sus siglas en inglés) es definido por (NIST, 2025) como un conjunto de algoritmos que pueden utilizarse para generar una firma digital. Las firmas digitales se emplean para detectar modificaciones no autorizadas de los datos y para autenticar la identidad del firmante. Además, el destinatario de los datos firmados puede utilizar una firma digital como prueba para demostrar a un tercero que la firma fue generada, en efecto, por el firmante declarado. Esto se conoce como no repudio, ya que el firmante no puede repudiar fácilmente la firma posteriormente.

2.3.7 CRYSTALS-Kyber

Kyber es un mecanismo de encapsulación de claves (KEM) seguro según IND-CCA2, cuya seguridad se basa en la dificultad de resolver el problema de aprendizaje con errores (LWE) sobre redes modulares. Kyber es uno de los finalistas del proyecto de criptografía post-cuántica del NIST. La propuesta incluye tres conjuntos de parámetros diferentes que buscan distintos niveles de seguridad. En concreto, Kyber-512 busca una seguridad similar a la de AES-128, Kyber-768 busca una seguridad similar a la de AES-192 y Kyber-1024 busca una seguridad similar a la de AES-256. (CRYSTALS, 2020)

2.3.8 CRYSTALS-Dilithium

Dilithium es un esquema de firma digital altamente seguro contra ataques de mensajes seleccionados, basado en la complejidad de los problemas de red sobre las redes de módulos. Este concepto de seguridad implica que un adversario con acceso a un oráculo de firma no puede generar la firma de un mensaje cuya firma aún no ha visto, ni generar una firma diferente de un mensaje que ya ha visto firmado. Dilithium es uno de los algoritmos candidatos presentados al proyecto de criptografía post-cuántica del NIST. (CRYSTALS, 2021)

2.3.9 Algoritmo RSA

En 1978, Ron Rivest, Adi Shamir y Leonard Adleman introdujeron un algoritmo criptográfico que, en esencia, sustituyó al algoritmo menos seguro de la Oficina Nacional de Normas (NBS). Lo más importante es que RSA implementa un criptosistema de clave pública, así como firmas digitales. RSA se inspira en los trabajos publicados por Diffie y Hellman varios años antes, quienes describieron la idea de dicho algoritmo, pero nunca llegaron a desarrollarlo. Introducido en un momento en que se esperaba la llegada del correo electrónico, RSA implementó dos ideas importantes:

- Cifrado de clave pública. Esta idea elimina la necesidad de un "mensajero" que entregue las claves a los destinatarios a través de otro canal seguro antes de transmitir el mensaje original. En RSA, las claves de cifrado son públicas, mientras que las claves de descifrado no lo son, por lo que solo quien tenga la clave de descifrado correcta puede descifrar un mensaje cifrado. Cada persona tiene sus propias claves de cifrado y descifrado. Las claves deben crearse de tal manera que la clave de descifrado no pueda deducirse fácilmente de la clave de cifrado pública.
- Firmas digitales. El receptor puede necesitar verificar que un mensaje transmitido realmente se originó en el remitente (firma) y no simplemente desde allí (autenticación). Esto se realiza mediante la clave de descifrado del remitente, y la firma puede ser verificada posteriormente por cualquier persona mediante la clave de cifrado pública correspondiente. Por lo tanto, las firmas no pueden falsificarse. Además, ningún firmante puede negar posteriormente haber firmado el mensaje. (Milanov, 2009)

2.3.10 Algoritmo ECC

La criptografía de curva elíptica es un tipo criptográfico importante en la criptografía de clave pública, que se basa en una curva elíptica para cifrar y descifrar. Una curva elíptica es una curva afín suave con género 1 en el dominio, y su expresión puede escribirse como

$$y^2 = x(x-1)(x \quad -\lambda), \lambda \neq 0,1$$

ó

$$y^2 + ay = x^3 + bx^2 + cx + d$$

Si las características del dominio no son 2 y 3, entonces también se puede escribir como:

$$y^2 = x^3 + ax + b$$

Las curvas elípticas tienen diversas aplicaciones, como la criptografía de curva elíptica (ECC), el algoritmo de firma digital de curva elíptica (ECDSA), etc. (Yan, 2019)

2.3.11 Algoritmo de Shor

Según (FORTINET, 2025) es un algoritmo cuántico diseñado para factorizar de manera eficiente números enteros grandes, una tarea computacionalmente inviable para los ordenadores clásicos. La seguridad de los sistemas criptográficos ampliamente utilizados, como RSA, se basa en la dificultad de factorizar números grandes. Cuando se implementa en un ordenador cuántico suficientemente potente, el algoritmo de Shor puede romper el cifrado RSA reduciendo exponencialmente el tiempo necesario para encontrar factores primos, lo que hace que los métodos de cifrados clásicos sean vulnerables.

2.3.12 Cifrado asimétrico

El cifrado asimétrico, conocido como cifrado de clave pública, funciona creando un par de claves, una clave pública que cualquiera puede utilizar para cifrar datos y una clave privada, que solo los que la posean podrán descifrar esos datos. La principal ventaja del cifrado asimétrico es que elimina la necesidad de un intercambio de claves seguro, que es el punto más vulnerable del cifrado simétrico. Sin embargo, el cifrado asimétrico es notablemente más lento y consume una mayor cantidad de recursos que el cifrado simétrico siendo este el principal motivo por el que las organizaciones y aplicaciones de mensajería confían en un método de cifrado hibrido, que utiliza el cifrado asimétrico para la distribución de claves y el cifrado simétrico para el intercambio de datos posterior.

La clave pública y privada funcionan como códigos complejos necesarios para abrir una caja fuerte, si las claves no coinciden, o la clave privada es incorrecta, los usuarios no pueden descifrar el texto original. En general, cuanto mayor sea el tamaño de la clave, mayor será la seguridad, aunque también se debe tomar en cuenta, como es el caso de este proyecto, que longitud me conviene en términos de rendimiento sin perder la seguridad

En el cifrado asimétrico, las dos claves sirven para diferentes propósitos:

- La clave pública: cifra los datos o verifica las firmas digitales y se puede distribuir y compartir libremente.
- La clave privada: descifra los datos y crea firmas digitales, pero debe permanecer en secreto para garantizar la seguridad.

El cifrado asimétrico también puede funcionar como una especie de firma digital para garantizar la autenticación. Por ejemplo, un remitente puede cifrar un mensaje utilizando su clave privada y enviarlo a un destinatario. El destinatario puede utilizar la clave pública del remitente para descifrar el mensaje, confirmando así que fue el remitente original quien lo envió (IBM, 2024).

2.3.13 Infraestructura de clave pública (PKI)

La infraestructura de clave pública (PKI) se refiere a las herramientas utilizadas para crear y administrar claves públicas para el cifrado, un método común para proteger las transferencias de datos en internet. La PKI está integrada en todos los navegadores web actuales y ayuda a proteger el tráfico de internet público. Las organizaciones pueden

usarla para proteger sus comunicaciones internas y garantizar que los dispositivos conectados se conecten de forma segura. […] La infraestructura de clave pública (PKI) funciona mediante la implementación de dos tecnologías: certificados y claves. Una clave es un número largo que se utiliza para cifrar datos. Cada elemento de un mensaje se cifra utilizando la fórmula de la clave. Por ejemplo, si se desea escribir un mensaje donde cada letra se reemplaza por la siguiente, entonces A se convertirá en B, C en D, etc. Si alguien posee esta clave, recibirá un mensaje aparentemente sin sentido y podrá descifrarlo.

Con PKI, la clave implica conceptos matemáticos avanzados mucho más complejos. En el ejemplo alfabético anterior, existe una sola clave, y si el destinatario la posee, puede descifrar fácilmente el mensaje. En cambio, con PKI, existen dos claves: una privada y una pública. La clave pública está disponible para cualquiera que la solicite y se utiliza para cifrar un mensaje que alguien envía. La clave privada se utiliza para descifrar el mensaje una vez recibido. Aunque las claves privada y pública están relacionadas, esta relación se facilita mediante dicha ecuación. Por lo tanto, es extremadamente difícil determinar la clave privada utilizando los datos de la clave pública. Los certificados, emitidos por una autoridad de certificación (CA), permiten verificar que la persona o el dispositivo con el que se desea comunicar es quien dice ser. Cuando el certificado correcto está asociado a un dispositivo, este se considera auténtico. La validez del certificado se puede autenticar mediante un sistema que comprueba su autenticidad. El concepto más importante asociado con la PKI son las claves criptográficas, que forman parte del proceso de cifrado y sirven para autenticar a las personas o dispositivos que intentan comunicarse con la red. (FORTINET, 2025)

2.3.14 Complejidad algorítmica

Según (Cormen, Leiserson, Rivest, & Stein, 2022) la complejidad algorítmica es una medida de cuántos recursos computacionales (principalmente tiempo y espacio o memoria) consume un algoritmo para ejecutarse, en función del tamaño de los datos de entrada. No mide el tiempo exacto en segundos, sino cómo crece el número de operaciones a medida que aumentan los datos. Se utiliza para comparar la eficiencia y escalabilidad de diferentes algoritmos.

- Complejidad Temporal (Tiempo): Cuántas operaciones o pasos realiza el algoritmo.
- Complejidad Espacial (Espacio): Cuánta memoria necesita el algoritmo.

CAPÍTULO III: CONFIGURACIÓN DEL ENTORNO DE LABORATORIO

3.1 Diseño del entorno de laboratorio

Para el diseño de este entorno de laboratorio se implementaron máquinas virtuales conectadas a través de una red virtual interna que simulan comunicación cliente-servidor, a continuación, se detalla la configuración de cada máquina virtual:

Sistema anfitrión:

- Sistema operativo: Windows 11 Pro
- Software de virtualización: Oracle Virtual Box Versión 7.1.8 r168469 (Qt6.5.3)
- Hardware: Procesador Intel i5 14400 y 16GB RAM

Máquina virtual Alice (Cliente):

- Sistema operativo: Xubuntu 22.04 LTS
- Recursos asignados: 3 vCPUs y 4 GB RAM
- Adaptadores de red:
    - Adaptador 1: Red interna con IP estática: 192.168.100.10/24
    - Adaptador 2: Tipo NAT con DHCP

Máquina virtual Bob (Servidor):

- Sistema operativo: Ubuntu Server 22.04 LTS
- Recursos asignados: 3 vCPUs y 4 GB RAM
- Adaptadores de red:
    - Adaptador 1: Red interna con IP estática: 192.168.100.11/24
    - Adaptador 2: Tipo NAT con DHCP

Se confirmó tanto la conexión entre las dos máquinas virtuales como la conexión externa de cada una de ellas. Finalmente se tomó un *snapshot* (copia del estado actual de la máquina virtual) de cada máquina virtual para usarlo como un punto de restauración.

3.2 Herramientas y librerías criptográficas utilizadas

Las herramientas utilizadas en este entorno de laboratorio para el desarrollo de las pruebas y la medición del rendimiento de cada algoritmo criptográfico fueron tomadas en cuenta con base a su reconocimiento en la comunidad de criptografía post cuántica (PQC) para garantizar la validez y replicabilidad de los resultados.

Como paso inicial a ambas máquinas virtuales (Ubuntu Server y Xubuntu) se les instaló las dependencias de compilación esenciales directamente de los repositorios oficiales de Ubuntu (vía apt) y del código fuente en GitHub (vía git) con el siguiente comando en la terminal:

```
apt install build-essential git cmake ninja-build python3-venv python3-pip libssl-dev -y
```

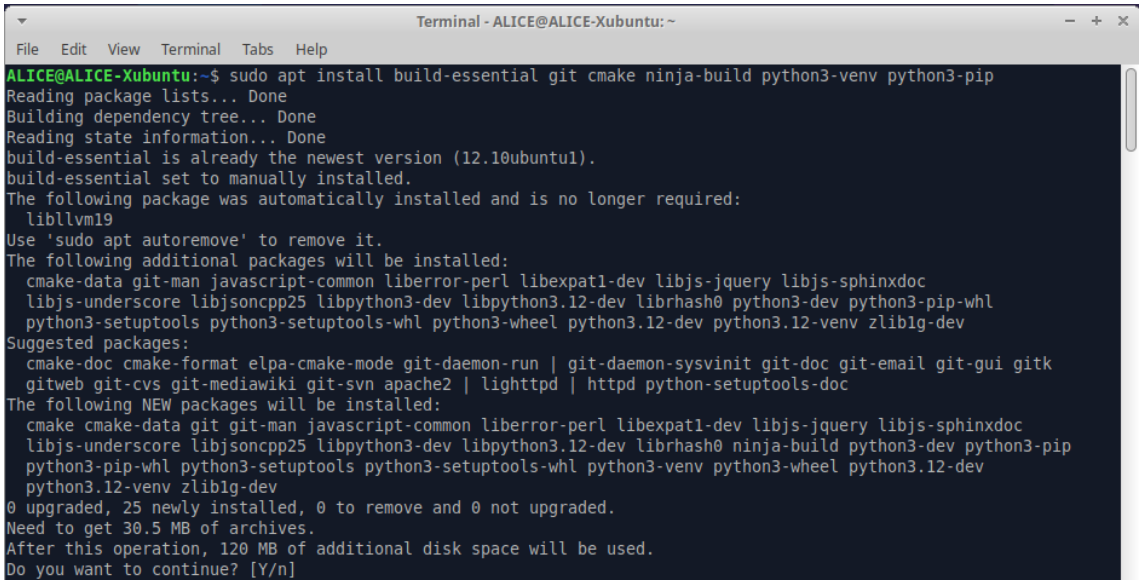
El propósito de cada componente es el siguiente:

- build-essential: Paquete que provee las herramientas de compilación de C/C++ (gcc, g++) y la utilidad make. Es indispensable para traducir el código fuente de liboqs y OpenSSL (escritos en C) a código máquina ejecutable.
- git: Sistema de control de versiones distribuido, utilizado para clonar (git clone) el código fuente más reciente de los repositorios de liboqs y OpenSSL desde GitHub.
- cmake: Generador del sistema de compilación. Inspecciona el sistema local y lee los archivos CMakeLists.txt del proyecto liboqs para generar los archivos de compilación nativos (build.ninja).
- ninja-build: Sistema de compilación de alto rendimiento. Es la herramienta que lee los archivos generados por cmake y ejecuta las tareas de compilación de forma eficiente y paralela.
- python3-venv: Módulo para la creación de entornos virtuales de Python. Se utiliza para crear un entorno aislado y evitar conflictos de dependencias con el sistema operativo.
- python3-pip: Instalador de paquetes de Python. Se utiliza para instalar librerías como oqs-python dentro del entorno virtual.

Este comando se ejecutó en ambas máquinas virtuales

Ilustración 1

Instalación del entorno base para PQC



El siguiente paso es instalar la librería liboqs que es el núcleo criptográfico del laboratorio. Esta librería es una implementación de código abierto en lenguaje C mantenida por el proyecto Open Quantum Safe y es usada en este laboratorio por ser la implementación de referencia utilizada por el NIST durante el proceso de estandarización.

Para asegurar el uso de la versión más reciente la librería se compiló desde la fuente como se detalla a continuación:

- git clone --depth 1 https://github.com/open-quantum-safe/liboqs.git
- cd liboqs
- mkdir build && cd build
- cmake -GNinja ..
- ninja
- sudo ninja install

Ilustración 2

Configuración de GNinja con Cmake

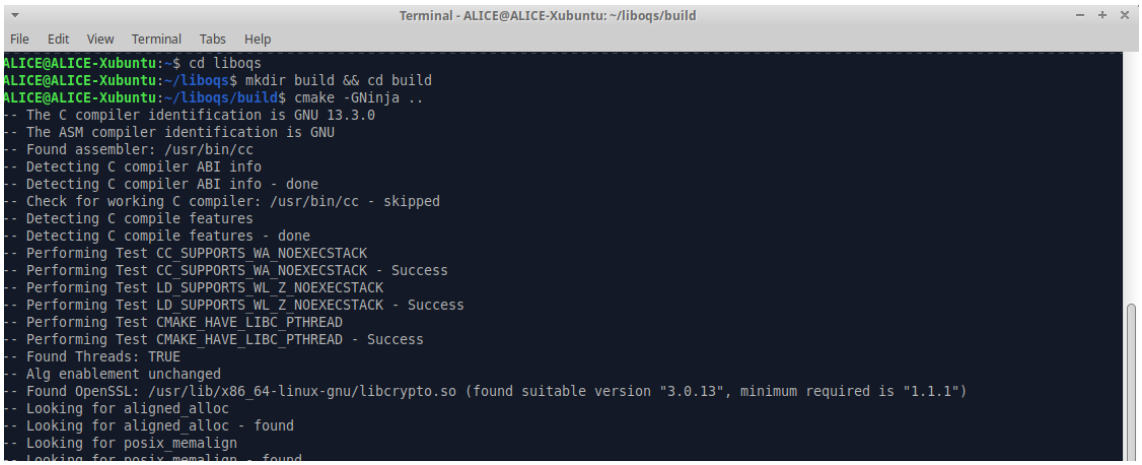
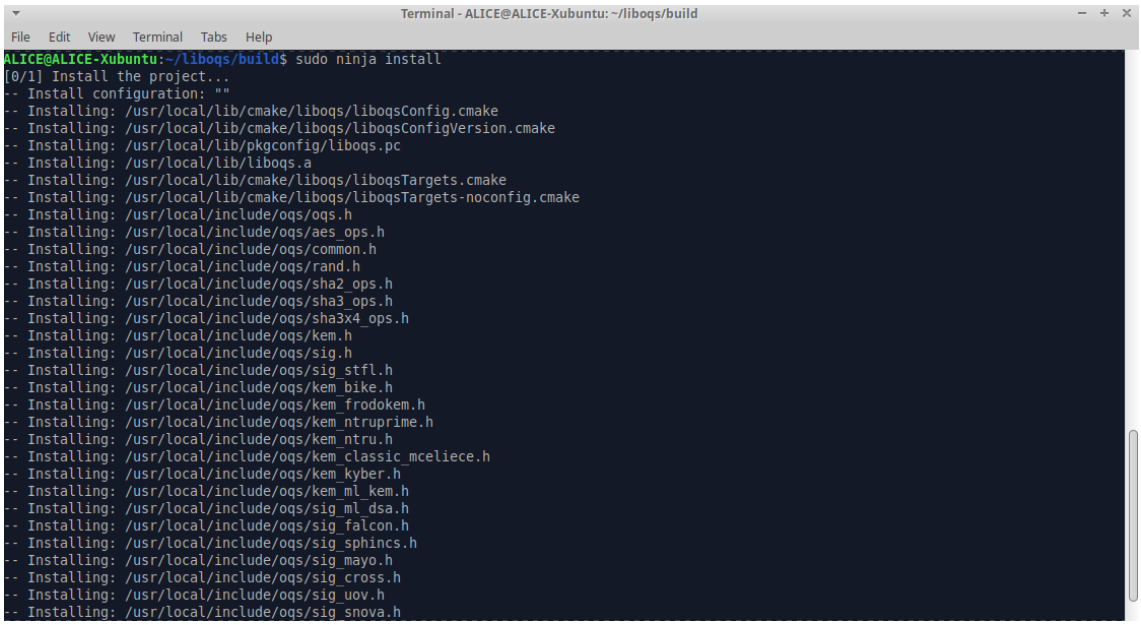


Ilustración 3

Instalación de ninja dentro del entorno



Al finalizar este proceso, la librería liboqs.so queda disponible en el sistema (/usr/local/lib), lista para ser utilizada por otras herramientas como los *wrappers* de Python o la versión modificada de OpenSSL.

Como último paso, se configuro un entorno virtual de Python en el directorio de trabajo de cada máquina virtual Para la creación de los scripts de prueba (descritos en la sección 3.3), se utilizó el lenguaje Python 3. La interacción entre los scripts (Python) y la librería liboqs (escrita en C) se logra a través del wrapper oficial oqs-python. Asimismo, para capturar, procesar y visualizar los datos de rendimiento generados por las pruebas, se instalaron las librerías estándar del ecosistema de ciencia de datos de Python. Estas se instalaron dentro del mismo entorno virtual (venv) para mantener la cohesión del proyecto.

El comando de instalación agrupó el *wrapper* `oqs` junto con las herramientas de análisis:

- cd ~
- python3 -m venv proyectopqc
- source proyectopqc/bin/activate
- git clone --depth 1 https://github.com/open-quantum-safe/liboqs-python.git
- cd liboqs-python
- pip install .
- pip install pandas cryptography matplotlib seaborn
    - **pandas**: Herramienta fundamental para el análisis de datos. Se utiliza para cargar los resultados de las pruebas (guardados en formato CSV) en *DataFrames* para su fácil manipulación y cálculo estadístico (promedio, mediana, etc.).
    - **matplotlib** y **seaborn**: Librerías de visualización. Se utilizan para generar los gráficos comparativos (diagramas de barras, gráficos de líneas) que se presentarán en el Capítulo 4, permitiendo una demostración visual de las diferencias de rendimiento y la complejidad algorítmica.

Ilustración 4

Instalación de PQC en el entorno

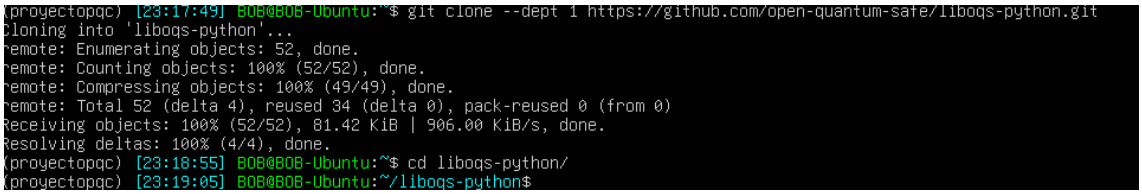
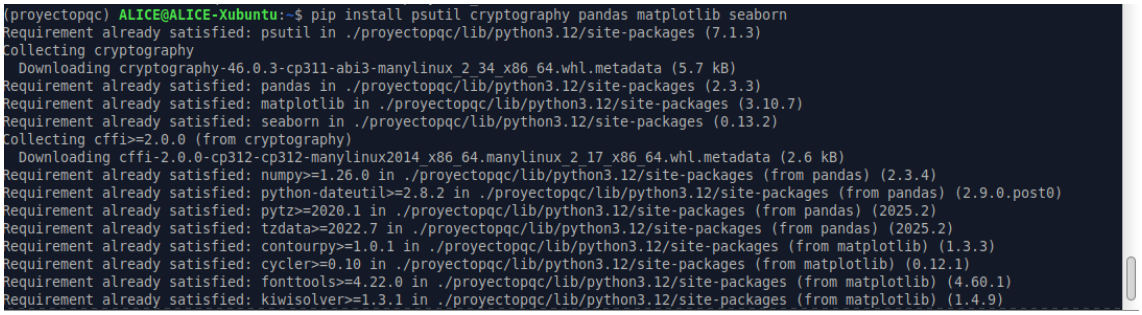


Ilustración 5

Instalación de librerías necesarias



3.3 Escenarios de evaluación

Basado en los escenarios definidos previamente en el alcance, se diseñaron dos conjuntos de scripts en Python que ejecutan los escenarios fundamentales de una comunicación segura simulando el *handshake* en una estructura de comunicación cliente-servidor. Estos scripts están diseñados de forma modular para ejecutar un proceso de prueba idéntico en todos los algoritmos de este estudio, permitiendo una comparación directa de su rendimiento.

3.3.1 Escenario KEM

Este escenario mide el rendimiento del *handshake* o establecimiento de la sesión. Se diseñó un script con el nombre test_kem.py el cual ejecuta, mide y promedia sobre mil iteraciones las siguientes operaciones para cada uno de los algoritmos KEM:

- Algoritmo PQC: CRYSTALS-Kyber en sus niveles de seguridad definidos.
- Algoritmos clásicos: RSA para intercambio de claves y ECDH.

El flujo de prueba es el siguiente:

- Paso 1: El servidor (Bob) genera su par de claves con KeyGen (pública y secreta).
- Paso 2: El cliente (Alice) recibe la clave pública del servidor y ejecuta la operación de encapsulamiento (Encaps) para generar el texto cifrado y el secreto compartido.
- Paso 3: El servidor (Bob) recibe el texto cifrado y ejecuta la operación de desencapsulamiento (Decaps) para recuperar el secreto compartido.

Se estableció un ciclo de mil iteraciones por algoritmo. Esta magnitud muestral fue seleccionada para garantizar la estabilidad estadística de los resultados y mitigar el impacto de las fluctuaciones aleatorias (ruido) propias del planificador de procesos del sistema operativo.

3.3.2 Escenario DSA

Este escenario mide el rendimiento de la autenticación mediante firmas digitales con el script creado con el nombre test_dsa.py que mide las operaciones de firma y verificación para cada uno de los algoritmos DSA:

- Algoritmo PQC: CRYSTALS-Dilithium en sus niveles de seguridad definidos.
- Algoritmos clásicos: ECDSA.

El flujo de prueba es el siguiente:

- Paso 1: El servidor (Bob) genera su par de claves de firma con KeyGen (pública y secreta).

- Paso 2: El servidor genera la signatura digital de un archivo de muestra.
- Paso 3: Alice recibe la clave pública de verificación, el archivo original y la signatura digital correspondiente.
- Paso 4: Alice ejecuta la operación Verify para validar la autenticidad e integridad del archivo.

3.4 Métricas de evaluación

Para capturar los datos y su previo análisis de rendimiento en relación con cada escenario definido en la sección 3.3, se emplearon herramientas que permiten un análisis técnico profundo y alineado con los estándares de benchmarking criptográfico. Las métricas de evaluación seleccionadas son:

3.4.1 Costo computacional (Ciclos de CPU)

Para la obtención de una medida precisa del costo computacional de cada operación requerida para el análisis de escalabilidad y rendimiento práctico, se descartó el tiempo del reloj estándar. Esta métrica es inherentemente "ruidosa", ya que se ve fácilmente afectada por la carga del sistema operativo, procesos externos o interrupciones, lo cual resta fiabilidad al benchmarking.

Si bien el contador de ciclos de CPU es la métrica ideal, se constató que la función incorporada para la medición directa de ciclos no estaba disponible en la versión de la librería de Python utilizada. En su lugar, se adoptó la alternativa de más alta precisión disponible en el ecosistema de Python: la función time.perf_counter_ns().

Esta función provee un contador de tiempo monotónico con resolución a nivel de nanosegundos. Su diseño está optimizado para benchmarking y es la métrica más fidedigna para reflejar el trabajo computacional puro. Aunque mide tiempo y no ciclos, su precisión extrema es un proxy confiable para el costo computacional real de los algoritmos y garantiza la portabilidad de las pruebas.

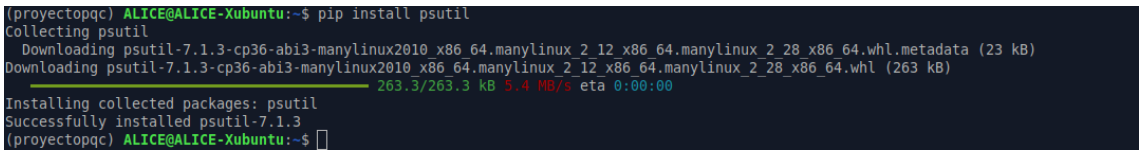
3.4.2 Huella de memoria (Uso de RAM)

Para medir el uso de la memoria RAM se utilizó la librería psutil que se la puede instalar con el siguiente comando:

pip install psutil

Ilustración 6

Instalación de Psutil para capturar el uso de RAM



Se tomaron las mediciones antes, durante y después de los escenarios de prueba para registrar el pico de memoria consumido por cada uno de los algoritmos.

3.4.3 Sobrecarga de canal (Tamaño de primitivas)

Esta métrica es esencial para el análisis porque determina la sobrecarga que cada algoritmo impone directamente al canal de comunicación. A diferencia de la latencia (costo de CPU) o la memoria (costo de RAM), esta métrica mide el costo del ancho de banda, que es un factor crítico en la implementación de PQC, especialmente en entornos con redes restringidas o dispositivos IoT.

Esta métrica es esencial para este análisis porque determina la sobrecarga que cada algoritmo impone directamente al canal de comunicación. A diferencia del costo de CPU o memoria RAM, esta métrica mide el costo del ancho de banda que es un factor crítico en la adopción de algoritmos post-cuánticos, especialmente en redes limitadas.

La sobrecarga se mide cuantificando el tamaño de bytes o primitivas criptográficas, que son bloques de datos que los algoritmos generan para que la comunicación segura funciones, las primitivas que se midieron son:

- Tamaño de clave pública: mide el costo del ancho de banda del handshake inicial.
- Tamaño de clave privada: mide el costo de almacenamiento en el servidor o cliente.
- Tamaño del texto cifrado en caso de KEM: mide el costo del ancho de banda del intercambio de claves.
- Tamaño de la signatura digital en caso de DSA: mide el costo de ancho de banda de autenticación de mensajes.

Para obtener estas métricas se utilizó la función nativa de Python len() aplicada directamente a variables de tipo byte que contienen dichas primitivas y este resultado proporciona una métrica directa del costo de transmisión para cada algoritmo.

3.5 Procedimiento de pruebas

Las pruebas fueron realizadas sistemáticamente de manera rigurosa con el objetivo principal de garantizar una base comparativa justa para los algoritmos post-cuánticos y algoritmos clásicos y asegurar la replicabilidad de los resultados.

Este proceso se ejecutó de la siguiente manera para cada nivel de seguridad:

- Inicio del servidor: Primero, se ejecutó el script de servidor en la máquina de Bob, el cual pone a la escucha a la máquina en la red interna del laboratorio, en este caso Bob tiene la IP 192.168.100.11:
  python3 test_kem.py --modo servidor
- Ejecución del cliente: Una vez iniciado el servidor, en la máquina virtual Alice se lanzaba el script especificando los parámetros concretos de la prueba. Esta es la parte que inicia la conexión, ejecuta el bucle de N iteraciones y registra las métricas. El comando especificaba todos los parámetros tal como se muestra en este ejemplo:
  python3 test_kem.py --modo cliente --algoritmo KYBER_768 --iteraciones 1000

- Recolección de métricas: El script cliente establecía conexión con el servidor y ejecutaba el escenario completo el número N de veces definido y mientras tanto se invocaron las herramientas de métricas para capturar los ciclos de la CPU y el uso de memoria RAM. Las primitivas criptográficas se midieron una vez por algoritmo ya que esto mide el ancho de banda y no necesita repetirse N veces.
- Almacenamiento de resultados y retroalimentación: Al finalizar las iteraciones el script del cliente almacena los datos dentro de un archivo CSV. Aquí se generan dos tipos de salidas:
    - Retroalimentación en consola: El script imprimía un indicador de progreso que imprimía un punto cada vez que se iteraba 10 veces mientras se ejecutaba para confirmar que el script no se haya bloqueado y al finalizar el bucle imprime la confirmación exitosa del proceso.
    - Almacenamiento en archivo CSV: El script utiliza la librería panda para guardar los resultados completos en un archivo CSV con un nombre que se genera automáticamente de esta manera: resultados_[tipo]_[algoritmo].csv. Las columnas del archivo son:
        - Algoritmo: el algoritmo usado en la prueba.
        - Iteración: el número de iteraciones (1 a N).
        - Ciclos_key_gen_ns: tiempo necesario para generar la clave.
        - Ciclos_operacion_1_ns: ciclos de CPU para la operación principal del cliente (Encaps o Sign).
        - Ciclos_operacion_2_ns: ciclos de CPU para la operación principal del servidor (Decaps o Verify).
        - Ram_pico_bytes: pico de memoria RAM detectada durante la prueba.
        - Tamano_pk_bytes: tamaño de la clave pública en bytes (primitiva).
        - Tamano_sk_bytes: tamaño de la clave secreta en bytes (primitiva).
        - Tamano_text_bytes: tamaño del texto o firma cifrado (primitiva).

CAPÍTULO IV: PRUEBAS Y ANÁLISIS DE RESULTADOS

4.1 Resultados obtenidos

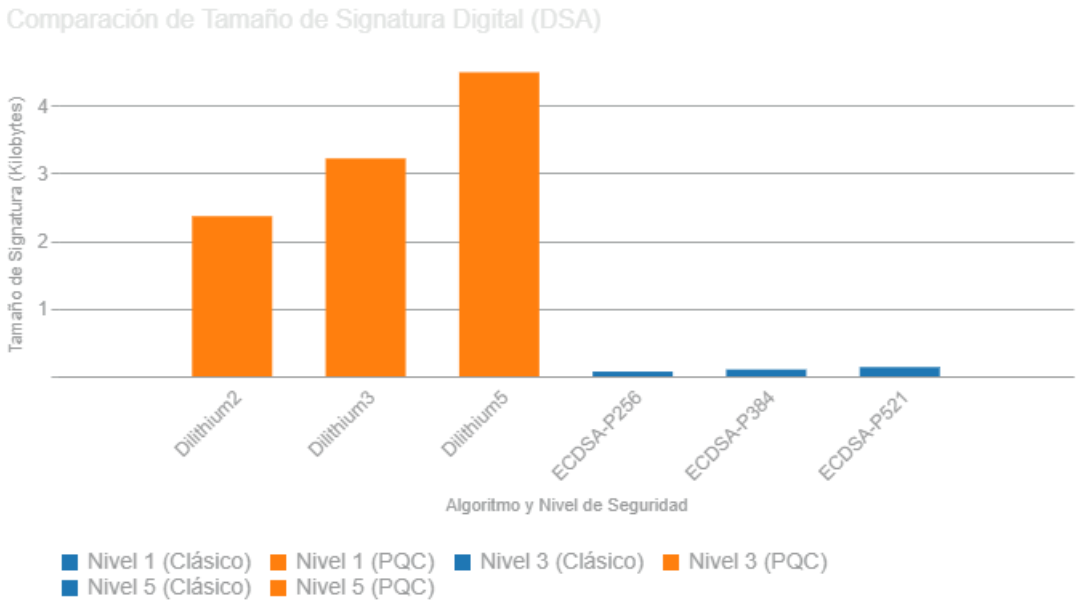
El presente capítulo expone y examina detalladamente los datos recabados durante la ejecución del proceso de negociación o handshake gestionado por el script al interactuar con diversos algoritmos criptográficos tanto del espectro clásico como del post-cuántico, dividiendo el análisis técnico en dos segmentos diferenciados para abordar el escenario de Algoritmos de Firma Digital y el escenario de Mecanismos de Encapsulamiento de Claves mientras se contrastan las métricas fundamentales de rendimiento que fueron establecidas y definidas con anterioridad en el tercer capítulo de esta investigación.

4.1.1 Resultados obtenidos del escenario DSA

La fase inicial de la evaluación se centró en los Algoritmos de Firma Digital responsables de garantizar la autenticación e integridad mediante la ejecución del script sobre el estándar post-cuántico Dilithium y el protocolo clásico ECDSA reservando la comparativa con RSA-Sign para una etapa posterior del estudio, sometiendo a cada algoritmo a un ciclo de mil iteraciones para cuantificar con precisión el coste computacional asociado a las operaciones de generación de claves, firma y verificación junto con la sobrecarga inherente a las primitivas.

Figura 1

Comparación de tamaño de signatura digital



El análisis empírico concerniente a los algoritmos DSA inicia abordando la métrica de sobrecarga del canal representada por el tamaño de la firma que constituye la principal desventaja en la migración hacia la Criptografía Post-Cuántica tal como se evidencia en la Figura 1 donde se contrasta directamente a Dilithium frente a ECDSA en los tres niveles de seguridad del NIST, confirmando gráficamente que el incremento en la robustez de seguridad conlleva un aumento consecuente en el consumo de ancho de banda dado que en el nivel 1 la firma de Dilithium2 con 2.36 KB supera en treinta y cuatro veces el tamaño de su contraparte ECDSA-P256 de 0.069 KB tal como se detalla en los valores exactos de la **Tabla 1**.

Esta diferencia se mantiene constante a través de todos los niveles evaluados observándose que la firma de Dilithium5 supera a ECDSA-P521 en una proporción análoga, lo cual demuestra empíricamente que si bien Dilithium escala de manera casi lineal al igual que ECDSA, opera bajo un factor de sobrecarga masivamente superior que plantea consideraciones críticas para el diseño de arquitecturas en infraestructuras sensibles al ancho de banda así como para la implementación en dispositivos limitados dentro del ecosistema de Internet de las Cosas.

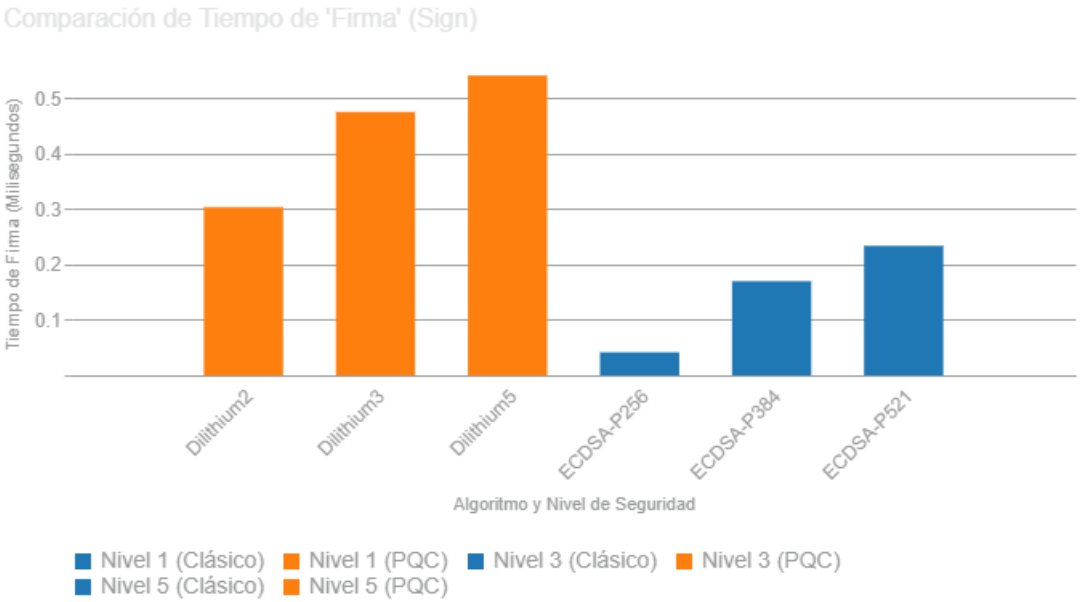
Tabla 1

Datos específicos para la **Figura 1**

Algoritmo	Nivel NIST	Tamaño Signatura (KB)
Dilithium2	Nivel 1	2.3633
Dilithium3	Nivel 3	3.2158
Dilithium5	Nivel 5	4.4873
ECDSA-P256	Nivel 1	0.0693
ECDSA-P384	Nivel 3	0.1006
ECDSA-P521	Nivel 5	0.1353

Figura 2

Comparación de tiempo de firma



La **Figura 2** compara el costo computacional de la operación de firma. Los resultados revelan que el algoritmo clásico ECDSA es significativamente más eficiente que Dilithium en todos los niveles de seguridad. Específicamente, en el nivel 1, Dilithium2 registró un tiempo de 0.3032 ms, resultando 7.32 veces más lento que ECDSA-P256 (0.0414 ms), cuyos detalles se observan en la Tabla 2.

Esta brecha en el rendimiento operativo persiste al escalar la seguridad del sistema aunque se observa una reducción en la disparidad hasta un factor de 2.3 veces al llegar al nivel 5 lo cual sugiere que la implementación de Dilithium impondrá mayores exigencias de procesamiento sobre los servidores o clientes encargados de la generación de firmas para mantener el caudal transaccional, no obstante, esta latencia adicional en la fase de generación encuentra su balance técnico en la eficiencia del proceso de verificación cuyo análisis detallado se abordará en la siguiente métrica.

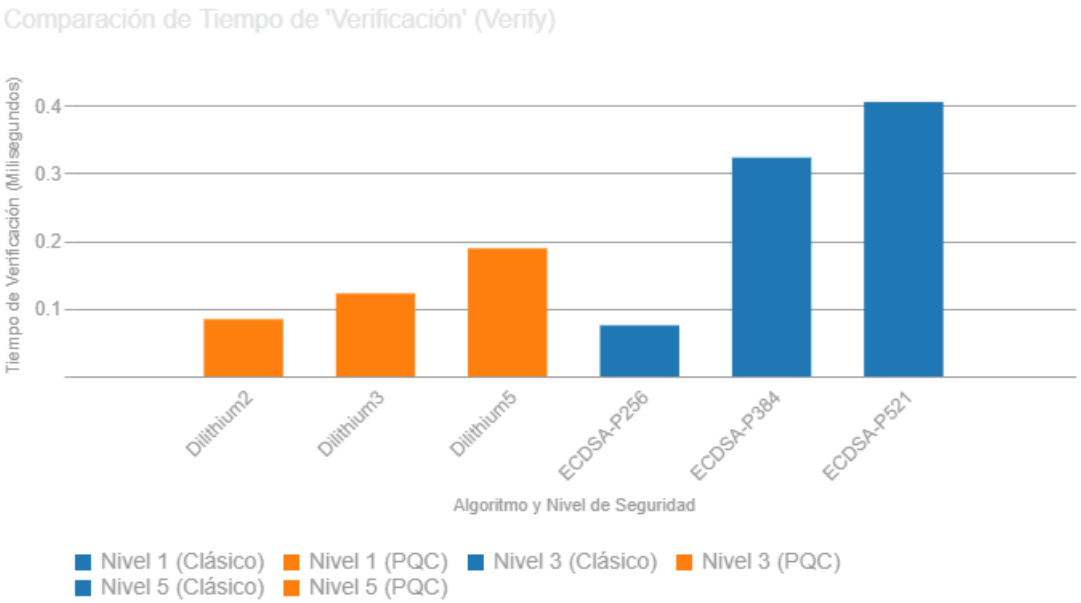
Tabla 2

Tiempo de Firmas para **Figura 2**

Algoritmo	Nivel NIST	Tiempo Firma (ms)
Dilithium2	Nivel 1	0.3032
Dilithium3	Nivel 3	0.4752
Dilithium5	Nivel 5	0.5409
ECDSA-P256	Nivel 1	0.0414
ECDSA-P384	Nivel 3	0.1697
ECDSA-P521	Nivel 5	0.2336

Figura 3

Comparación de tiempo de verificación



La **Figura 3** detalla el comportamiento del costo computacional asociado a la operación de verificación la cual resulta determinante para el rendimiento percibido del cliente demostrando a través de los datos empíricos que el algoritmo Dilithium se encuentra altamente optimizado para estos procesos particularmente en los estratos de seguridad más elevados correspondientes al nivel 5, pues si bien en el nivel 1 se observa una equivalencia técnica donde Dilithium2 y ECDSA-P256 presentan tiempos de ejecución casi idénticos. No obstante, esta tendencia se rompe drásticamente a medida que se incrementan los requisitos de robustez dado que el costo operativo de ECDSA se dispara desproporcionadamente llegando al extremo en el que la verificación con ECDSA-P521 resulta ser 2.14 veces más lenta que con Dilithium5, ventaja estratégica que evidencia una filosofía de diseño orientada a liberar de carga computacional al verificador o cliente para concentrarla en el firmante o servidor siendo esta configuración idónea para despliegues masivos donde los dispositivos receptores poseen capacidades de hardware restringidas. Los datos exactos se encuentran en la **Tabla 3**.

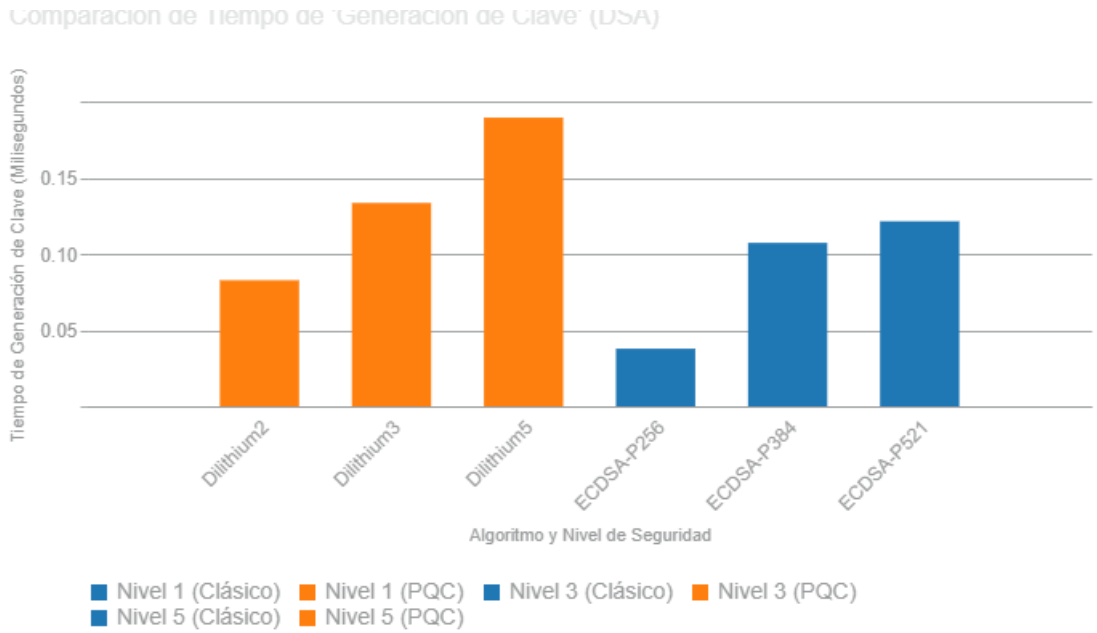
Tabla 3

Datos Específicos para **Figura 3**

Algoritmo	Nivel NIST	Tiempo Verificación (ms)
Dilithium2	Nivel 1	0.0843
Dilithium3	Nivel 3	0.1223
Dilithium5	Nivel 5	0.1889
ECDSA-P256	Nivel 1	0.0751
ECDSA-P384	Nivel 3	0.3229
ECDSA-P521	Nivel 5	0.4049

Figura 4

Comparación de tiempo de generación de clave



Finalmente, se aborda el análisis de la etapa de Generación de Claves representado en la **Figura 4** la cual expone el costo temporal que conlleva la creación del par criptográfico público y privado para cada algoritmo evaluado evidenciando que si bien ambos esquemas operan en fracciones reducidas de milisegundos el estándar clásico ECDSA mantiene una ligera ventaja de eficiencia sobre la propuesta post-cuántica Dilithium a través de todo el espectro de seguridad analizado.

Al examinar el nivel 1 se aprecia que ECDSA-P256 completa la operación en aproximadamente 0.04 ms mientras que Dilithium2 requiere cerca del doble de tiempo situándose alrededor de los 0.08 ms marcando una diferencia de rendimiento que se atribuye a la complejidad matricial de los retículos frente a la aritmética de curvas elípticas, brecha que se expande ligeramente en el nivel 5 donde Dilithium5 registra el mayor costo computacional de la prueba rozando los 0.19 ms en contraste con los 0.12 ms de ECDSA-P521, tal como se detalla en la **Tabla 4**. No obstante, es importante destacar que, aunque existe una degradación del rendimiento en el ámbito post-cuántico, esta operación se ejecuta con mucha menor frecuencia que la firma o verificación por lo que el impacto global en la latencia del servicio resulta marginal en entornos de producción real.

Tabla 4

Datos Específicos para **Figura 4**

Algoritmo	Nivel NIST	Tiempo KeyGen (ms)
Dilithium2	Nivel 1	0.0827
Dilithium3	Nivel 3	0.1334
Dilithium5	Nivel 5	0.1894
ECDSA-P256	Nivel 1	0.0378
ECDSA-P384	Nivel 3	0.1072
ECDSA-P521	Nivel 5	0.1214

4.1.1.1 Análisis comparativo

El análisis integral de los datos recopilados para el escenario de firma digital revela un panorama técnico de gran complejidad donde la métrica más preponderante resulta ser la sobrecarga del canal cuantificada mediante el tamaño de la firma en kilobytes, lo cual evidencia que la implementación de seguridad PQC demanda una infraestructura de ancho de banda significativamente mayor dado que en el nivel 1 la firma de Dilithium2 supera en treinta y cuatro veces a la de ECDSA-P256 estableciendo una disparidad constante, comportamiento que confirma empíricamente que el protocolo Dilithium opera bajo un factor de sobrecarga estructural masivamente superior al de sus contrapartes clásicas independientemente de la robustez criptográfica seleccionada.

En lo referente al costo computacional, se observa una clara dicotomía operativa donde la eficiencia del proceso de firma favorece inequívocamente al algoritmo clásico pues ECDSA-P256 resulta ser 7.3 veces más veloz que Dilithium2 manteniendo esta brecha de rendimiento favorable incluso en el nivel 5 donde aún conserva una ventaja de 2.3 veces, sin embargo esta relación de desempeño se invierte drásticamente durante la operación de verificación dado que si bien en el nivel 1 ambos algoritmos presentan tiempos idénticos de 0.08 ms el escalado de seguridad penaliza a ECDSA mientras Dilithium optimiza su respuesta logrando que en el nivel 5 sea 2.1 veces más rápido que ECDSA-P521, situación que contrasta con la generación de claves donde, aunque el algoritmo clásico es consistentemente superior la ejecución en el rango de microsegundos, permite considerar dicha diferencia como despreciable para efectos prácticos de implementación en entornos reales.

4.1.1.2 Discusión técnica

La adopción del estándar Dilithium trasciende la categoría de una simple actualización de protocolos para constituirse como una nueva arquitectura fundamental del perfil de rendimiento del sistema identificando un intercambio técnico o trade-off evidente, donde el costo operativo de la migración se manifiesta en una doble vertiente que inicia con una demanda ineludible de ancho de banda producto de firmas treinta veces más voluminosas que impactan directamente a redes con restricciones severas como las del Internet de las Cosas y continúa con un incremento sustancial en el costo de procesamiento para el firmante dado que la operación de generación de firma experimenta una latencia entre dos y siete veces superior a los estándares actuales.

Como contrapartida a estos costos la migración ofrece una ganancia inesperada en la eficiencia de los procesos de verificación que complementa la garantía de seguridad post-cuántica sugiriendo un diseño algorítmico deliberado que optimiza la validación a expensas de la generación y traslada estratégicamente la carga computacional desde el receptor o cliente de recursos limitados hacia el emisor o servidor de alta capacidad, arquitectura que resulta ideal para escenarios de distribución masiva de uno a muchos tales como la firma de software o la autenticación de transmisiones donde un activo digital se firma una única vez pero debe ser verificado millones de veces por diversos usuarios finales. Estos resultados experimentales son consistentes con la teoría de retículos descrita por los autores de CRYSTALS-Dilithium, quienes predicen un aumento lineal en el tamaño de la firma a cambio de seguridad post-cuántica. A diferencia de ECDSA, cuya seguridad se basa en el problema del logaritmo discreto y permite claves muy compactas, la estructura algebraica de Dilithium impide tal compresión. Los datos obtenidos corroboran que no existe una 'anomalía' en la implementación, sino que el costo de ancho de banda es una característica intrínseca y matemática del algoritmo, tal como se anticipaba en el marco teórico.

4.1.1.3 Análisis estadístico y fiabilidad de medición

La **Tabla 5** presenta las estadísticas descriptivas para los algoritmos de Signatura Digital.

Tabla 5

Análisis Estadístico de Rendimiento (DSA)

Algoritmo	Operación	Media (ms)	Desviación Estándar (ms)	Coeficiente de Variación (CV %)
Dilithium2	KeyGen	0.0827	0.0957	115.71%
	Sign	0.3032	0.2072	68.33%
	Verify	0.0843	0.0532	63.06%
Dilithium5	KeyGen	0.1894	0.0623	**32.90%**
	Sign	0.5409	0.3291	60.84%
	Verify	0.1889	0.0594	**31.44%**

Algoritmo	Operación	Media (ms)	Desviación Estándar (ms)	Coeficiente de Variación (CV %)
ECDSA-P256	KeyGen	0.0378	0.5399	**1429.31%**
	Sign	0.0414	0.0480	115.94%
	Verify	0.0751	0.0377	50.16%

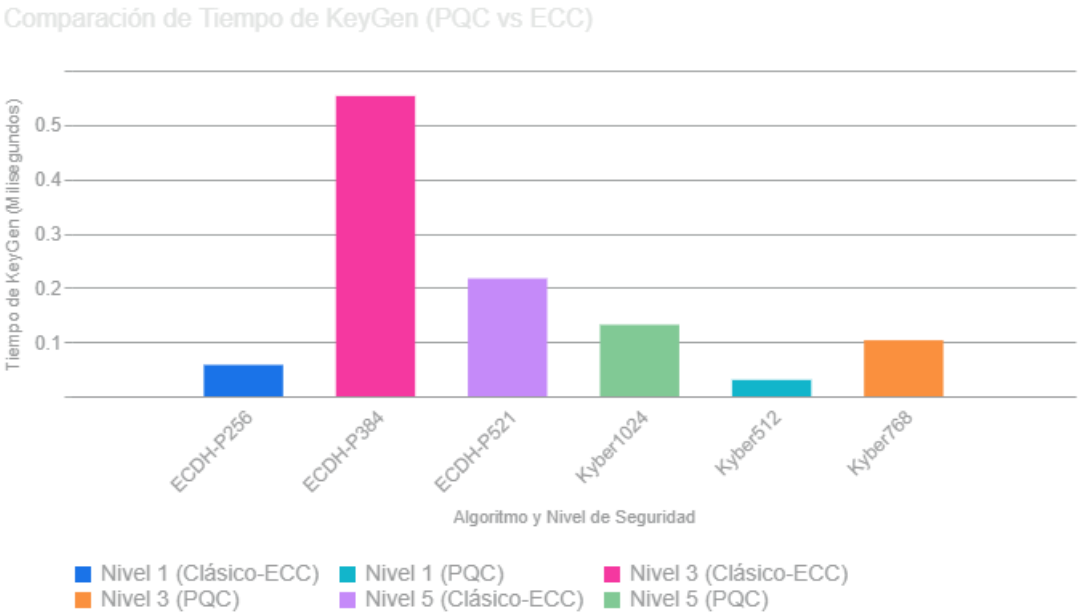
La Tabla 5 comprende las estadísticas descriptivas aplicadas a los algoritmos de firma digital permitiendo una evaluación que trasciende el simple promedio aritmético para adentrarse en la estabilidad operativa mediante el análisis de la desviación estándar y el coeficiente de variación, métricas que revelan comportamientos críticos en la consistencia de los tiempos de ejecución particularmente en el caso del algoritmo clásico ECDSA-P256 el cual a pesar de ostentar el promedio más bajo en la generación de claves con 0.0378 ms presenta una inestabilidad extrema reflejada en un coeficiente de variación del 1429.31% lo que sugiere la presencia de latencias esporádicas significativas o valores atípicos que distorsionan la predictibilidad del sistema en entornos de alta demanda.

En contraste, con la imprevisibilidad observada en el escenario clásico las variantes post-cuánticas muestran comportamientos dispares según el nivel de seguridad implementado observándose que Dilithium5 ofrece una consistencia operativa superior a Dilithium2 tal como lo evidencian sus coeficientes de variación en las fases de generación de claves y verificación que se mantienen cercanos al 30% frente a los valores superiores al 60% o 100% de la versión más ligera, hallazgo que permite inferir que la mayor complejidad matemática y el tamaño de los parámetros en el nivel 5 actúan paradójicamente como un factor estabilizador amortiguando las fluctuaciones relativas y ofreciendo a los arquitectos de sistemas un perfil de latencia más predecible a pesar del mayor costo computacional absoluto.

4.1.2 Resultados obtenidos del escenario KEM

Figura 5

Comparación de tiempo de KeyGen



La evaluación del mecanismo de encapsulamiento de claves inicia con el análisis de la **Figura 5** que ilustra claramente la disparidad existente en la etapa de generación de claves entre las distintas familias de algoritmos siendo esta una métrica fundamental debido a que dicho proceso representa el cuello de botella más significativo en la adopción de estándares tradicionales como RSA para la gestión de claves de sesión efímeras tal como se evidencia al observar que el algoritmo RSA-3072 requirió un promedio de ejecución de 254.88 milisegundos. Los datos detallados se encuentran en la **Tabla 6**.

Al realizar la comparativa directa con el algoritmo más eficiente dentro del mismo nivel de seguridad denominado Kyber512 que registró apenas 0.0301 ms se demuestra matemáticamente que el estándar RSA-3072 resulta ser 8,462 veces más lento en la generación de claves lo cual constituye la prueba empírica de que la complejidad algorítmica superpolinomial inherente a RSA lo convierte en una tecnología obsoleta para esta aplicación específica independientemente de la amenaza cuántica futura mientras que las alternativas modernas como Kyber y ECDH presentan tiempos ultrarrápidos escalando de manera controlada y eficiente.

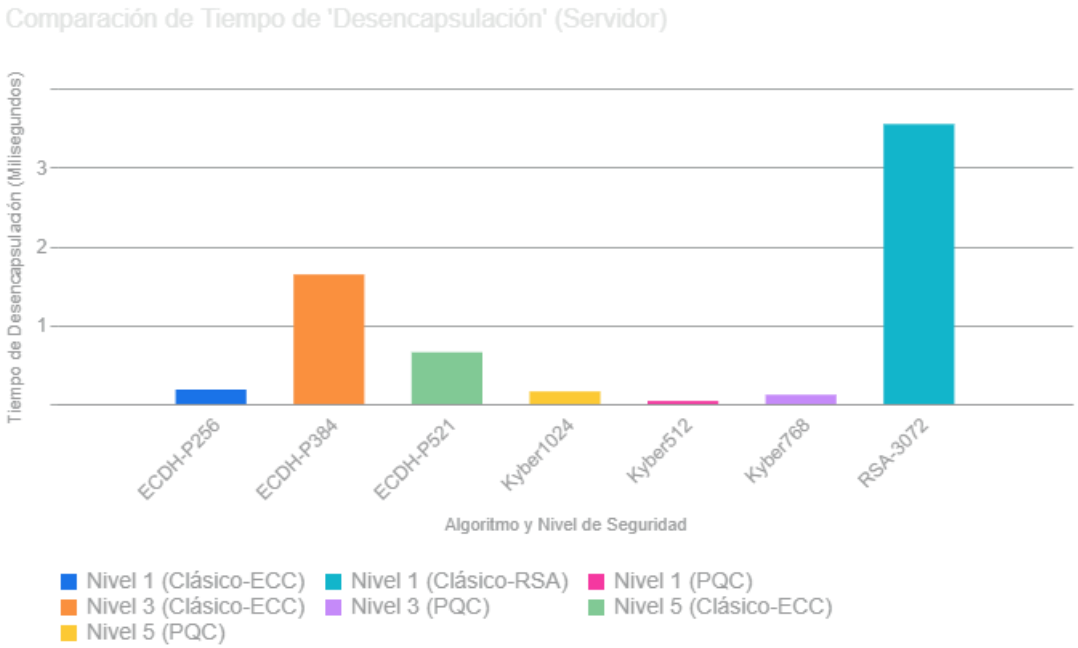
Tabla 6

Datos Específicos para **Figura 5**

Algoritmo	Nivel NIST	Tiempo KeyGen (ms)
ECDH-P256	Nivel 1	0.0577
ECDH-P384	Nivel 3	0.5530
ECDH-P521	Nivel 5	0.2169
Kyber1024	Nivel 5	0.1316
Kyber512	Nivel 1	0.0301
Kyber768	Nivel 3	0.1032
RSA-3072	Nivel 1	254.8853

Figura 6

Comparación de tiempo de desencapsulación



La **Figura 6** detalla el comportamiento temporal de la fase de desencapsulación la cual constituye una operación crítica para el servidor encargada de la recuperación del secreto compartido demostrando a través de los datos corregidos que el algoritmo Kyber se posiciona como la alternativa más eficiente en esta etapa superando con holgura tanto a la aritmética de RSA como a la de ECDH, supremacía que se hace evidente en el nivel 1 donde Kyber512 finaliza el proceso en 0.0377 ms resultando ser 4.84 veces más rápido que ECDH-P256 con sus 0.1827 ms y marcando una distancia abismal de 93.98 veces respecto a los 3.5463 ms requeridos por RSA-3072, tal como se detalla en la **Tabla 7**, hallazgo que tiene implicaciones directas para el diseño de servidores de alta demanda pues indica que la migración hacia Kyber tiene el potencial de liberar recursos sustanciales de procesamiento central permitiendo manejar una mayor concurrencia de conexiones por segundo a cambio de asumir el costo asociado al mayor consumo de ancho de banda.

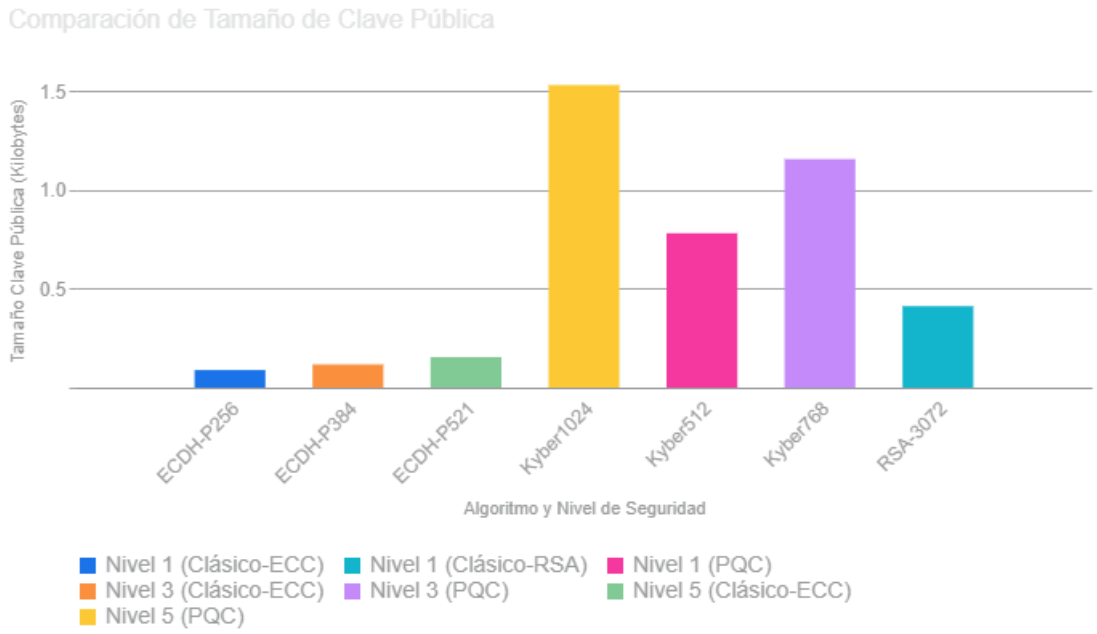
Tabla 7

Datos Específicos para **Figura 6**

Algoritmo	Nivel NIST	Tiempo Decaps (ms)
ECDH-P256	Nivel 1	0.1827
ECDH-P384	Nivel 3	1.6414
ECDH-P521	Nivel 5	0.6557
Kyber1024	Nivel 5	0.1590
Kyber512	Nivel 1	0.0377
Kyber768	Nivel 3	0.1170
RSA-3072	Nivel 1	3.5463

Figura 7

Comparación de tamaño de clave pública



La **Figura 7** aborda la dimensión crítica de la sobrecarga del canal estableciendo un contrapunto técnico respecto a las métricas temporales previamente analizadas donde Kyber demostró superioridad operativa, pues en este apartado se evidencia la desventaja estructural inherente a los algoritmos basados en retículos frente a la compacidad aritmética de las curvas elípticas tal como se manifiesta en el nivel 1 donde la clave pública de Kyber512 con un peso de 0.78 KB resulta ser 8.79 veces más voluminosa que su contraparte ECDH-P256 de apenas 0.09 KB. Los datos detallados se encuentran en la **Tabla 8**.

Si bien esta diferencia de magnitud es estadísticamente significativa resulta considerablemente inferior a la brecha observada en el escenario de firmas digitales donde la proporción alcanzaba un factor de treinta y cuatro veces, observándose además que la clave pública de Kyber es únicamente 1.9 veces mayor que la de RSA-3072 lo cual permite inferir que la transición desde sistemas heredados basados en factorización de enteros no conllevará un impacto tan drástico en el consumo de ancho de banda como el que experimentarían aquellas infraestructuras que migren desde entornos altamente optimizados con criptografía de curva elíptica.

Tabla 8

Datos Específicos para **Figura 7**

Algoritmo	Nivel NIST	Tamaño Clave Pública (KB)
ECDH-P256	Nivel 1	0.0889
ECDH-P384	Nivel 3	0.1172
ECDH-P521	Nivel 5	0.1543
Kyber1024	Nivel 5	1.5312
Kyber512	Nivel 1	0.7812
Kyber768	Nivel 3	1.1562
RSA-3072	Nivel 1	0.4121

4.1.2.1 Análisis comparativo

El análisis integral del mecanismo de encapsulamiento de claves revela una inversión fundamental en la jerarquía de rendimiento tradicional al demostrar que los algoritmos post-cuánticos basados en retículos logran superar la eficiencia computacional de los estándares vigentes eliminando efectivamente los cuellos de botella históricos asociados a la criptografía asimétrica robusta, pues los datos empíricos confirman que Kyber no solo reduce los tiempos de generación de claves en cuatro órdenes de magnitud respecto a RSA-3072 volviendo viable la rotación constante de claves efímeras, sino que también logra superar el rendimiento de las curvas elípticas en la operación crítica de desencapsulación del lado del servidor, hallazgo que sugiere que la adopción de Kyber optimizará drásticamente la latencia del proceso de negociación en servidores de alta concurrencia permitiendo procesar un volumen masivo de conexiones por segundo sin la penalización de procesamiento central que históricamente limitaba a los sistemas criptográficos de alta seguridad.

Por otro lado, la evaluación de la sobrecarga del canal introduce el contrapeso técnico necesario en este estudio al evidenciar que la superioridad temporal de los retículos conlleva un costo espacial inevitable aunque moderado en comparación con lo observado en las firmas digitales, ya que si bien las claves públicas de Kyber presentan

un incremento de tamaño aproximado de nueve veces frente a la compacidad extrema de ECDH lo cual impactará en la fragmentación de paquetes en redes altamente restringidas, la comparación con los sistemas legados sitúa este aumento en una perspectiva manejable al ser la clave de Kyber menos del doble que la de RSA-3072, comportamiento que indica que la migración desde infraestructuras basadas en factorización de enteros será prácticamente transparente en términos de ancho de banda mientras que la transición desde entornos puramente ECC requerirá ajustes en la gestión de la unidad máxima de transmisión para acomodar el incremento en la carga útil criptográfica.

4.1.2.2 Discusión técnica

Los resultados obtenidos en el escenario de encapsulamiento de claves configuran un panorama técnico concluyente donde la discusión sobre la migración hacia estándares post-cuánticos como Kyber presenta una dicotomía fascinante al evidenciar inicialmente que la arquitectura de RSA resulta obsoleta para aplicaciones de intercambio de claves efímeras no solo por su vulnerabilidad teórica ante amenazas futuras sino por un rendimiento inaceptable en la generación de claves que supera los 250 milisegundos lo cual lo vuelve inviable para sesiones dinámicas, situación que reduce la decisión ingenieril a una comparativa directa entre ECDH y Kyber donde se manifiesta un intercambio técnico ineludible entre la eficiencia del cómputo y la economía del almacenamiento obligando a los arquitectos de sistemas a aceptar que el costo operativo de la migración desde las curvas elípticas se paga íntegramente con un incremento sustancial en la sobrecarga del canal de comunicación dado que la velocidad de procesamiento se obtiene a expensas de un mayor volumen de datos transmitidos.

La cuantificación precisa de este impacto revela que la clave pública de Kyber es 8.6 veces más grande mientras que el texto cifrado que contiene el secreto encapsulado aumenta considerablemente, lo que resulta en un establecimiento de sesión que demanda la transmisión aproximada de 1.5 kilobytes de datos criptográficos frente a los escasos 0.12 kilobytes requeridos por el estándar ECDH constituyendo así un desafío de red significativo que debe ser meticulosamente evaluado en entornos con ancho de banda restringido como las infraestructuras del Internet de las Cosas, no obstante, es imperativo destacar que el beneficio de esta transición tecnológica trasciende la obvia garantía de confidencialidad a largo plazo propia de la seguridad post-cuántica.

4.1.2.3 Análisis estadístico de rendimiento

La **Tabla 9** presenta las estadísticas descriptivas para los algoritmos de Encapsulación de Claves.

Tabla 9

Análisis Estadístico de Rendimiento (KEM)

Algoritmo	Operación	Media (ms)	Desviación Estándar (ms)	Coeficiente de Variación (CV %)
Kyber512	KeyGen	0.0301	0.0236	**78.37%**
	Encaps	0.0346	0.0125	**35.99%**
	Decaps	0.0377	0.0095	**25.27%**
ECDH-P256	KeyGen	0.0577	0.2578	**446.82%**
	Encaps	0.1876	0.1314	70.03%
	Decaps	0.1827	0.1145	62.70%
RSA-3072	KeyGen	254.8853	183.5388	72.00%
	Encaps	0.1470	0.0686	46.68%
	Decaps	3.5463	0.9435	26.60%

La **Tabla 9** expone las métricas estadísticas descriptivas para los algoritmos de Encapsulación de Claves permitiendo identificar comportamientos profundos en la estabilidad del procesamiento que no son visibles mediante el simple promedio aritmético, destacándose de manera alarmante la irregularidad operativa del algoritmo ECDH-P256 en su fase de generación de claves donde registra un coeficiente de variación desproporcionado del 446.82% lo cual indica una dispersión extrema en los tiempos de ejecución y sugiere la presencia de latencias impredecibles que podrían degradar la calidad de servicio en aplicaciones de tiempo real, comportamiento errático que si bien se modera durante las fases de encapsulamiento y desencapsulamiento manteniéndose en un rango de variabilidad entre el 60% y 70% sigue evidenciando una falta de consistencia temporal que contrasta desfavorablemente con la estabilidad de las nuevas propuestas algorítmicas.

Por su parte el análisis del algoritmo Kyber512 revela un perfil de rendimiento altamente optimizado para la gestión de cargas de trabajo en servidores pues a pesar de mostrar cierta volatilidad inicial en la generación de claves con un 78.37% logra estabilizarse notablemente en la operación más crítica para el receptor que es la desencapsulación alcanzando un coeficiente de variación del 25.27% el cual representa la métrica de estabilidad más sólida del conjunto, cifra que curiosamente resulta técnica y estadísticamente comparable con la estabilidad de la desencapsulación de RSA-3072 que registra un 26.60% aunque operando en escalas temporales abismalmente distintas, demostrando con ello que la transición a Kyber permite conservar la predictibilidad operativa de los sistemas heredados robustos pero ejecutándola a velocidades de microsegundos que son órdenes de magnitud superiores.

CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- Se concluye que existe una divergencia estructural entre la criptografía clásica y la post-cuántica. Mientras que RSA y ECC basan su seguridad en la complejidad aritmética (factorización y logaritmos discretos) permitiendo claves compactas, los algoritmos basados en retículos (CRYSTALS-Kyber y Dilithium) fundamentan su resistencia en problemas geométricos multidimensionales. Este diseño matemático garantiza la inmunidad ante el algoritmo de Shor, pero conlleva inevitablemente un incremento en el tamaño de las primitivas criptográficas, confirmando que la mayor demanda de ancho de banda es una propiedad intrínseca del nuevo paradigma de seguridad y no un defecto de diseño.
- Se determinó que la integración de algoritmos post-cuánticos en sistemas actuales es técnicamente viable mediante el uso de herramientas de código abierto como liboqs y OpenSSL. El despliegue experimental en un entorno Linux demostró que no se requiere hardware especializado (como procesadores cuánticos) para ejecutar estos nuevos estándares. Las librerías actuales permiten una transición fluida a nivel de software, aunque se evidenció que la configuración de los parámetros de seguridad debe ser meticulosa para evitar incompatibilidades en el establecimiento del "handshake" TLS.
- Las pruebas de rendimiento revelaron un comportamiento asimétrico entre los algoritmos. En el intercambio de claves, CRYSTALS-Kyber demostró ser drásticamente más veloz que RSA y competitivo frente a ECC, eliminando la latencia de procesamiento en el servidor. Por el contrario, en las firmas digitales, CRYSTALS-Dilithium presentó una inversión de prioridades: su generación de firmas es más lenta que la de ECDSA, pero su verificación es extremadamente rápida. Sin embargo, la métrica más crítica fue la sobrecarga del canal, donde el volumen de datos transmitidos aumentó de manera masiva, identificándose como el nuevo cuello de botella del sistema.

- Al contrastar los datos experimentales con los estándares clásicos, se concluye que la migración es operativamente viable para servidores de alta demanda, pero presenta un desafío para redes limitadas. El trade-off final es claro: se gana seguridad a largo plazo y velocidad de CPU, a cambio de sacrificar ancho de banda. Por lo tanto, la adopción de CRYSTALS-Kyber y Dilithium se recomienda prioritariamente para infraestructuras críticas con conexiones robustas, mientras que en entornos IoT o redes inestables, la migración requerirá estrategias adicionales de optimización de red para mitigar el impacto del tamaño de las claves.

5.2 Recomendaciones

- Dado que la presente investigación aisló el costo computacional dentro de un entorno de laboratorio controlado se recomienda extender el alcance del estudio implementando estos algoritmos en escenarios de red adversos para analizar el comportamiento de las primitivas voluminosas de Kyber y Dilithium frente a fenómenos como la fragmentación de paquetes IP y la pérdida de tramas en redes móviles o satelitales, resultando imperativo que un trabajo futuro cuantifique el impacto real en la tasa de éxito de la conexión y la latencia percibida por el usuario cuando el proceso de negociación excede el tamaño de la ventana de congestión TCP inicial lo cual constituye una limitación crítica que no puede ser abordada con precisión en pruebas confinadas a una red de área local.
- Considerando que los algoritmos PQC analizados garantizan la seguridad desde una perspectiva algorítmica pero no aseguran la integridad física del canal de transmisión se recomienda a las organizaciones que operan infraestructuras críticas explorar arquitecturas de seguridad en profundidad investigando la viabilidad técnica y económica de modelos híbridos que combinen PQC para la autenticación en la última milla con la Distribución Cuántica de Claves para el núcleo de la red, combinación sinérgica que permitiría mitigar simultáneamente la amenaza de la computación cuántica mediante algoritmos matemáticos y la amenaza de interceptación física bajo la modalidad de almacenar ahora para descifrar después mediante los principios de la física cuántica inherentes a QKD.

- Si bien este trabajo determinó que Kyber y Dilithium son eficientes en tiempo de CPU sobre arquitectura x86 esto no se traduce directamente en eficiencia energética para dispositivos embebidos donde existe una disyuntiva no resuelta entre el ahorro de energía de procesamiento por la velocidad de Kyber y el mayor gasto energético de transmisión debido al tamaño de sus claves, por lo que se recomienda realizar mediciones precisas de consumo en Joules implementando estos algoritmos en microcontroladores reales como ARM Cortex-M4 o RISC-V para determinar si el costo energético de la radiotransmisión de claves grandes anula el ahorro obtenido en el procesamiento rápido siendo este paso vital para validar la viabilidad real de PQC en dispositivos operados por baterías o sistemas de recolección de energía.

Bibliografía

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*, 563-591.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms* (Cuarta ed.). Cambridge: The MIT Press.

CRYSTALS. (23 de Diciembre de 2020). *CRYSTALS*. Obtenido de CRYSTALS: https://pq-crystals.org/kyber

CRYSTALS. (16 de Febrero de 2021). *CRYSTALS*. Obtenido de CRYSTALS: https://pq-crystals.org/dilithium/index.shtml

FORTINET. (29 de Octubre de 2025). *Comprender los algoritmos de Shor y Grover y su impacto en la Ciberseguridad*. Obtenido de FORTINET: https://www.fortinet.com/lat/resources/cyberglossary/shors-grovers-algorithms

FORTINET. (4 de Noviembre de 2025). *What Is Public Key Infrastructure (PKI)?* Obtenido de https://www.fortinet.com/uk/resources/cyberglossary/public-key-infrastructure

IBM. (9 de Noviembre de 2022). *IBM Unveils 400 Qubit-Plus Quantum Processor and Next-Generation IBM Quantum System Two*. Obtenido de https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two?mhsrc=ibmsearch_a&mhq=osprey

IBM. (8 de Agosto de 2024). *¿Qué es el cifrado asimétrico?* Obtenido de IBM: https://www.ibm.com/es-es/think/topics/asymmetric-encryption?mhsrc=ibmsearch_a&mhq=crystals%20-%20kyber

IBM. (2 de Octubre de 2025). *IBM*. Obtenido de IBM: https://www.ibm.com/es-es/topics/cryptography

Milanov, E. (3 de Junio de 2009). *The RSA Algorithm.* Obtenido de Mathematics and Physics Colloquium Series II: http://susanka.org/MathPhysics2/RSA_Algorithm_Yevgeny.pdf

NIST. (1 de Octubre de 2025). *Digital Signature Standard (DSS)*. Obtenido de https://csrc.nist.gov/pubs/fips/186-5/final

NIST. (1 de Octubre de 2025). *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Obtenido de https://csrc.nist.gov/pubs/fips/203/final

NIST. (19 de Mayo de 2025). *NIST*. Obtenido de IBM: https://www.nist.gov/cybersecurity/what-quantum-cryptography

Schneider, J., & Smalley, I. (1 de Diciembre de 2023). *¿Qué es la criptografía cuántica?* Obtenido de https://www.ibm.com/mx-es/think/topics/quantum-cryptography

Schneider, J., & Smalley, I. (4 de Octubre de 2025). *¿Qué es la computación cuántica?* Obtenido de IBM: https://www.ibm.com/es-es/think/topics/quantum-computing?mhsrc=ibmsearch_a&mhq=computacion%20cuantica

Schneiider, & Smalley, I. (28 de Febrero de 2024). *¿Qué es un cúbit?* Obtenido de https://www.ibm.com/mx-es/topics/qubit?mhsrc=ibmsearch_a&mhq=cubit

Yan, Y. (2019). The Overview of Elliptic Curve Cryptography. *Journal of Physics:Conference Series*, 2386.

Anexos

7.1 Código para server_dsa.py

```
import socket
import oqs
import json
import traceback
from cryptography.hazmat.primitives.asymmetric import ec, rsa

HOST = "0.0.0.0"
PORT = 5000

print(f"[+] Servidor DSA escuchando en {HOST}:{PORT}")

def procesar_dsa(alg):
    """Procesa firmas digitales PQC y clásicas."""
    try:
        # === PQC: CRYSTALS-Dilithium ===
        if "Dilithium" in alg:
            print(f"[DSA] Ejecutando {alg}")
            with oqs.Signature(alg) as signer:
                pk = signer.generate_keypair()
                mensaje = b"Mensaje de prueba para DSA"
                firma = signer.sign(mensaje)
                ok = signer.verify(mensaje, firma, pk)
                if ok:
                    return {"status": "ok", "alg": alg, "msg": "Firma Dilithium verificada
correctamente"}
                else:
                    return {"status": "error", "error": "Verificación PQC fallida"}

        # === RSA (clásico) ===
```

```
        elif "RSA" in alg:
            print(f"[DSA] Ejecutando {alg}")
            key_size = 3072 if "3072" in alg else 7680
            rsa.generate_private_key(public_exponent=65537, key_size=key_size)
            return {"status": "ok", "alg": alg, "msg": "Firma RSA generada y procesada"}

        # === ECDSA (clásico) ===
        elif "ECDSA" in alg:
            print(f"[DSA] Ejecutando {alg}")
            curva = {
                "ECDSA-P256": ec.SECP256R1(),
                "ECDSA-P384": ec.SECP384R1(),
                "ECDSA-P521": ec.SECP521R1(),
            }.get(alg)
            if not curva:
                return {"status": "error", "error": f"Curva no soportada: {alg}"}
            ec.generate_private_key(curva)
            return {"status": "ok", "alg": alg, "msg": "Firma ECDSA procesada"}

        # === No reconocido ===
        else:
            return {"status": "error", "error": f"Algoritmo no soportado: {alg}"}

    except Exception as e:
        return {"status": "error", "error": str(e)}


def start_server():
    """Servidor TCP que recibe el nombre del algoritmo en JSON y responde con resultado JSON."""
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
```

```
        s.listen(5)
        print(f"[+] Escuchando conexiones DSA en {HOST}:{PORT}")

        while True:
            conn, addr = s.accept()
            with conn:
                print(f"[+] Conexión establecida desde {addr}")
                try:
                    conn.settimeout(3)
                    data = conn.recv(4096)
                    if not data:
                        continue
                    msg = json.loads(data.decode())
                    alg = msg.get("alg", "")
                    result = procesar_dsa(alg)
                    conn.sendall(json.dumps(result).encode())
                except Exception as e:
                    print(f"[!] Error: {e}")
                    traceback.print_exc()
                    conn.sendall(json.dumps({"status": "error", "error": str(e)}).encode())
                finally:
                    conn.close()

if __name__ == "__main__":
    start_server()
```

7.2 Código para server_kem.py

```
#!/usr/bin/env python3
import socket
import oqs
import json
```

```
import threading
import base64
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import rsa, ec, padding

HOST = "192.168.100.11"
PORT = 5555

# --- RSA y ECC (OpenSSL via cryptography) ---
def generate_rsa_keypair():
    key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
    return key, key.public_key()

def generate_ecc_keypair():
    key = ec.generate_private_key(ec.SECP384R1())
    return key, key.public_key()

def rsa_decrypt(sk, ct):
    return sk.decrypt(ct,
padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),
algorithm=hashes.SHA256(), label=None))

def ecdh_shared_secret(sk, peer_pub_bytes):
    peer_pub = serialization.load_pem_public_key(peer_pub_bytes)
    shared = sk.exchange(ec.ECDH(), peer_pub)
    return shared

# --- PQC Key Encapsulation Mechanism ---
def pqc_kem_shared_secret(alg_name, pk_bytes, sk_bytes, ct_bytes):
    with oqs.KeyEncapsulation(alg_name) as kem:
        kem.import_secret_key(sk_bytes)
```

```
        ss = kem.decapsulate(ct_bytes)
        return ss

# --- PQC Signature verification ---
def pqc_sign_verify(alg_name, msg, pk_bytes, sig_bytes):
    with oqs.Signature(alg_name) as signer:
        return signer.verify(msg, sig_bytes, pk_bytes)

def handle_client(conn, addr):
    print(f"[+] Conexión establecida desde {addr}")
    try:
        data = conn.recv(4096).decode()
        req = json.loads(data)
        op = req["operation"]

        if op == "KEM":
            alg = req["algorithm"]
            print(f"[KEM] Ejecutando {alg}")

            if alg == "Kyber512":
                with oqs.KeyEncapsulation(alg) as kem:
                    pk, sk = kem.generate_keypair()
                    conn.sendall(json.dumps({"public_key":
base64.b64encode(pk).decode()}).encode())
                    ct = base64.b64decode(conn.recv(4096))
                    ss = kem.decapsulate(ct)
                    conn.sendall(base64.b64encode(ss))
                    print("[KEM] Kyber completado.")

            elif alg == "RSA":
                sk, pk = generate_rsa_keypair()
```

```
            pem_pk = pk.public_bytes(encoding=serialization.Encoding.PEM,
                                     format=serialization.PublicFormat.SubjectPublicKeyInfo)
            conn.sendall(pem_pk)
            ct = base64.b64decode(conn.recv(4096))
            ss = rsa_decrypt(sk, ct)
            conn.sendall(base64.b64encode(ss))
            print("[KEM] RSA completado.")

        elif alg == "ECC":
            sk, pk = generate_ecc_keypair()
            pem_pk = pk.public_bytes(encoding=serialization.Encoding.PEM,
                                     format=serialization.PublicFormat.SubjectPublicKeyInfo)
            conn.sendall(pem_pk)
            peer_ct = base64.b64decode(conn.recv(4096))
            ss = ecdh_shared_secret(sk, peer_ct)
            conn.sendall(base64.b64encode(ss))
            print("[KEM] ECC completado.")

    elif op == "DSA":
        alg = req["algorithm"]
        print(f"[DSA] Ejecutando {alg}")

        msg = req["message"].encode()

        if alg == "Dilithium2":
            with oqs.Signature(alg) as signer:
                pk, sk = signer.generate_keypair()
                sig = signer.sign(msg, sk)
                conn.sendall(json.dumps({
                    "public_key": base64.b64encode(pk).decode(),
```

```
                "signature": base64.b64encode(sig).decode()
            }).encode())
            print("[DSA] Dilithium completado.")

        elif alg == "RSA":
            sk, pk = generate_rsa_keypair()
            sig = sk.sign(msg, padding.PSS(mgf=padding.MGF1(hashes.SHA256()),
                                           salt_length=padding.PSS.MAX_LENGTH),
                          hashes.SHA256())
            pem_pk = pk.public_bytes(encoding=serialization.Encoding.PEM,
                                     format=serialization.PublicFormat.SubjectPublicKeyInfo)
            conn.sendall(json.dumps({
                "public_key": pem_pk.decode(),
                "signature": base64.b64encode(sig).decode()
            }).encode())
            print("[DSA] RSA completado.")

        elif alg == "ECDSA":
            sk, pk = generate_ecc_keypair()
            sig = sk.sign(msg, ec.ECDSA(hashes.SHA256()))
            pem_pk = pk.public_bytes(encoding=serialization.Encoding.PEM,
                                     format=serialization.PublicFormat.SubjectPublicKeyInfo)
            conn.sendall(json.dumps({
                "public_key": pem_pk.decode(),
                "signature": base64.b64encode(sig).decode()
            }).encode())
            print("[DSA] ECDSA completado.")

    except Exception as e:
        print(f"[!] Error con cliente {addr}: {e}")
```

```
    finally:
        conn.close()

def start_server():
    print(f"Servidor escuchando en {HOST}:{PORT}")
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((HOST, PORT))
    s.listen()
    while True:
        conn, addr = s.accept()
        threading.Thread(target=handle_client, args=(conn, addr)).start()

if __name__ == "__main__":
    start_server()
```

7.3 Código para test_dsa.py

```
import oqs
import psutil
import json
import csv
import socket
import argparse
import time
from time import perf_counter_ns
from cryptography.hazmat.primitives.asymmetric import rsa, ec
from cryptography.hazmat.primitives import hashes, serialization

# ==================== Configuración general ====================
SERVER_HOST = "192.168.100.11"
SERVER_PORT = 5000
```

```
# ================== Función de envío estable ==================
def enviar_y_recibir(alg):
    try:
        with socket.create_connection((SERVER_HOST, SERVER_PORT), timeout=2) as
s:
            payload = json.dumps({"alg": alg}).encode()
            s.sendall(payload)
            s.settimeout(3)
            data = s.recv(4096)
            if not data:
                raise ValueError("Respuesta vacía del servidor")
            return json.loads(data.decode())
    except Exception as e:
        return {"error": str(e)}
    finally:
        time.sleep(0.3)  # pausa para liberar el socket del servidor


# ================== Algoritmos por nivel ==================
niveles = {
    "Nivel 1": {
        "PQC-DSA": ["Dilithium2"],
        "Clasico-DSA": ["RSA-3072", "ECDSA-P256"],
    },
    "Nivel 3": {
        "PQC-DSA": ["Dilithium3"],
        "Clasico-DSA": ["ECDSA-P384"],
    },
    "Nivel 5": {
        "PQC-DSA": ["Dilithium5"],
        "Clasico-DSA": ["ECDSA-P521"],
```

```
    },
}

# ==================== Medición de rendimiento ====================
def medir_rendimiento(alg, iteraciones):
    resultados = []

    for i in range(1, iteraciones + 1):
        try:
            proceso = psutil.Process()
            mem_inicial = proceso.memory_info().rss
            mensaje = b"Mensaje de prueba para firma digital"

            # --- Generación de clave ---
            t1 = perf_counter_ns()

            if "Dilithium" in alg:
                with oqs.Signature(alg) as signer:
                    pk = signer.generate_keypair()
                    ciclos_keygen = perf_counter_ns() - t1

                    # --- Firma ---
                    t2 = perf_counter_ns()
                    firma = signer.sign(mensaje)
                    ciclos_operacion_1 = perf_counter_ns() - t2

                    # --- Enviar al servidor (verificación) ---
                    t3 = perf_counter_ns()
                    resp = enviar_y_recibir(alg)
                    ciclos_operacion_2 = perf_counter_ns() - t3
```

```
        mem_final = proceso.memory_info().rss
        ram_pico = mem_final - mem_inicial
        tam_pk = len(pk)
        tam_sk = len(signer.export_secret_key())
        tam_texto = len(firma)

    elif "RSA" in alg:
        key = rsa.generate_private_key(public_exponent=65537, key_size=3072)
        ciclos_keygen = perf_counter_ns() - t1

        # --- Firma ---
        t2 = perf_counter_ns()
        firma = key.sign(
            mensaje,
            padding=rsa.padding.PKCS1v15(),
            algorithm=hashes.SHA256(),
        )
        ciclos_operacion_1 = perf_counter_ns() - t2

        # --- Enviar al servidor ---
        t3 = perf_counter_ns()
        resp = enviar_y_recibir(alg)
        ciclos_operacion_2 = perf_counter_ns() - t3

        mem_final = proceso.memory_info().rss
        ram_pico = mem_final - mem_inicial
        tam_pk = len(
            key.public_key().public_bytes(
                encoding=serialization.Encoding.PEM,
```

```
            format=serialization.PublicFormat.SubjectPublicKeyInfo,
        )
    )
    tam_sk = len(
        key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption(),
        )
    )
    tam_texto = len(firma)

elif "ECDSA" in alg:
    curva = {
        "ECDSA-P256": ec.SECP256R1(),
        "ECDSA-P384": ec.SECP384R1(),
        "ECDSA-P521": ec.SECP521R1(),
    }.get(alg)

    key = ec.generate_private_key(curva)
    ciclos_keygen = perf_counter_ns() - t1

    # --- Firma ---
    t2 = perf_counter_ns()
    firma = key.sign(mensaje, ec.ECDSA(hashes.SHA256()))
    ciclos_operacion_1 = perf_counter_ns() - t2

    # --- Enviar al servidor ---
    t3 = perf_counter_ns()
    resp = enviar_y_recibir(alg)
```

```
    ciclos_operacion_2 = perf_counter_ns() - t3

    mem_final = proceso.memory_info().rss
    ram_pico = mem_final - mem_inicial
    tam_pk = len(
        key.public_key().public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo,
        )
    )
    tam_sk = len(
        key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption(),
        )
    )
    tam_texto = len(firma)

else:
    raise ValueError(f"Algoritmo {alg} no reconocido")

resultados.append({
    "Algoritmo": alg,
    "Iteración": i,
    "Ciclos_keygen_ns": ciclos_keygen,
    "Ciclos_operacion_1_ns": ciclos_operacion_1,
    "Ciclos_operacion_2_ns": ciclos_operacion_2,
    "Ram_pico_bytes": ram_pico,
    "Tamano_pk_bytes": tam_pk,
```

```
            "Tamano_sk_bytes": tam_sk,
            "Tamano_text_bytes": tam_texto
        })

    except Exception as e:
        print(f"[!] Error en {alg}: {e}")

  return resultados

# ==================== Ejecución principal ====================
def main():
  parser = argparse.ArgumentParser()
  parser.add_argument("--iter", type=int, default=10, help="Número de iteraciones")
  args = parser.parse_args()

  with open("resultados_dsa_full.csv", "w", newline="") as csvfile:
    fieldnames = [
      "Algoritmo", "Iteración", "Ciclos_keygen_ns",
      "Ciclos_operacion_1_ns", "Ciclos_operacion_2_ns",
      "Ram_pico_bytes", "Tamano_pk_bytes",
      "Tamano_sk_bytes", "Tamano_text_bytes"
    ]
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()

    print("\n=== Benchmark extendido de DSA ===\n")

    for nivel, grupos in niveles.items():
      print(f"\n ◇  {nivel}")
      for categoria, algoritmos in grupos.items():
```

```
            for alg in algoritmos:
                print(f"  → Ejecutando {alg} ({categoria})")
                resultados = medir_rendimiento(alg, args.iter)
                for fila in resultados:
                    writer.writerow(fila)

    print("\n☑ Resultados guardados en resultados_dsa_full.csv\n")

if __name__ == "__main__":
    main()
```

7.4 Código para test_kem.py

```
import oqs
import psutil
import json
import csv
import socket
import argparse
import time
from time import perf_counter_ns
from cryptography.hazmat.primitives.asymmetric import rsa, ec
from cryptography.hazmat.primitives import serialization

# ==================== Configuración general ====================
SERVER_HOST = "192.168.100.11"
SERVER_PORT = 5000

# ==================== Función de envío estable ====================
def enviar_y_recibir(alg):
```

```
    try:
        with socket.create_connection((SERVER_HOST, SERVER_PORT), timeout=2) as
s:
            payload = json.dumps({"alg": alg}).encode()
            s.sendall(payload)
            s.settimeout(3)
            data = s.recv(4096)
            if not data:
                raise ValueError("Respuesta vacía del servidor")
            return json.loads(data.decode())
    except Exception as e:
        return {"error": str(e)}
    finally:
        time.sleep(0.3)  # pausa para liberar el socket del servidor


# ==================== Algoritmos por nivel ====================
niveles = {
    "Nivel 1": {
        "PQC-KEM": ["Kyber512"],
        "Clasico-KEM": ["RSA-3072", "ECDH-P256"],
    },
    "Nivel 3": {
        "PQC-KEM": ["Kyber768"],
        "Clasico-KEM": ["ECDH-P384"],
    },
    "Nivel 5": {
        "PQC-KEM": ["Kyber1024"],
        "Clasico-KEM": ["ECDH-P521"],
    },
}
```

```
# ================== Medición de rendimiento ==================
def medir_rendimiento(alg, iteraciones):
    resultados = []

    for i in range(1, iteraciones + 1):
        try:
            proceso = psutil.Process()
            mem_inicial = proceso.memory_info().rss

            t1 = perf_counter_ns()

            # --- Operación KeyGen ---
            if "Kyber" in alg:
                with oqs.KeyEncapsulation(alg) as client:
                    pk = client.generate_keypair()
                    ciclos_keygen = perf_counter_ns() - t1

                    # --- Operación principal cliente (Encaps) ---
                    t2 = perf_counter_ns()
                    ct, ss_client = client.encap_secret(pk)
                    ciclos_operacion_1 = perf_counter_ns() - t2

                    # --- Enviar al servidor ---
                    t3 = perf_counter_ns()
                    resp = enviar_y_recibir(alg)
                    ciclos_operacion_2 = perf_counter_ns() - t3

                    mem_final = proceso.memory_info().rss
                    ram_pico = mem_final - mem_inicial
```

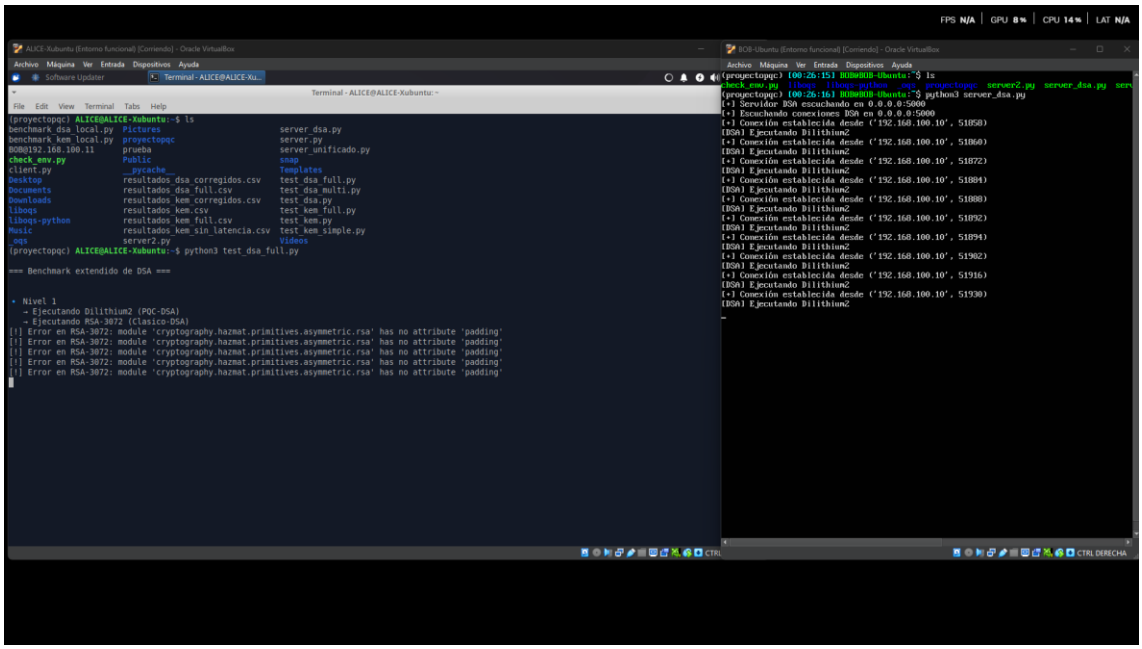
```
        tam_pk = len(pk)
        tam_sk = len(client.export_secret_key())
        tam_texto = len(ct)

    elif "RSA" in alg:
        key = rsa.generate_private_key(public_exponent=65537, key_size=3072)
        ciclos_keygen = perf_counter_ns() - t1

        t2 = perf_counter_ns()
        pubkey = key.public_key()
        pub_bytes = pubkey.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo,
        )
        ciclos_operacion_1 = perf_counter_ns() - t2

        t3 = perf_counter_ns()
        resp = enviar_y_recibir(alg)
        ciclos_operacion_2 = perf_counter_ns() - t3

        mem_final = proceso.memory_info().rss
        ram_pico = mem_final - mem_inicial
        tam_pk = len(pub_bytes)
        tam_sk = len(
            key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.PKCS8,
                encryption_algorithm=serialization.NoEncryption(),
            )
        )
```

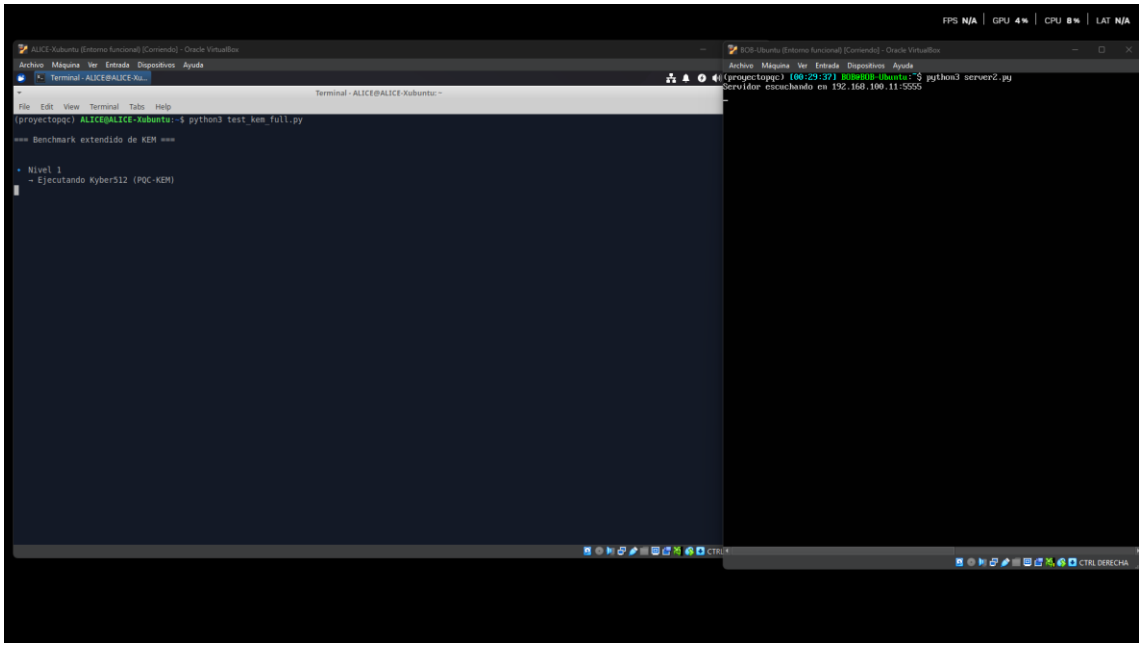
```
    tam_texto = tam_pk

elif "ECDH" in alg:
    curve = {
        "ECDH-P256": ec.SECP256R1(),
        "ECDH-P384": ec.SECP384R1(),
        "ECDH-P521": ec.SECP521R1(),
    }.get(alg)

    key = ec.generate_private_key(curve)
    ciclos_keygen = perf_counter_ns() - t1

    t2 = perf_counter_ns()
    pubkey = key.public_key()
    pub_bytes = pubkey.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo,
    )
    ciclos_operacion_1 = perf_counter_ns() - t2

    t3 = perf_counter_ns()
    resp = enviar_y_recibir(alg)
    ciclos_operacion_2 = perf_counter_ns() - t3

    mem_final = proceso.memory_info().rss
    ram_pico = mem_final - mem_inicial
    tam_pk = len(pub_bytes)
    tam_sk = len(
        key.private_bytes(
            encoding=serialization.Encoding.PEM,
```

```
                format=serialization.PrivateFormat.PKCS8,
                encryption_algorithm=serialization.NoEncryption(),
            )
        )
        tam_texto = tam_pk

    else:
        raise ValueError(f"Algoritmo {alg} no reconocido")

    resultados.append({
        "Algoritmo": alg,
        "Iteración": i,
        "Ciclos_keygen_ns": ciclos_keygen,
        "Ciclos_operacion_1_ns": ciclos_operacion_1,
        "Ciclos_operacion_2_ns": ciclos_operacion_2,
        "Ram_pico_bytes": ram_pico,
        "Tamano_pk_bytes": tam_pk,
        "Tamano_sk_bytes": tam_sk,
        "Tamano_text_bytes": tam_texto
    })

  except Exception as e:
    print(f"[!] Error en {alg}: {e}")

return resultados

# ==================== Ejecución principal ====================
def main():
  parser = argparse.ArgumentParser()
  parser.add_argument("--iter", type=int, default=10, help="Número de iteraciones")
```

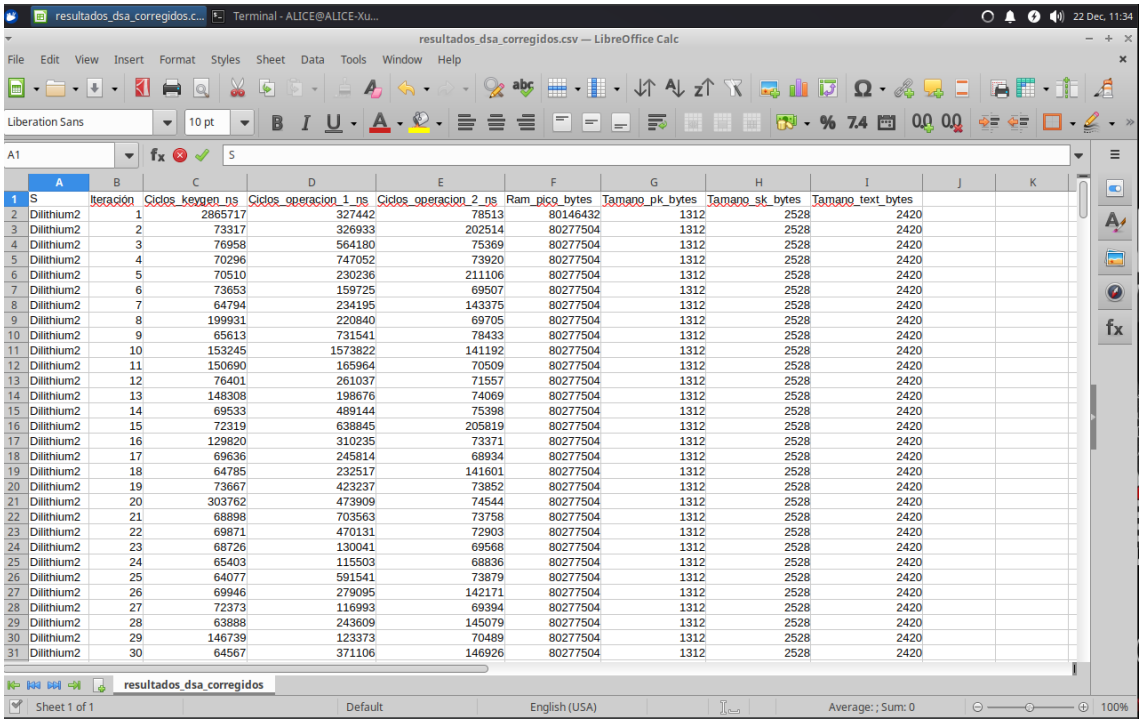
```
    args = parser.parse_args()

    with open("resultados_kem_full.csv", "w", newline="") as csvfile:
        fieldnames = [
            "Algoritmo", "Iteración", "Ciclos_keygen_ns",
            "Ciclos_operacion_1_ns", "Ciclos_operacion_2_ns",
            "Ram_pico_bytes", "Tamano_pk_bytes",
            "Tamano_sk_bytes", "Tamano_text_bytes"
        ]
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

        print("\n=== Benchmark extendido de KEM ===\n")

        for nivel, grupos in niveles.items():
            print(f"\n ◇  {nivel}")
            for categoria, algoritmos in grupos.items():
                for alg in algoritmos:
                    print(f"  → Ejecutando {alg} ({categoria})")
                    resultados = medir_rendimiento(alg, args.iter)
                    for fila in resultados:
                        writer.writerow(fila)

    print("\n☑ Resultados guardados en resultados_kem_full.csv\n")

if __name__ == "__main__":
    main()
```

7.5 Ejecución del entorno de pruebas



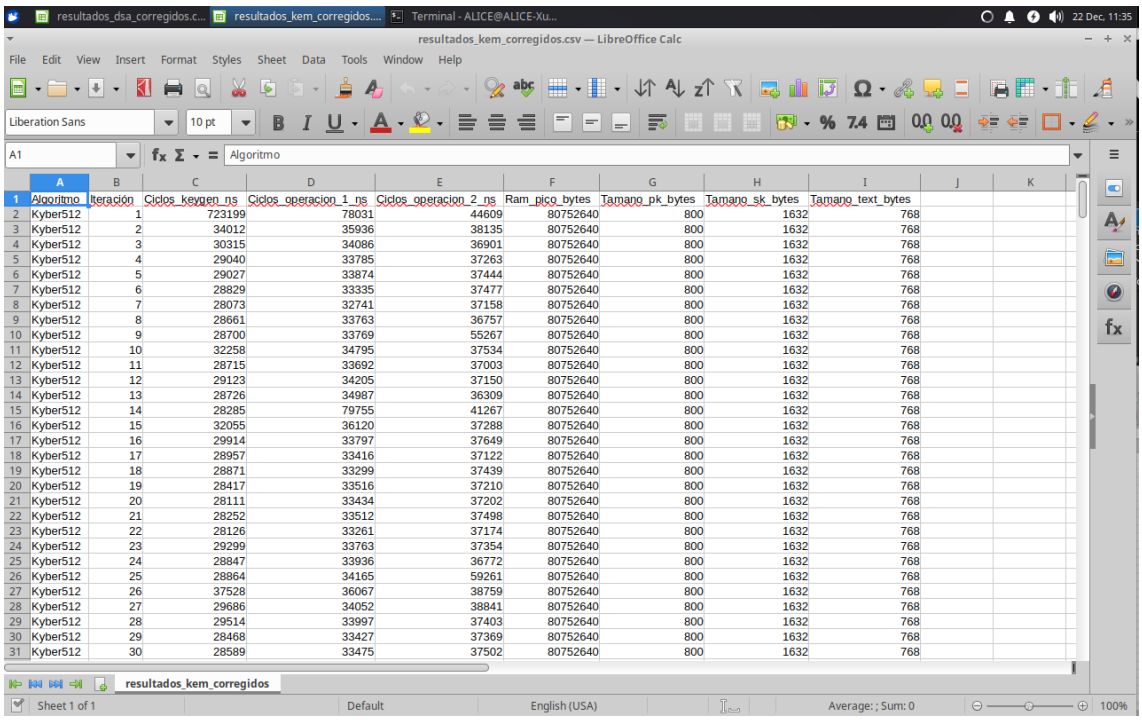


7.6 Resultados de ejecución



resultados_dsa_corregidos.csv — LibreOffice Calc

S	Iteración	Ciclos_keygen_ns	Ciclos_operacion_1_ns	Ciclos_operacion_2_ns	Ram_pico_bytes	Tamano_pk_bytes	Tamano_sk_bytes	Tamano_text_bytes
Dilithium2	1	2865717	327442	78513	80146432	1312	2528	2420
Dilithium2	2	73317	326933	202514	80277504	1312	2528	2420
Dilithium2	3	76958	564180	75369	80277504	1312	2528	2420
Dilithium2	4	70296	747052	73920	80277504	1312	2528	2420
Dilithium2	5	70510	230236	211106	80277504	1312	2528	2420
Dilithium2	6	73653	159725	69507	80277504	1312	2528	2420
Dilithium2	7	64794	234195	143375	80277504	1312	2528	2420
Dilithium2	8	199931	220840	69705	80277504	1312	2528	2420
Dilithium2	9	65613	731541	78433	80277504	1312	2528	2420
Dilithium2	10	153245	1573822	141192	80277504	1312	2528	2420
Dilithium2	11	150690	165964	70509	80277504	1312	2528	2420
Dilithium2	12	76401	261037	71557	80277504	1312	2528	2420
Dilithium2	13	148308	198676	74069	80277504	1312	2528	2420
Dilithium2	14	69533	489144	75398	80277504	1312	2528	2420
Dilithium2	15	72319	638845	205819	80277504	1312	2528	2420
Dilithium2	16	129820	310235	73371	80277504	1312	2528	2420
Dilithium2	17	69636	245814	68934	80277504	1312	2528	2420
Dilithium2	18	64785	232517	141601	80277504	1312	2528	2420
Dilithium2	19	73667	423237	73852	80277504	1312	2528	2420
Dilithium2	20	303762	473909	74544	80277504	1312	2528	2420
Dilithium2	21	68898	703563	73758	80277504	1312	2528	2420
Dilithium2	22	69871	470131	72903	80277504	1312	2528	2420
Dilithium2	23	68726	130041	69568	80277504	1312	2528	2420
Dilithium2	24	65403	115503	68836	80277504	1312	2528	2420
Dilithium2	25	64077	591541	73879	80277504	1312	2528	2420
Dilithium2	26	69946	279095	142171	80277504	1312	2528	2420
Dilithium2	27	72373	116993	69394	80277504	1312	2528	2420
Dilithium2	28	63888	243609	145079	80277504	1312	2528	2420
Dilithium2	29	146739	123373	70489	80277504	1312	2528	2420
Dilithium2	30	64567	371106	146926	80277504	1312	2528	2420



resultados_kem_corregidos.csv — LibreOffice Calc

Algoritmo	Iteración	Ciclos_keygen_ns	Ciclos_operacion_1_ns	Ciclos_operacion_2_ns	Ram_pico_bytes	Tamano_pk_bytes	Tamano_sk_bytes	Tamano_text_bytes
Kyber512	1	723199	78031	44609	80752640	800	1632	768
Kyber512	2	34012	35936	38135	80752640	800	1632	768
Kyber512	3	30315	34086	36901	80752640	800	1632	768
Kyber512	4	29040	33785	37263	80752640	800	1632	768
Kyber512	5	29027	33874	37444	80752640	800	1632	768
Kyber512	6	28829	33335	37477	80752640	800	1632	768
Kyber512	7	28073	32741	37158	80752640	800	1632	768
Kyber512	8	28661	33763	36757	80752640	800	1632	768
Kyber512	9	28700	33769	55267	80752640	800	1632	768
Kyber512	10	32258	34795	37534	80752640	800	1632	768
Kyber512	11	28715	33692	37003	80752640	800	1632	768
Kyber512	12	29123	34205	37150	80752640	800	1632	768
Kyber512	13	28726	34987	36309	80752640	800	1632	768
Kyber512	14	28285	79755	41267	80752640	800	1632	768
Kyber512	15	32055	36120	37288	80752640	800	1632	768
Kyber512	16	29914	33797	37649	80752640	800	1632	768
Kyber512	17	28957	33416	37122	80752640	800	1632	768
Kyber512	18	28871	33299	37439	80752640	800	1632	768
Kyber512	19	28417	33516	37210	80752640	800	1632	768
Kyber512	20	28111	33434	37202	80752640	800	1632	768
Kyber512	21	28252	33512	37498	80752640	800	1632	768
Kyber512	22	28126	33261	37174	80752640	800	1632	768
Kyber512	23	29299	33763	37354	80752640	800	1632	768
Kyber512	24	28847	33936	36772	80752640	800	1632	768
Kyber512	25	28864	34165	59261	80752640	800	1632	768
Kyber512	26	37528	36067	38759	80752640	800	1632	768
Kyber512	27	29686	34052	38841	80752640	800	1632	768
Kyber512	28	29514	33997	37403	80752640	800	1632	768
Kyber512	29	28468	33427	37369	80752640	800	1632	768
Kyber512	30	28589	33475	37502	80752640	800	1632	768