ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Análisis y optimización de algoritmos criptográficos post-cuánticos en sistemas embebidos

Analysis and optimization of post-quantum cryptographic algorithms in embedded systems

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Guillermo Fernández Osle

Septiembre - 2025

Agradecimientos

Quiero expresar mi agradecimiento a mi tutor Luis Muñoz Gutiérrez, tanto por su dedicación y orientación a lo largo del desarrollo del proyecto, como por su excelente docencia en el grado.

Asimismo, me gustaría agradecer a los docentes del grado que me han brindado su apoyo y transmitido sus conocimientos durante todo este camino.

A mis amigos y compañeros, gracias por su compañía, ánimos y tantos buenos momentos en el camino.

Finalmente, a mi familia y pareja, gracias por su apoyo incondicional y confianza para traerme hasta aquí, incluso en los momentos más difíciles.

Este trabajo es también fruto del acompañamiento de todas estas personas, a quienes siempre estaré agradecido.

Índice general

Índice de figuras 1
Índice de códigos 2
1. Introducción a las comunicaciones y la criptografía 5
2. Estado del arte 10
2.1. Historia y fundamentos de la computación cuántica 10
2.1.1. Principios Fundamentales de la Mecánica Cuántica 12
2.1.2. Fundamentos de la computación cuántica 15
2.1.3. Relación con la Computación Cuántica 15
2.1.4. Desarrollo Tecnológico 16
2.2. Criptografía y seguridad en las comunicaciones 17
2.2.1. Criptografía simétrica 17
2.2.2. Criptografía asimétrica 18
2.2.3. Vulnerabilidades ante la computación cuántica 20
2.2.4. Algoritmos post-cuánticos 22
2.3. Análisis comparativo y plataformas de prueba 24
2.3.1. Importancia de medición de rendimiento en sistemas reales 24
2.3.2. Diferencias entre arquitecturas 25
2.3.3. Herramientas y métodos de medición 27
3. Desarrollo del proyecto 29
3.1. Planteamiento inicial y entorno de pruebas 29
3.1.1. Planteamiento inicial y objetivos operativos 29
3.1.2. Entornos evaluados 30
3.1.3. Software y dependencias 33
3.2. Implementación del análisis comparativo 34
3.2.1. Metodología experimental 34
3.2.2. Módulo criptográfico ML-KEM 35
3.2.3. Medición de ciclos por arquitectura 36
3.2.4. Programas desarrollados 38
3.2.5. Configuración del sistema 39
3.2.6. Automatización y scripts 39
3.3. Análisis estadístico 40

3.3.1. Métricas calculadas 41
3.3.2. Histogramas y funciones de densidad de probabilidad 42
3.3.3. Comparativa de rendimiento 43
3.3.4. Interpretación de los resultados 44
3.4. Diseño e implementación del protocolo cliente-servidor 45
3.4.1. Arquitectura general 46
3.4.2. Requisitos funcionales y no funcionales 47
3.4.3. Modelado de mensajes y estructuras 48
3.4.4. Implementación y sincronización 49
3.5. Algoritmo de decisión de longitud de clave 51
3.5.1. Objetivo y criterios 51
3.5.2. Cálculo de decisión 51
3.5.3. Justificación estadística y umbrales 52
3.6. Validación y pruebas 53
4. Conclusiones 55
5. Lista de acrónimos 57
6. Bibliografía 59

Índice de figuras

Figura 1: Experimento doble rendija 12
Figura 2:Computador cuántico de IBM 16
Figura 3: Raspberry Pi 4 y 3 respectivamente 32
Figura 4: Datos de entrada (número de ciclos) 41
Figura 5: Análisis estadístico Raspberry Pi 4 con KEM768 41
Figura 6: Histograma para Raspberry Pi 5 con KEM768 42
Figura 7: Densidad de probabilidad Raspberry Pi 4 con KEM768 43
Figura 8: Comparación de histogramas para KEM768 44
Figura 9: Comparativa estadística con KEM 768 45
Figura 10: Diagrama de secuencia del protocolo 46
Figura 11: Estructuras de paquete 48
Figura12: Recepción de datos y envió de ACK 50
Figura 13: Percentiles 95 51
Figura 14: Selección de nivel de seguridad 52
Figura 15: Raspberry Pi 4 como cliente 53
Figura 16: Raspberry Pi 3 como servidor 54

Índice de códigos

Código 1: Resultados VM 31
Código 2:Implementacino de medición de ciclos con perf_event 37
Código 3: Función para leer contador 38
Código 4: Obtención de ciclos totales 38

Resumen

En este trabajo de fin de grado se propone una implementación del algoritmo *Federal Information Processing Standards* (FIPS*)* 203 creado por el *National Institute of Standards and Technology* (NIST) para crear un canal de comunicaciones entre dispositivos con bajas capacidades de cómputo, resistente a ataques originados por computadores cuánticos. El documento introduce al lector en el mundo de las comunicaciones, analizando aspectos como la seguridad de la información y la criptografía, para exponer los problemas actuales y la motivación para mitigarlos. Posteriormente, se presenta el estado del arte sobre los fundamentos de la computación cuántica, la criptografía, la seguridad en las comunicaciones y una evaluación comparativa de plataformas de pruebas.

En el desarrollo del proyecto, se plantea una solución para proteger dispositivos con recursos limitados frente a ataques basados en el uso de computadores cuánticos. Se plantea de un entorno inicial con máquinas virtuales para, posteriormente, analizar dispositivos más limitados, estudiar sus resultados y diseñar e implementar un protocolo cliente-servidor que permita decidir el nivel de seguridad óptimo en función de la capacidad de cómputo, gracias a un algoritmo de decisión adaptativo.

Palabras clave: criptografía post-cuántica; FIPS 203 NIST; seguridad en comunicaciones; dispositivos IoT; benchmarking; protocolo cliente-servidor; computación cuántica

Abstract

This Bachelor's dissertation proposes an implementation of the FIPS 203 algorithm, developed by NIST, to establish a communication channel between devices with low computational capabilities, resistant to quantum computer attacks. The document introduces the reader to the world of communications, analysing aspects such as information security and cryptography, in order to present current challenges and the motivation to address them. Subsequently, the state of the art is presented, covering the fundamentals of quantum computing, cryptography, communication security, and the use of benchmarking and testing platforms.

In the development of the project, a solution is proposed to secure resource-constrained devices against quantum computing based attacks. The work begins with an initial environment based on virtual machines and then moves on to analyze more limited devices, study their results, and design and implement a client-server protocol that allows the optimal security level to be selected according to the computational capacity, thanks to an adaptive decision algorithm.

Keywords: post-quantum cryptography; FIPS 203 NIST; communication security; IoT devices; benchmarking; client-server protocol; quantum computing

1. Introducción a las comunicaciones y la criptografía

La necesidad de comunicarse en la especie humana ha sido sin duda una de las claves del desarrollo tecnológico adquirido en toda nuestra historia. Actualmente, vivimos en un mundo en el que nos comunicamos a través de una plétora de fibras ópticas, sobre las cuales se propaga la información a la velocidad de la luz, permitiendo la comunicación con cualquier parte del mundo en tan solo unos instantes. Para alcanzar este nivel de desarrollo, el ser humano ha utilizado todas las herramientas contemporáneas para crear nuevas formas de comunicación, desde pinturas en cuevas, jeroglíficos, señales de humo, hasta pasar a inventos tan revolucionarios como el telégrafo, el teléfono, la radio o la televisión.

Desde la prehistoria el ser humano ha mantenido interés y necesidad por intercambiar mensajes a distancia. Las señales de humo fueron utilizadas por las tribus americanas entre otras, para alertar sobre posibles peligros o coordinar movimientos a distancia. Por otro lado, en la antigua Grecia se utilizaban antorchas y espejos para transmitir señales luminosas entre torres situadas en colinas. Cada civilización creó sus propios métodos de comunicación, de forma que se adaptaban a las tecnologías propias de su contexto histórico.

La llegada de la escritura abrió puerta a la posibilidad de compartir y registrar información de manera más precisa y duradera. Los antiguos papiros egipcios, las tablillas de arcilla de Mesopotamia y los pergaminos romanos hicieron posible que mensajes, leyes y conocimientos viajaran a lomos de caballos, en barcos o a través de rutas comerciales como la famosa Ruta de la Seda. Durante siglos, los mensajeros a caballo, como los célebres chasquis del Imperio Inca o el sistema de postas persa, fueron el principal medio para llevar información, aunque su alcance y velocidad estaban limitados por la geografía y las condiciones del terreno.

Sin embargo, el siglo XIX revolucionó la historia de las telecomunicaciones. Esta revolución se inició con el telégrafo eléctrico, un dispositivo que, utilizando impulsos eléctricos, era capaz de transmitir información a grandes distancias a través de cables tendido entre ciudades e incluso continentes. El sistema de código Morse [1], desarrollado por Samuel Morse, permitía codificar la información de forma eficiente a través de puntos y rayas, representados como pulsaciones cortas o largas respectivamente. Este sistema revolucionó por completo sectores como el periodismo, el comercio o la gestión de emergencias, sentando las bases de la sociedad de la información moderna.

El telégrafo cableado fue un gran invento, pero lo mejor estaba por llegar. Guglielmo Marconi desarrolló un telégrafo inalámbrico a finales del siglo XIX, siendo uno de los pioneros en el desarrollo de la radio. La primera comunicación inalámbrica de la historia tuvo lugar en 1899 y logró comunicar dos extremos entre el Canal de la Mancha [2]. Dos años después, en 1901, se dio la primera comunicación transatlántica [3] gracias a este telégrafo inalámbrico. Este gran avance permitió enviar información sin necesidad de cables, abriendo la puerta a la era de las comunicaciones inalámbricas y sentando las bases para el desarrollo de la radio, la televisión y, posteriormente, las redes móviles y el *Wireless Fidelity* (Wi-Fi).

En 1876, Alexander Graham Bell [4] desarrolló el teléfono, un dispositivo que permitía la comunicación de voz en tiempo real entre dos puntos situados a kilómetros de distancia. Inicialmente, su distribución fue limitada, como en la mayoría de nuevas tecnologías, pero rápidamente se incrementó su distribución hasta que a principios del siglo XX las redes telefónicas ya no solo conectaban pequeñas ciudades, sino que conectaban países, transformando por completo la vida y los negocios. La televisión, desarrollada en la década de 1920 por inventores como John Logie Baird y Philo Farnsworth, añadió la imagen al sonido y revolucionó la cultura, la educación y el entretenimiento.

El siglo XX supuso el inicio de una nueva era, la informática y la digitalización acababan de llegar para reinventar las comunicaciones. El desarrollo de los ordenadores personales, las redes y la creación de Internet, han dado lugar a una sociedad hiperconectada. En 1969, un proyecto militar destinado al intercambio resiliente de información entre distintas bases militares llamado *Advanced Research Projects Agency Network* (ARPANET) [5], fue el inicio de lo que hoy conocemos como internet, expandiéndose a nivel global para dar lugar a el correo electrónico, las páginas web y la mensajería instantánea entre otros servicios.

Lo que conocemos como telefonía móvil, comenzó en la década de 1980 con terminales muy voluminosos y costosos, sin embargo, la tecnología avanzo bruscamente y permitió crear terminales como los que tenemos hoy en día. La llegada de los smartphones y las redes 3G, 4G y 5G [6] ha hecho posible que miles de millones de personas estén conectadas en todo momento, accediendo a información y servicios desde casi cualquier lugar del planeta.

En la actualidad, la mayoría de los objetos cotidianos como los vehículos o electrodomésticos están conectados haciendo tangible el paradigma de la *Internet de las Cosas* o *Internet of Things* (IoT) [7], una red que permite automatizar procesos, monitorizar y controlar de forma remota aquellos. Esta tecnología no solo aplica en el ámbito doméstico, también es fundamental en entornos industriales para control y monitorización de procesos.

Todos estos avances han sido fundamentales para el desarrollo de la humanidad a lo largo de la historia, pero hoy en día la sociedad se enfrenta a nuevos retos en las comunicaciones. Consciente o inconscientemente, todos los días se intercambia información entre dispositivos, estando está en riesgo de ser robada por terceras partes para fines maliciosos.

La criptografía, inicialmente utilizada para entornos militares y diplomáticos, es la rama dedicada a proteger los mensajes que viajan a través de la red. Se ha convertido en una herramienta esencial para garantizar la confidencialidad, la integridad y la autenticidad de los datos, utilizando diferentes tipos de algoritmos que se presentaran más adelante.

El exponencial crecimiento de las redes y de servicios críticos, han traído consigo riesgos de ciberataques. El robo de datos o el espionaje son cada vez más refinados y sofisticados, siendo actualmente el nuevo reto de las comunicaciones. Algunos algoritmos cuánticos, como el de Shor [8] podrían romper los sistemas criptográficos actuales, poniendo en peligro la seguridad de las comunicaciones.

En este contexto, la criptografía post-cuántica [9] surge como solución al auge de los computadores cuánticos y su gran capacidad de romper algunos de los algoritmos actualmente utilizados. El objetivo de este trabajo es enfrentar abordar este problema y proponer una solución para sistemas embebidos. En este caso, se implementa y evalúa el uso del algoritmo FIPS 203, concretamente el algoritmo *Module Lattice-Based Key-*

Encapsulation Mechanism (ML-KEM) [10] en sistemas embebidos, planteando una solución para proteger estos sistemas de ataques. Además, se busca analizar el rendimiento de estos sistemas en plataformas reales y proponer un algoritmo de decisión que permita seleccionar dinámicamente el nivel de seguridad óptimo.

Una vez presentada la introducción histórica de las comunicaciones y la importancia de la seguridad en las mismas, se expone un estado del arte en principios básicos cuánticos, criptografía y *benchmarking*. A continuación, se presenta el desarrollo del proyecto, desarrollando el entorno, la metodología, la implementación y los resultados. Finalmente, se exponen las conclusiones y se proponen posibles futuras líneas de trabajo.

2. Estado del arte

2.1. Historia y fundamentos de la computación cuántica

La física cuántica surgió a principios del siglo XX como una revolución que desafió las nociones de la física clásica, abriendo la puerta a un mundo de fenómenos a escala atómica y subatómica [11]. Su historia se inicia con la famosa "catástrofe ultravioleta", un problema que la física de la época no podía resolver al intentar explicar la radiación de un cuerpo negro. En 1900, Max Planck propuso audazmente que la energía se emitía y absorbía en paquetes discretos o "cuantos" [12], una idea que sentó las bases de toda la teoría. Poco después, en 1905, Albert Einstein utilizó la misma noción para explicar el efecto fotoeléctrico, proponiendo que la luz estaba compuesta por partículas de energía llamadas fotones [13]. Estos descubrimientos, aunque inicialmente vistos con escepticismo, allanaron el camino para que otros pioneros como Niels Bohr y Werner Heisenberg desarrollaran modelos del átomo y el famoso principio de incertidumbre, respectivamente. Este camino culminaría con el trabajo de Erwin Schrödinger, cuya ecuación fundamental describió el comportamiento de las partículas cuánticas en términos de "funciones de onda", marcando un hito en nuestra comprensión del universo.

El estado de un sistema físico se define por el conjunto de sus propiedades observables, tales como la posición, la velocidad o la temperatura. En la mecánica clásica, este estado se describe mediante funciones matemáticas que son completamente deterministas de posición y de tiempo. Esto significa que, si se conocen

las condiciones iniciales del sistema, su comportamiento futuro puede predecirse con total exactitud. El modelo clásico asume que todas las propiedades de un sistema pueden ser medidas simultáneamente y con una precisión ilimitada. Por el contrario, el estado cuántico de un sistema, aunque también está ligado a sus propiedades observables, se describe de una manera fundamentalmente distinta. Su evolución está regida por la función de onda, que a su vez obedece a la ecuación de Schrödinger. Esta descripción es válida solo mientras el sistema no interactúe con el entorno a través de una medición, lo cual representa una diferencia radical con el enfoque clásico.

La ecuación de Schrödinger es el sustento de la mecánica cuántica. A diferencia de las leyes de movimiento de Newton, que predicen la trayectoria exacta de un objeto, esta ecuación no nos da la posición precisa de una partícula. En su lugar, describe cómo evoluciona la función de onda (representada por la letra griega *psi*, *Ψ*) de una partícula a lo largo del tiempo y el espacio. Esta función de onda es una herramienta matemática que contiene toda la información posible sobre el estado de la partícula, incluyendo la probabilidad de encontrarla en un lugar determinado. La ecuación puede verse como un mapa que nos permite predecir el comportamiento probabilístico de los sistemas cuánticos, revelando la dualidad onda-partícula de la materia, uno de los principios más extraños y fascinantes de este campo [14].

$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) = \hat{H}\Psi(\mathbf{r},t) \tag{2.1}$$

El operador Hamiltoniano $\hat{H}$ referenciado en la ecuación (2.2), es un elemento central para la evolución de los sistemas cuánticos. Este operador representa la energía total del sistema, la suma de su energía cinética y potencial. En esencia, dirige cómo evoluciona la función de onda en el tiempo. Así, el Hamiltoniano actúa como el motor de la dinámica cuántica [15].

$$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r},t) \tag{2.2}$$

Mientras un sistema evoluciona sin ser perturbado por una observación, su estado no está definido de forma singular, sino que existe en una superposición de estados. Esto implica que la partícula puede ocupar múltiples configuraciones o estados simultáneamente. Solo cuando se realiza una medición se produce el colapso de la función de onda, un proceso que fuerza al sistema a adoptar uno de los estados posibles con una probabilidad específica.

2.1.1. Principios Fundamentales de la Mecánica Cuántica

La dualidad onda-partícula [16] es uno de los conceptos más anti intuitivos y esenciales de la física cuántica. En nuestro mundo cotidiano, las cosas son o bien partículas (como una pelota) o bien ondas (como el sonido o la luz). Sin embargo, a nivel cuántico, la materia y la energía no se adhieren a esta distinción. Partículas como los electrones, y formas de energía como la luz, pueden exhibir comportamientos de onda y de partícula al mismo tiempo. El famoso experimento de la doble rendija, figura 1, lo demuestra de forma contundente: cuando los electrones o fotones se disparan uno a uno hacia una barrera con dos rendijas, no se comportan como partículas que pasan por una u otra, sino que crean un patrón de interferencia similar al de una onda, lo que implica que de alguna manera pasan por ambas rendijas a la vez.

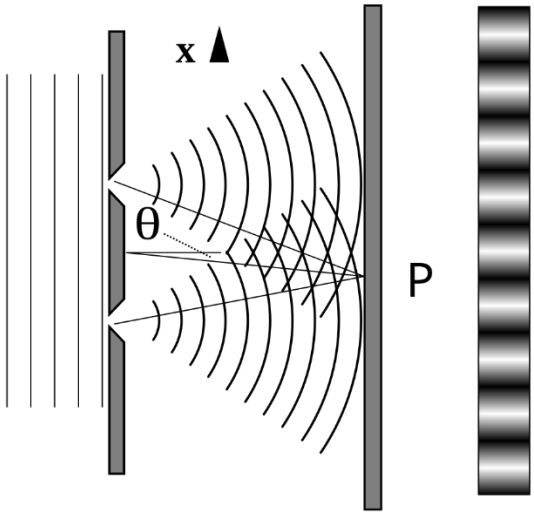


Figura 1: Experimento doble rendija

La cuantización de la energía es la idea de que la energía no es continua, sino que existe en paquetes o "cuantos" discretos. En la física clásica, se pensaba que la energía podía tener cualquier valor, como un termómetro que puede marcar cualquier temperatura. Sin embargo, Max Planck y Albert Einstein demostraron que la energía solo se puede emitir o absorber en valores específicos. El modelo atómico de Bohr ilustró esto al postular que los electrones solo pueden orbitar el núcleo en niveles de energía fijos, y solo pueden saltar entre estos niveles emitiendo o absorbiendo un "cuanto" de energía exacto. El efecto fotoeléctrico es otro ejemplo: la luz solo puede arrancar electrones de un metal si tiene una energía mínima (un cuanto) suficiente, independientemente de su intensidad.

El principio de superposición establece que un sistema cuántico puede existir en múltiples estados posibles de forma simultánea, como si una moneda estuviera, al mismo tiempo, en estado de "cara" y "cruz" antes de ser lanzada. En el mundo de las partículas, un electrón puede estar en múltiples ubicaciones o tener múltiples niveles de espín a la vez. No es hasta que se realiza una medición u observación que la superposición colapsa, forzando al sistema a "elegir" uno de esos estados posibles de manera aleatoria. Un estado de superposición se representa como una combinación lineal de los estados posibles (2.3), donde $|\psi\rangle$ es el estado del sistema, $|0\rangle$ y $|1\rangle$ son los estados base (por ejemplo, cara y cruz de un qubit), y α y β son los coeficientes de probabilidad.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.3}$$

El entrelazamiento [17] es un fenómeno en el que dos o más partículas se vinculan de tal manera que el estado cuántico de una depende del estado de la otra, sin importar la distancia que las separe. La medición de una de las partículas afecta instantáneamente el estado de la otra, como si estuvieran conectadas por un hilo invisible.

Este concepto, que Albert Einstein llamó "acción fantasmal a distancia", fue debatido en el experimento EPR (Einstein-Podolsky-Rosen) y posteriormente confirmado por las desigualdades de Bell. El entrelazamiento es una propiedad que no tiene análogo en la física clásica y es clave para el desarrollo de tecnologías como la computación cuántica.

El principio de incertidumbre de Heisenberg, formulado por Werner Heisenberg en 1927, establece una limitación fundamental en la física cuántica: no se puede conocer simultáneamente y con total precisión pares de propiedades de una partícula, como su posición (x) y su momento lineal (p). Este concepto surge porque cualquier intento de medir una de estas propiedades perturba inevitablemente a la otra, alterando el estado original del sistema. Esta restricción no se debe a la imperfección de nuestros instrumentos de medición, sino que es una característica intrínseca de la naturaleza. La relación se expresa matemáticamente mediante una desigualdad, donde el producto de la incertidumbre en la posición (Δx) y la incertidumbre en el momento (Δp) siempre debe ser mayor o igual a una cantidad fija, la mitad de la constante de Planck reducida ($\hbar$) (2.4). En otras palabras, cuanto más precisamente se conoce la posición de una partícula, menos se puede conocer su momento, y viceversa. Esta constante ($\hbar$) es un valor universal de la naturaleza que desempeña un papel central en la mecánica cuántica.

$$\Delta x \, \Delta p \geq \frac{\hbar}{2} \tag{2.4}$$

La decoherencia es el proceso por el cual un estado cuántico pierde su "coherencia" o propiedades cuánticas (como la superposición y el entrelazamiento) debido a la interacción con el entorno. Cuando un sistema cuántico entra en contacto con el ambiente circundante, por ejemplo, chocando con partículas de aire o luz, la información sobre su estado se "filtra" al entorno. Esto provoca que el sistema pierda sus características cuánticas y comience a comportarse de una manera que se aproxima

al mundo clásico, determinista y familiar. La decoherencia es la razón por la que no observamos fenómenos como la superposición en objetos grandes, ya que la interacción constante con su entorno hace que los efectos cuánticos desaparezcan casi al instante.

2.1.2. Fundamentos de la computación cuántica

La computación cuántica se basa en principios de la mecánica cuántica para procesar información de una manera radicalmente diferente a la de las computadoras clásicas. En lugar del bit tradicional, que solo puede ser 0 o 1, la unidad fundamental de información es el qubit. Gracias al principio de superposición, un qubit puede representar 0, 1, o una combinación de ambos estados simultáneamente [18]. La manipulación de estos qubits se realiza a través de operaciones cuánticas y puertas lógicas, pero que operan sobre la superposición de estados. Finalmente, la información se extrae del sistema mediante el proceso de medición, que provoca el colapso del estado del qubit a un valor clásico de 0 o 1, revelando así el resultado del cálculo.

2.1.3. Relación con la Computación Cuántica

Estos principios cuánticos no solo son teóricos, sino que son la base de los ordenadores cuánticos. La capacidad de los qubits para existir en una superposición permite que una computadora cuántica explore múltiples soluciones a un problema a la vez, lo que se conoce como paralelismo cuántico. Esto contrasta fundamentalmente con la computación clásica, donde los bits deben procesarse de forma secuencial, uno por uno. Este paralelismo inherente a la mecánica cuántica permite a los ordenadores cuánticos resolver ciertos problemas, como la factorización de números grandes o la simulación de sistemas moleculares complejos, de manera exponencialmente más rápida que los ordenadores más potentes de la actualidad [19].

2.1.4. Desarrollo Tecnológico

Los primeros ordenadores cuánticos eran dispositivos experimentales rudimentarios, desarrollados en laboratorios de investigación para validar los principios teóricos. Hoy en día, hemos avanzado a un estado en la que grandes empresas como Google, IBM y Microsoft, junto con laboratorios de élite, están construyendo y operando máquinas cuánticas, como la que se muestra en la figura 2, con decenas y cientos de qubits.

Figura 2:Computador cuántico de IBM

Sin embargo, persisten grandes retos tecnológicos, como la decoherencia y los errores inherentes a los qubits. La decoherencia es la pérdida de la superposición y el entrelazamiento debido a la interacción con el entorno, lo que exige que los ordenadores cuánticos funcionen en condiciones extremadamente frías y aisladas para mantener su estabilidad [20]. A pesar de estos desafíos, la investigación y el desarrollo de tecnologías de corrección de errores cuánticos están en auge, con el objetivo de construir ordenadores cuánticos a gran escala, tolerantes a fallos y funcionales.

2.2. Criptografía y seguridad en las comunicaciones

2.2.1. Criptografía simétrica

La criptografía moderna se fundamenta en la criptografía simétrica y la criptografía asimétrica. La primera se caracteriza por el uso de una misma clave tanto para el cifrado como para el descifrado de la información, lo que implica una alta eficiencia en términos de velocidad y consumo de recursos, aunque también presenta desafíos relacionados con la distribución segura de claves. Dentro de los algoritmos simétricos más relevantes se encuentra el *Data Encryption Standard* (DES), propuesto en la década de 1970, que, si bien marcó un hito en la estandarización de la criptografía, con el tiempo demostró ser vulnerable ante la capacidad de cómputo moderna debido a su longitud de clave de 56 bits. Como respuesta a estas limitaciones surgió el *Advanced Encryption Standard* (AES) [21], adoptado por el NIST en 2001, que opera con longitudes de clave de 128, 192 y 256 bits, ofreciendo una mayor resistencia frente a ataques de fuerza bruta y convirtiéndose en el estándar predominante en la actualidad para la protección de datos sensibles en múltiples ámbitos, desde comunicaciones hasta sistemas embebidos.

La superioridad de AES frente a *Data Encryption Standard* (DES) no radica únicamente en la mayor longitud de clave, sino también en su construcción matemática más compleja. Mientras que DES se basa en permutaciones y sustituciones relativamente simples, AES emplea operaciones algebraicas sobre cuerpos finitos, específicamente el cuerpo de Galois $\mathrm{GF}(2^8)$. Estas operaciones incluyen transformaciones lineales y no lineales, que se deriva de funciones inversas en dicho cuerpo, y mezclas de columnas que explotan propiedades de la teoría de números y del álgebra modular. Gracias a esta estructura, AES ofrece una mayor resistencia frente a ataques avanzados, como la criptoanálisis diferencial o lineal, que resultaron eficaces contra DES en escenarios de claves reducidas. De este modo, AES constituye no solo una ampliación cuantitativa en términos de clave, sino también una mejora cualitativa al incorporar fundamentos matemáticos que incrementan significativamente la seguridad.

2.2.2. Criptografía asimétrica

La criptografía asimétrica, o de clave publica, se ha consolidado como el estándar para garantizar la seguridad en las comunicaciones digitales, resolviendo el desafío de la distribución segura de claves en entornos no seguros. A diferencia de la criptografía simétrica, que emplea una única clave para el cifrado y descifrado, los sistemas asimétricos utilizan un par de claves matemáticamente relacionadas: una clave pública, que puede ser compartida libremente, y una clave privada, que debe mantenerse en secreto.

El principio fundamental es que la información cifrada con la clave pública solo puede ser descifrada por la clave privada correspondiente. Por otro lado, la firma digital utiliza este mismo principio para garantizar la autenticidad y la integridad de aquella. En este caso, el emisor utiliza su clave privada para firmar la información. El receptor, al recibir el mensaje, emplea la clave pública del emisor para verificar la firma, confirmando así que el mensaje proviene del emisor esperado y que no ha sido modificado. Esta dualidad es la base de la autenticación, el no repudio y la confidencialidad en protocolos como *Transport Layer Security* (TLS).

Dentro de este paradigma, los dos algoritmos más relevantes y de mayor uso en la actualidad son *Rivest-Shamir-Adleman* (RSA) [22] y la criptografía basada en curvas elípticas o *Elliptic Curve Cryptography* (ECC). Si bien ambos cumplen la misma función, se basan en problemas matemáticos diferentes y presentan características de rendimiento distintas. La seguridad de RSA reside en la dificultad computacional de la factorización de números enteros grandes, mientras que la de ECC se basa en el problema del logaritmo discreto en curvas elípticas. La principal ventaja de ECC sobre RSA es que ofrece un nivel de seguridad comparable con longitudes de clave significativamente más cortas, lo que se traduce en un menor consumo de recursos computacionales, menor ancho de banda y mayor velocidad, haciéndola ideal para sistemas embebidos, IoT y dispositivos móviles. Por ejemplo, una clave ECC de 256 bits

proporciona una seguridad equivalente a una clave RSA de 3072 bits, lo que demuestra la eficiencia de ECC para aplicaciones con limitaciones de recursos.

El algoritmo RSA es, quizás, el criptosistema asimétrico más conocido y ampliamente adoptado, y su fortaleza radica en un problema matemático que ha desafiado a los expertos durante siglos: la factorización de números primos. La clave del sistema RSA se genera a partir de la elección de dos números primos muy grandes, *p* y *q*, que son mantenidos en secreto.

$$N = p * q \tag{2.5}$$

La clave pública se deriva del producto de estos dos primos, como en la ecuación 2.5. Cifrar un mensaje *M* con el exponente público e y el módulo *N* es una operación computacionalmente trivial, dado por la fórmula 2.6.

$$\text{Cifrado} \equiv M^e (\text{modN}) \tag{2.6}$$

Sin embargo, el descifrado requiere el exponente privado d, el cual está intrínsecamente ligado a los factores primos originales *p* y *q*.

La seguridad de RSA depende de que, aunque sea fácil multiplicar *p* y *q* para obtener *N*, es extremadamente difícil, para un ordenador clásico, revertir este proceso para encontrar *p* y *q* a partir de *N* cuando los números son lo suficientemente grandes. Este método es lo que hace a RSA seguro. Sin embargo, esta dificultad es un problema para la computación clásica, pero no lo será para la computación cuántica.

2.2.3. Vulnerabilidades ante la computación cuántica

La fortaleza de la criptografía asimétrica, como la que acabamos de describir para RSA, se basa en la suposición de que los problemas matemáticos subyacentes son computacionalmente difíciles o imposibles de resolver para los ordenadores clásicos. Sin embargo, la llegada de la computación cuántica representa una amenaza existencial para estos sistemas. Los ordenadores cuánticos, al operar bajo los principios de la mecánica cuántica (superposición y entrelazamiento), tienen el potencial de resolver ciertos problemas matemáticos de una manera exponencialmente más rápida que sus contrapartes clásicas. En particular, la capacidad de un futuro ordenador cuántico de gran escala pondría en jaque los fundamentos de la criptografía de clave pública.

El principal responsable de esta amenaza es el algoritmo de Shor [23], que, una vez que se implemente en un computador cuántico, será capaz de resolver el problema de la factorización de números enteros grandes y el problema del logaritmo discreto en un tiempo polinomial. Esto contrasta dramáticamente con el tiempo exponencial que consume un ordenador clásico. En la práctica, esto significa que una clave RSA de 2048 bits que requeriría miles de millones de años para ser factorizada por un superordenador clásico, podría ser factorizada por un ordenador cuántico a gran escala en cuestión de minutos u horas.

El algoritmo de Shor es un algoritmo cuántico diseñado para encontrar los factores primos de un número compuesto. Aunque su explicación completa es compleja, se puede resumir en la siguiente secuencia de pasos clásicos y cuánticos.

El primer paso, que se realiza en un ordenador clásico, es convertir el problema de la factorización de un número compuesto *N*, en un problema de búsqueda de un período de una función. Para un número *a* aleatorio y coprimo con *N*, se busca el número entero más pequeño *r* (el período) tal que:

$$a^r \equiv 1 \ (mod N) \tag{2.7}$$

Si se encuentra *r*, los factores de *N* se pueden calcular fácilmente a partir de:

$$MCM\left(a^{\frac{r}{2}}-1,N\right) y\ MCM\left(a^{\frac{r}{2}}+1,N\right) \tag{2.8}$$

El problema es que encontrar este período *r* para números grandes es tan difícil como la factorización original. Aquí es donde entra en juego la computación cuántica.

El corazón del algoritmo de Shor reside en la capacidad de un ordenador cuántico para resolver el problema de encontrar el período *r* de forma eficiente. El proceso se desarrolla así:

1. Creación de un estado de superposición: Un ordenador cuántico utiliza un registro de qubits para crear una superposición uniforme de todos los posibles valores de entrada. Esto significa que el ordenador evalúa la función 2.9 para todos los valores de *x* simultáneamente.

$$f(x) = a^x (mod N) \tag{2.9}$$

2. Medición Parcial: Se realiza una medición parcial en el registro de salida. Esta medición fuerza al estado cuántico a colapsar en un estado que contiene solo los valores de *x* que dan el mismo resultado para la función 2.9. Debido a la periodicidad de la función, estos valores de *x* estarán separados por múltiplos del período *r*.
3. La Transformada Cuántica de Fourier (QFT): La QFT se aplica al registro de qubits. Esta es la parte más poderosa del algoritmo. La QFT es capaz de identificar la frecuencia de los estados en superposición, que, en este caso, se corresponde directamente con el período *r* que se está buscando.
4. Extracción del período: La medición final del registro de qubits de la QFT proporciona una aproximación del período *r*. Con algunas correcciones y el algoritmo de Euclides extendido, se puede obtener el valor exacto de *r*.

Una vez que se ha encontrado el período *r*, se regresa al componente clásico para calcular los factores primos de *N*. El poder del algoritmo de Shor reside en que el paso de la QFT, que es el cuello de botella en un ordenador clásico. En un ordenador cuántico, esta operación se realiza en tiempo polinomial. Sin este paso, el problema de la factorización seguiría siendo irresoluble en un tiempo razonable.

El algoritmo de Shor no es una mejora incremental, es un salto cuántico en la capacidad de resolver un problema que la criptografía asimétrica ha asumido como indescifrable. Su implementación práctica, que requiere un gran número de qubits estables y una tasa de error muy baja, siendo esto el factor limitante actual. Su existencia ha impulsado la carrera para desarrollar soluciones de seguridad que resistan esta amenaza: la criptografía post-cuántica (*Post-Quantum Cryptography*, PQC).

2.2.4. Algoritmos post-cuánticos

La criptografía de clave pública actual, que se basa en la dificultad de resolver ciertos problemas matemáticos para los ordenadores clásicos, se verá comprometida con la llegada de la computación cuántica. En respuesta, el NIST ha desarrollado la PQC, un conjunto de algoritmos diseñados para resistir tanto a ataques clásicos como cuánticos. Tras un proceso de estandarización iniciado en 2016, el NIST publicó en 2024 sus primeros estándares PQC, conocidos como FIPS 203, 204 [24] y 205 [25]. Estos documentos definen los nuevos algoritmos que liderarán la transición hacia una seguridad resistente a la computación cuántica.

El FIPS 203 define el estándar para *Cryptographic Suite for Algebraic Lattices – Kyber* (CRYSTALS-Kyber), un esquema de encapsulación de claves *Key Encapsulation Mechanism* (KEM) que se basa en un problema matemático conocido como aprendizaje con *errores* o *Learning With Errors* (LWE). Kyber se ha posicionado como el candidato más prometedor para reemplazar a los mecanismos de intercambio de claves clásicos, como el de *Diffie-Hellman* (DH) y RSA, en entornos de alta criticidad, sistemas embebidos y aplicaciones con recursos limitados. La seguridad de Kyber reside en la

dificultad de resolver el problema de LWE, considerado intratable para los ordenadores cuánticos. El problema LWE se puede describir como un sistema de ecuaciones lineales que ha sido intencionalmente perturbado con un pequeño ruido aleatorio.

Este principio se utiliza para construir un KEM que asegura el intercambio de una clave simétrica. El proceso de encapsulación de claves en Kyber se simplifica en los siguientes pasos:

1. Generación de claves (receptor): El receptor genera un par de claves. La clave pública se construye a partir de una matriz aleatoria *A* y una clave secreta *s*, a la que se le añade un vector de error aleatorio. La clave privada es simplemente la clave secreta *s*.
2. Encapsular (emisor): El emisor, con la clave pública del receptor, genera una clave de sesión simétrica aleatoria *k*. Luego, la "cifra" usando un vector aleatorio y un vector de error, creando un texto cifrado. Este texto cifrado incluye un vector y un valor de error, que en esencia es la clave de sesión oculta por el ruido.
3. Extraer (receptor): El receptor usa su clave secreta para descifrar el texto cifrado. La clave secreta permite eliminar el ruido que se añadió durante la encapsulación, permitiendo al receptor recuperar la clave de sesión original *k*.

El FIPS 204 estandariza el *Cryptographic Suite for Algebraic Lattices – Dilithium* (CRYSTALS-Dilithium), un algoritmo de firma digital que se basa en las propiedades de las retículas estructuradas. Este algoritmo ofrece un equilibrio ideal entre el tamaño de la clave, la velocidad de firma y verificación, y una robustez probada contra los futuros ataques cuánticos. Sus características lo hacen perfecto para aplicaciones que requieren autenticar la procedencia de la información de manera rápida y eficiente, como en la firma de software y en las comunicaciones seguras.

Por su parte, el FIPS 205 establece *Stateless Practical Hash-based Incredibly Nice Cryptographic Signature Plus* (SPHINCS+), un esquema de firma digital que se basa en un concepto criptográfico diferente: las funciones hash seguras. La principal fortaleza de SPHINCS+ reside en que no necesita mantener un "estado" interno, lo que simplifica su implementación y lo hace muy resistente a los errores que podrían comprometer la seguridad. Su fuerte base teórica y su resistencia a fallos de implementación lo convierten en una opción extremadamente confiable para entornos de alta criticidad, sirviendo como una alternativa sólida a los algoritmos basados en retículas.

A diferencia de las propuestas de criptografía cuántica, estos estándares no requieren hardware cuántico para su implementación. Su enfoque es matemático, diseñando algoritmos que resistan el poder de cálculo de los futuros ordenadores cuánticos, utilizando la tecnología clásica existente.

2.3. Análisis comparativo y plataformas de prueba

2.3.1. Importancia de medición de rendimiento en sistemas reales

La transición de los algoritmos de PQC del laboratorio a su implementación en la práctica, especialmente en sistemas embebidos, exige una rigurosa medición de rendimiento. A diferencia de los análisis teóricos, que se centran en la complejidad asintótica, el rendimiento en un sistema real se mide en términos tangibles: tiempo de ejecución, consumo de memoria *Random Access Memory* (RAM) y tamaño del código en memoria *Read Only Memory* (ROM). Para un sistema embebido, estos parámetros son cruciales, ya que determinan si un algoritmo es viable para estos dispositivos.

El análisis comparativo permite identificar cuellos de botella y optimizar la implementación. Un algoritmo que es teóricamente eficiente podría no serlo en la práctica si su implementación requiere una gran cantidad de operaciones costosas o si genera una carga de trabajo que el hardware no puede manejar. Las mediciones empíricas proporcionan datos necesarios para tomar decisiones de diseño, como la elección entre diferentes algoritmos PQC o la optimización de parámetros específicos (por ejemplo, el nivel de seguridad o el tamaño de la clave).

En un sistema embebido, cada byte de memoria y cada ciclo de reloj son valiosos. Un algoritmo de PQC que no ha sido optimizado podría consumir demasiada energía, reducir la vida útil de la batería o ser demasiado lento para aplicaciones en tiempo real. Por ello, la medición de rendimiento no es solo una actividad de evaluación del sistema, sino que es un paso fundamental para asegurar la viabilidad, la eficiencia y la seguridad de los sistemas embebidos en entornos no seguros.

2.3.2. Diferencias entre arquitecturas

La elección de la arquitectura del procesador es un factor crítico en el diseño de sistemas, especialmente en el contexto de la computación embebida y la optimización de algoritmos post-cuánticos. Esta decisión se basa en dos tipos de arquitecturas; las *Complex Instruction Set Computer* (CISC) y las *Reduced Instruction Set Computer* (RISC).

Por un lado, la filosofía CISC utiliza un conjunto de instrucciones muy amplio y complejo, donde cada instrucción puede ejecutar múltiples operaciones de bajo nivel (como cargar datos, realizar una operación aritmética y almacenar el resultado) con un solo comando. Esto simplifica la programación a nivel de lenguaje ensamblador, pero a cambio hace que la lógica del procesador sea más compleja, lo que se traduce en un mayor número de transistores, un consumo energético más elevado y, a menudo, una menor eficiencia.

Por otro lado, las arquitecturas RISC persiguen la simplicidad y la eficiencia. Emplea un conjunto de instrucciones reducido, con cada instrucción realizando una única operación simple. Si bien esto requiere más líneas de código para completar una tarea, el diseño simplificado del procesador permite que las instrucciones se ejecuten muy rápidamente, a menudo en un solo ciclo de reloj. Este enfoque resulta en procesadores más pequeños, con menor consumo de energía y con menos disipación de potencia, lo que los hace ideales para entornos donde la eficiencia es prioritaria.

Las arquitecturas x64 y *Advance RISC Machine* (ARM) son los máximos exponentes de los paradigmas CISC y RISC, respectivamente, y su predominio en diferentes mercados refleja sus filosofías de diseño.

La arquitectura x64, desarrollada originalmente por Intel y AMD, domina el mercado de los ordenadores personales, servidores y estaciones de trabajo. Su robustez y retrocompatibilidad la hacen perfecta para ejecutar software complejo y exigente, donde el rendimiento es la prioridad. Sin embargo, su complejidad de diseño conlleva un mayor consumo de energía y una menor eficiencia, lo que limita su uso en dispositivos con baterías.

En contraposición, la arquitectura ARM ha conquistado el mercado de los sistemas embebidos, dispositivos móviles, y ahora está expandiéndose a los servidores de alta eficiencia. Su diseño RISC se traduce en un consumo de energía extremadamente bajo y un excelente rendimiento por vatio, lo que la convierte en la opción ideal para dispositivos con recursos limitados como la Raspberry Pi. Para los sistemas embebidos, la optimización de algoritmos para un procesador ARM no se centra tanto en el rendimiento bruto, sino en reducir el consumo de recursos, minimizar las operaciones y adaptar el código a su conjunto de instrucciones simplificado.

2.3.3. Herramientas y métodos de medición

La medición precisa del rendimiento de los algoritmos criptográficos en sistemas embebidos y, en general, en cualquier sistema, requiere el uso de herramientas especializadas que permitan la cuantificación a nivel hardware. La evaluación no se limita únicamente al tiempo total de ejecución, sino que también considera métricas de bajo nivel, como el número de ciclos de reloj, que proporcionan una visión más granular y precisa de la eficiencia del procesador.

Para la medición de los ciclos de reloj, se emplean diferentes métodos dependiendo de la arquitectura del procesador. En arquitecturas *x64*, se utiliza comúnmente la instrucción *Read Time-Stamp Counter* (RDTSC). Esta instrucción lee un contador interno del procesador que se incrementa con cada ciclo de reloj, permitiendo obtener una medida muy precisa del tiempo de ejecución de un bloque de código.

Por otro lado, en arquitecturas ARM, la medición de ciclos se realiza a través de mecanismos como *perf_event*, una interfaz del kernel de Linux [26] que permite acceder a los contadores de rendimiento del procesador. A diferencia de RDTSC, que es una instrucción simple, *perf_event* es una API que ofrece una gran flexibilidad para medir una variedad de eventos, incluidos los ciclos de reloj, instrucciones ejecutadas o errores de caché. Esta herramienta proporciona un medio robusto y estandarizado para la obtención de métricas de rendimiento en procesadores ARM.

Una vez que se han obtenido los datos de rendimiento, el análisis se realiza a través de librerías de programación. Python se ha consolidado como un lenguaje de elección para el análisis de datos debido a su vasta colección de librerías [27], entre las que destacan *Pandas, SciPyK* y *Matplotlib*.

Pandas es una librería fundamental para la manipulación y el análisis de datos, que permite la lectura y gestión de archivos de texto estructurados (como *txt* o *csv*), organizando los datos en estructuras similares a hojas de cálculo, conocidas como *DataFrames* (DF). Una vez cargados los datos, *Pandas* simplifica el cálculo de estadísticas descriptivas, como la media, la mediana y los percentiles, que son esenciales para resumir y comprender el comportamiento de los algoritmos. Esta librería proporciona una base sólida para el análisis de grandes conjuntos de datos de rendimiento, permitiendo limpiar y manipular la información de manera eficiente y legible para su posterior uso.

Para el análisis estadístico y el modelado de distribuciones de datos, se emplea *SciPy*, que ofrece funciones avanzadas como la estimación de la densidad del kernel gaussiano *gaussian_kde()* para la creación de gráficos de funciones de densidad de probabilidad. Esta función permite visualizar la forma y la concentración de los datos de rendimiento de manera continua, complementando el análisis discreto de los histogramas. Por su parte, la librería *NumPy* proporciona el soporte para operaciones matemáticas eficientes con grandes arrays y matrices, que son la base de los cálculos estadísticos en los que se fundamenta este tipo de análisis.

Finalmente, la visualización de los resultados es un paso crucial para una comprensión clara. Las librerías *Matplotlib* y *Seaborn* permiten crear gráficos de alta calidad. Mientras que *Matplotlib* ofrece un control detallado sobre cada aspecto de la figura, *Seaborn* proporciona una interfaz de alto nivel para crear visualizaciones estadísticas como los histogramas, facilitando la presentación de los resultados de forma legible e intuitiva. El uso conjunto de estas librerías permite un flujo de trabajo completo, desde la recolección de datos hasta la generación de conclusiones respaldadas visualmente a través de los gráficos.

3. Desarrollo del proyecto

3.1. Planteamiento inicial y entorno de pruebas

3.1.1. Planteamiento inicial y objetivos operativos

El punto de partida de este proyecto surge en base a una necesidad evidente: proteger las comunicaciones entre sistemas embebidos (como las Raspberry Pi 3 y 4) frente a ataques cuánticos. Los sistemas embebidos con dispositivos electrónicos diseñados para realizar unas funciones específicas de forma optimizada. Estos dispositivos son utilizados en diversos ámbitos, desde la automoción hasta dispositivos médicos. La protección de la información es crucial, debido a que suelen manejar información sensible y es vital mantener la información segura de ataques de cibercriminales.

La estandarización de ML-KEM (derivado de CRYSTALS-Kyber) en FIPS 203 abre la puerta a una generación e intercambio de claves resistente a ataques cuánticos, pero su coste computacional no es despreciable en plataformas ARM de bajo consumo. El reto central es, por tanto, encontrar una solución de compromiso entre seguridad y rendimiento con una lógica de decisión que seleccione dinámicamente la longitud de clave más adecuada al contexto de ejecución.

Con esta motivación, se fijaron los siguientes objetivos:

- Implementar un protocolo cliente-servidor que negocie capacidades de cómputo y ejecute un benchmarking integrado del módulo ML-KEM.
- Medir el coste de ciclos y tiempo de las operaciones de generación de clave, encapsulamiento y extracción.
- Decidir en función de los datos obtenidos, una de las 3 posibles longitudes de clave, respetando los objetivos de latencia propuestos en base a estudios estadísticos.

3.1.2. Entornos evaluados

Inicialmente, se optó por implementar el desarrollo en *máquinas virtuales* (VM) debido a su rapidez de arranque y flexibilidad. Este entorno permitió estudiar el repositorio que ejecuta el algoritmo ML-KEM, avanzar con la integración en C e iniciar con las pruebas, pero rápidamente presentó limitaciones en los ciclos obtenidos como resultados del benchmarking.

Para que los resultados sean coherentes cabría esperar obtener resultados con reducida desviación. Sin embargo, en las VM esto no ha sucedido, dado que el hipervisor introduce variabilidad en la planificación de hilos, los relojes de alto nivel carecen de invariancia temporal y el acceso a contadores de hardware está restringido o emulado. Como consecuencia, los resultados temporales presentaban ruido no controlable y no eran comparables entre ejecuciones.

En el código 1, se aprecia que los resultados de las pruebas en VM, confirmando que no son coherentes, porque los valores de ciclos mínimos, máximos y promedios están muy dispersos y no guardan una relación estable entre sí. A modo de ejemplo, en KEM512 la generación de claves muestra un promedio cercano a 177.000 ciclos, pero al mismo tiempo se reporta un máximo de más de $21 \cdot 10^6$ de ciclos, una diferencia desproporcionada. Algo similar ocurre en otras operaciones, donde los máximos son decenas de veces mayores que los mínimos, y esto arrastra los promedios lejos de lo que cabría esperar en una ejecución consistente. Al ejecutar la prueba de rendimiento dentro de una VM, el hipervisor no tiene el control exclusivo sobre el hardware, lo que resulta en métricas que no reflejan el comportamiento real.

```
kem512 tests passed

=== KEM512 Performance Statistics (1000 iterations) ===
Operation    | Min Cycles  | Max Cycles  | Avg Cycles
-------------|-------------|-------------|-------------
KeyGen       |       90444 |    21103936 |      177515
Encaps       |       93220 |    16531710 |      146029
Decaps       |      110418 |     9414428 |      162912
Total        |      298160 |    21439120 |      486456

kem768 tests passed

=== KEM768 Performance Statistics (1000 iterations) ===
Operation    | Min Cycles  | Max Cycles  | Avg Cycles
-------------|-------------|-------------|-------------
KeyGen       |      143944 |    20628668 |      206452
Encaps       |      141842 |     6455984 |      181131
Decaps       |      165190 |    35031636 |      241258
Total        |      453344 |    35346638 |      628842

kem1024 tests passed

=== KEM1024 Performance Statistics (1000 iterations) ===
Operation    | Min Cycles  | Max Cycles  | Avg Cycles
-------------|-------------|-------------|-------------
KeyGen       |      209492 |     5753100 |      273058
Encaps       |      214056 |     7696020 |      265687
Decaps       |      240512 |     9111072 |      321261
Total        |      664060 |     9658210 |      860008

all tests passed

=== Total Program Performance ===
Total execution cycles: 1984401468
Total iterations: 1000 x 3 KEM types = 3000 operations
Average cycles per complete KEM operation: 661467
```

Código 1: Resultados VM

Para superar estas limitaciones de rendimiento, se migró a hardware físico, en este caso, a las Raspberry Pi 3 y 4. En la figura 3 se muestra una imagen con las Raspberry Pi 3 y 4.

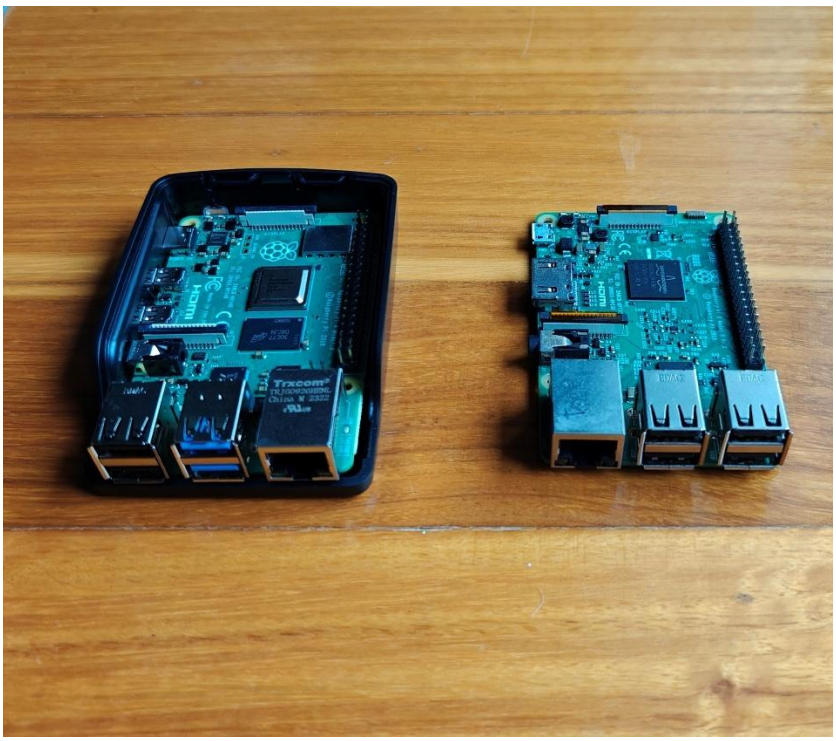


Figura 3: Raspberry Pi 4 y 3 respectivamente

Se seleccionaron Raspberry Pi 3 y 4, representativas de plataformas ARM de bajo consumo, y un PC *x64* como referencia de escritorio. La Raspberry permite acceso a las unidades de monitoreo de rendimiento o PMU (*Performance Monitoring Units*) del ARM para contar ciclos con precisión mediante *perf_event*, mientras que en *x64* es posible usar la instrucción RDTSC. Esta transición fue clave para obtener medidas estables y replicables, y para fundamentar la toma de decisiones sobre la longitud de clave.

3.1.3. Software y dependencias

El proyecto se ha implementado en C estándar, compilado con *gcc* en Linux, y Python. El lenguaje C se ha utilizado para el benchmarking, ejecutar el ML-KEM y para generar el cliente-servidor utilizado en los sistemas embebidos para concretar la longitud de clave adecuada. Para la medición de rendimiento se han empleado mecanismos de bajo nivel y alta precisión: *perf_event* en ARM (a través de llamadas de sistema para configurar los contadores de ciclos) y *rdtsc* en arquitecturas RISC *x64*. Complementariamente, se han desarrollado scripts de automatización en Python para lanzar baterías de pruebas, consolidar resultados en CSV y generar gráficos y tablas que facilitan el análisis.

El punto de partida para el desarrollo fue el repositorio "*fips203ipd*" de *pablotron*, una implementación del FIPS 203 en el lenguaje de programación C, siendo esta uno de los pilares de este proyecto. La estructura resultante mantiene la organización original del código mientras incorpora los nuevos componentes desarrollados:

- "*client_challenge.c*" y "*server_challenge.c*": implementación del protocolo de negociación desarrollado específicamente para este proyecto.
- "*main_raspberry.c*` y "*main_rdtsc.c*": variantes optimizadas para medición de ciclos en ARM y x64 respectivamente.
- "*scripts/*": utilidades Python para procesamiento de datos, generación de gráficos y análisis estadístico.
- "*output/*" y "*resultados/*": directorios para almacenamiento de datos experimentales y artefactos generados.

El repositorio incluye varios *Makefiles* y scripts de compilación que provienen del código base original. En este trabajo se han utilizado para compilar y ejecutar las pruebas de rendimiento.

Para facilitar el desarrollo y las pruebas en las Raspberry Pi se configuró acceso remoto mediante SSH, lo que permitió trabajar de forma más cómoda desde el entorno principal de desarrollo. Esta configuración resultó especialmente útil para la transferencia de archivos, la ejecución remota de scripts de benchmarking y la monitorización en tiempo real de los experimentos sin necesidad de conectar periféricos adicionales a las placas. El acceso SSH también facilitó la automatización de pruebas mediante scripts que podían ejecutarse de forma desatendida, registrando resultados en archivos CSV para su posterior análisis.

3.2. Implementación del análisis comparativo

3.2.1. Metodología experimental

El diseño experimental se enfoca en obtener mediciones precisas y reproducibles del rendimiento de ML-KEM en las Raspberry Pi 3 y 4, que se toman como muestra de sistemas embebidos. Se implementaron dos variantes de medición específicas para cada arquitectura: "*main_raspberry.c*" para ARM utilizando "*perf_event*", y "*main_rdtsc.c*" para *x64* usando RDTSC.

Para garantizar la consistencia de las mediciones, se ejecutan 100.000 iteraciones de cada operación criptográfica (keygen, encaps, decaps) por cada nivel de seguridad (512, 768, 1024 bits). El proceso incluye 50 ejecuciones de "calentamiento" inicial para estabilizar el sistema antes de comenzar las mediciones reales.

Los resultados se almacenan en archivos de texto plano, que posteriormente sirvieron para hacer un análisis estadístico de ciclos, como cálculo de medias, varianzas, histogramas y funciones de densidad de probabilidad. Esta información permite calcular métricas como percentiles 95 que fundamentan la lógica de decisión de longitud de clave.

3.2.2. Módulo criptográfico ML-KEM

El módulo criptográfico ML-KEM utilizado, esta sacado del repositorio de GitHub "*fips203ipd*" creado por el usuario usuario *pablotron*. Este módulo implementa las tres operaciones fundamentales que constituyen la base de las mediciones: generación de pares de claves *keygen*, encapsulación de material criptográfico usando la clave pública del destinatario *encaps* y desencapsulación para recuperar dicho material mediante la clave privada correspondiente *decaps*. La implementación cumple con las especificaciones oficiales del estándar FIPS 203 en cuanto a dimensiones de claves, textos cifrados y claves compartidas resultantes, utilizando fuentes de entropía del sistema Linux para garantizar la calidad aleatoria requerida.

Para permitir mediciones precisas y estadísticamente significativas, el código itera cada test 100.000 veces, de tal forma que se pueden extraer datos fiables y consistentes para su posterior análisis. El módulo registra tanto los ciclos consumidos por operación individual como los totales acumulados, proporcionando la granularidad necesaria para el análisis detallado del rendimiento.

Los tres niveles de seguridad disponibles (512, 768 y 1024 bits) se corresponden con la longitud de clave, asociada a una seguridad especifica. A mayor longitud de clave, mayor coste computacional y de memoria, lo que justifica la necesidad de un mecanismo de decisión adaptativo que seleccione el nivel óptimo según las capacidades de la plataforma.

3.2.3. Medición de ciclos por arquitectura

En sistemas ARM no siempre existe una instrucción accesible por usuario equivalente a RDTSC en *x64*. En su lugar, se accede al PMU del procesador, que expone contadores de eventos como ciclos de *Central Processing Unit* (CPU), instrucciones ejecutadas, fallos de caché, etc.

Linux proporciona la interfaz *perf_event_open* para interactuar con el PMU. El código 2 ilustra la implementación de un contador de ciclos de reloj utilizando la interfaz *perf_event*, que es la forma estándar y soportada en Linux para ARM, mientras que el acceso directo al PMU requiere privilegios adicionales. La primera función, *perf_event_open*, es una envoltura para la llamada al sistema que permite configurar y abrir un contador de rendimiento. En este caso, se especifica el tipo de evento de hardware (*PERF_TYPE_HARDWARE*) y, de manera más precisa, los ciclos de CPU (*PERF_COUNT_HW_CPU_CYCLES*). La función *init_perf_cycles* se encarga de la configuración inicial. Utiliza *perf_event_open* para crear un descriptor de archivo para el contador, lo configura para ignorar los ciclos del kernel y el hipervisor (*exclude_kernel, exclude_hv*), y lo deja deshabilitado (*disabled = 1*). Posteriormente, las llamadas *ioctl* reinician y habilitan el contador. Finalmente, la función *read_cycles* lee el valor del contador directamente del descriptor de archivo, proporcionando una medida precisa del número de ciclos de reloj consumidos por el código que se está ejecutando.

```c
// Syscall para perf_event_open
static long perf_event_open(struct perf_event_attr *hw_event, pid_t pid,
                            int cpu, int group_fd, unsigned long flags) {
    return syscall(__NR_perf_event_open, hw_event, pid, cpu, group_fd, flags);
}

// Initialize perf_event for cycle counting
static void init_perf_cycles() {
    struct perf_event_attr pe;
    memset(&pe, 0, sizeof(pe));

    pe.type = PERF_TYPE_HARDWARE;
    pe.size = sizeof(pe);
    pe.config = PERF_COUNT_HW_CPU_CYCLES;
    pe.disabled = 1;
    pe.exclude_kernel = 1;
    pe.exclude_hv = 1;

    perf_fd = perf_event_open(&pe, 0, -1, -1, 0);
    if (perf_fd == -1) {
        printf("ERROR: perf_event no disponible. No se pueden contar ciclos exactos.\n");
        exit(1);
    } else {
        ioctl(perf_fd, PERF_EVENT_IOC_RESET, 0);
        ioctl(perf_fd, PERF_EVENT_IOC_ENABLE, 0);
        printf("[ PMU ] Contador hardware via perf_event habilitado (EXACTO)\n");
    }
}

// Read cycles using perf_event
static inline uint64_t read_cycles() {
    uint64_t cycles;
    if (read(perf_fd, &cycles, sizeof(cycles)) == sizeof(cycles)) {
        return cycles;
    }
    // Si falla la lectura, terminar el programa
    printf("ERROR: No se pudieron leer los ciclos de perf_event\n");
    exit(1);
}
```

Código 2:Implementacino de medición de ciclos con perf_event

Por otro lado, el código 3 y 4 muestran la implementación de medición de ciclos. En *x64* se utiliza la instrucción *__rdtsc()* que lee directamente el *Time-Stamp Counter* (TSC) del procesador, un contador de ciclos de reloj. El programa implementa una función *read_cycles()* que simplemente ejecuta *return __rdtsc();* para obtener el valor actual del contador. Para medir una operación criptográfica se lee el TSC antes de ejecutar la función *keygen, encaps o decaps*, se ejecuta la operación, y después se vuelve a leer el TSC. La diferencia entre ambos valores proporciona exactamente el número de ciclos consumidos por la operación. Esta medición es directa y no requiere configuración especial del sistema ni permisos adicionales. A continuación, se expone el código que mide los ciclos con esta función.

```c
// Simple cycle counter using x86 intrinsics
static inline uint64_t read_cycles() {
    return __rdtsc();
}
```

Código 3: Función para leer contador

```c
// Measure total program execution cycles
uint64_t program_start = read_cycles();

run_kem512_tests();
run_kem768_tests();
run_kem1024_tests();

uint64_t program_end = read_cycles();
uint64_t total_program_cycles = program_end - program_start;
```

Código 4: Obtención de ciclos totales

3.2.4. Programas desarrollados

A continuación, se presentan los programas desarrollados para leer los ciclos y analizar los datos detallando en cada uno de ellos sus principales características y funciones más relevantes. En cuanto a los archivos encargados de contabilizar los ciclos de reloj, se ha propuesto un programa distinto para cada arquitectura, de forma que se debe conocer la arquitectura del sistema antes de la medición.

Para las arquitecturas ARM, como el caso de las Raspberry Pi 3 y 4 utilizadas en este proyecto, se utilizó el archivo *main_raspberry.c*. En el código 2 anteriormente mencionado, se exponen las distintas funciones utilizadas para medir los ciclos. Este código inicializa *perf_event* para el contador de ciclos de CPU mediante *perf_event_open()*, configurando un descriptor que monitorea específicamente *PERF_COUNT_HW_CPU_CYCLES*. Para cada medición, el descriptor habilita el contador con *ioctl(perf_fd, PERF_EVENT_IOC_ENABLE*), ejecuta la operación criptográfica, deshabilita el contador con *ioctl(perf_fd, PERF_EVENT_IOC_DISABLE*), y lee los ciclos acumulados con *read()*. Se ejecutan 100.000 iteraciones por nivel de seguridad y se generan estadísticas completas con mínimo, máximo y promedio de ciclos para su posterior análisis.

Por otro lado, de cara a medir los ciclos en arquitecturas *x64*, el archivo *main_rdtsc.c*, con código fuente fundamental ilustrado en los códigos 3 y 4, utiliza la función *__rdtsc()* de forma directa para medir ciclos. Implementa una función *read_cycles()* que simplemente retorna *__rdtsc().* Para cada medición lee el contador antes de la operación, ejecuta la función criptográfica *(keygen, encaps o decaps)*, y vuelve a leer el contador para calcular la diferencia. Ejecuta 100.000 iteraciones por nivel de seguridad y genera las mismas estadísticas de rendimiento que la versión ARM.

3.2.5. Configuración del sistema

Para las mediciones en Raspberry Pi es necesario compilar el proyecto mediante el archivo *Makefile,* incluido en el repositorio base que implementa el FIPS203. La metodología empleada consiste en sustituir manualmente el contenido del archivo *main.c* según la arquitectura objetivo, ya que este es el archivo que el *Makefile* está configurado para compilar por defecto.

3.2.6. Automatización y scripts

Para el procesamiento de datos experimentales se desarrollaron 2 scripts específicos en Python, de cara a analizar los resultados obtenidos en las Raspberry Pi 3 y 4, y otro programa destinado a la comparación de las Raspberry Pi para un mismo nivel de seguridad.

El programa analyze_cycles.py utiliza las librerías *pandas* para el manejo eficiente de grandes cantidades de datos, *numpy* para cálculos estadísticos optimizados, y *matplotlib/seaborn* para generación de gráficos. El programa procesa archivos CSV con los ciclos acumulados por operación criptográfica completa.

Implementa análisis estadístico mediante *numpy.percentile()* para calcular percentiles como el 95 y el 99, funciones estadísticas nativas de *pandas* para métricas descriptivas (media, mediana, desviación típica, varianza), y *scipy.stats.gaussian_kde* para estimación de función de densidad de probabilidad mediante Kernel Density Estimation. Las visualizaciones se generan usando *seaborn.histplot()* para histogramas y *matplotlib.pyplot* para gráficos de funciones densidad de probabilidad. Todos los datos se exportan en archivos CSV y en figuras.

Por otro lado, el programa *compare_cycles.py* genera un análisis comparativo entre plataformas. Este script extiende el análisis individual implementando comparaciones cruzadas mediante *matplotlib* para superposición de distribuciones y *seaborn* para visualizaciones comparativas multiplataforma. Utiliza la misma infraestructura de *pandas/numpy* para procesamiento de datos, pero implementa bucles iterativos que procesan simultáneamente archivos de las Raspberry Pi3 y 4 para cada nivel KEM. El script automatiza la generación de reportes mediante escritura programática de archivos Markdown, creando tablas estructuradas que leen las estadísticas previamente calculadas usando *csv.reader()* y calculan diferencias absolutas entre plataformas, asi como imágenes comparativas.

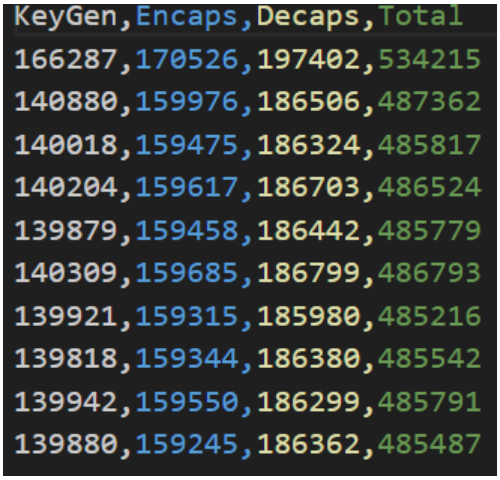
Estos scripts permiten realizar un eficiente análisis de datos, reproducible para cualquier escenario, siempre y cuando los datos de entrada a los programas estén formateados correctamente.

3.3. Análisis estadístico

De cara a ilustrar los análisis estadísticos llevados a cabo, se ejemplificará el proceso con la Raspberry Pi 4 y el KEM768. El análisis llevado a cabo para las distintas máquinas y niveles de seguridad es el mismo, generando asi los mismos archivos finales y conclusiones.

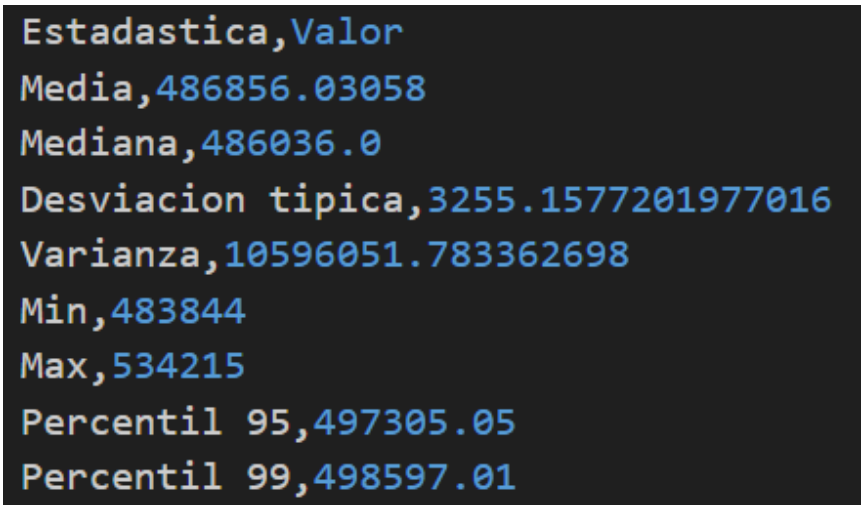
3.3.1. Métricas calculadas

A partir de entrada en formato CSV ilustrados en la figura 4, se calculan métricas descriptivas: mínimo, máximo, media, mediana, desviación típica y varianza. Estas métricas se guardan en un archivo CSV tal y como se muestra en la figura 5.



```
KeyGen,Encaps,Decaps,Total
166287,170526,197402,534215
140880,159976,186506,487362
140018,159475,186324,485817
140204,159617,186703,486524
139879,159458,186442,485779
140309,159685,186799,486793
139921,159315,185980,485216
139818,159344,186380,485542
139942,159550,186299,485791
139880,159245,186362,485487
```

Figura 4: Datos de entrada (número de ciclos)



```
Estadastica,Valor
Media,486856.03058
Mediana,486036.0
Desviacion tipica,3255.1577201977016
Varianza,10596051.783362698
Min,483844
Max,534215
Percentil 95,497305.05
Percentil 99,498597.01
```

Figura 5: Análisis estadístico Raspberry Pi 4 con KEM768

Se presta especial atención al percentil 95 como estimador conservador del rendimiento en demanda. La conversión de ciclos a tiempo (ms) se realiza con la expresión 3.1.

$$\text{tiempo\_ms} = (\#\text{ciclos} / \text{frecuencia\_Hz}) \times 1000 \qquad (3.1)$$

3.3.2. Histogramas y funciones de densidad de probabilidad

La visualización mediante histogramas permite identificar la forma de la distribución y la presencia de colas. El script genera visualizaciones detalladas como se muestra en la figura 6 donde se observa la distribución de ciclos para KEM768 en Raspberry Pi 4, con un histograma que revela la concentración de mediciones alrededor de 485.000 ciclos y la función de densidad de probabilidad superpuesta que muestra en la figura 7 la forma de la distribución.

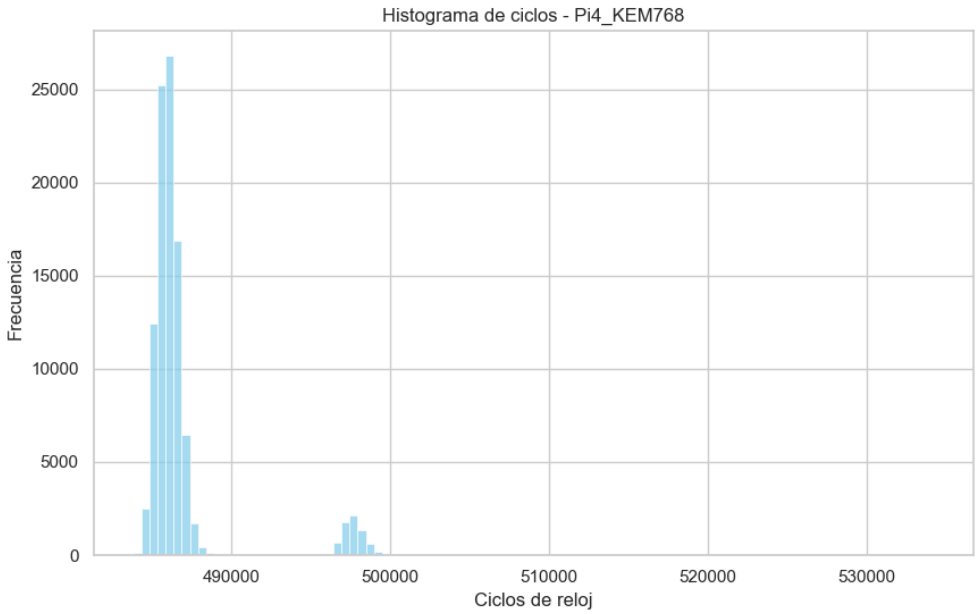


Figura 6: Histograma para Raspberry Pi 5 con KEM768

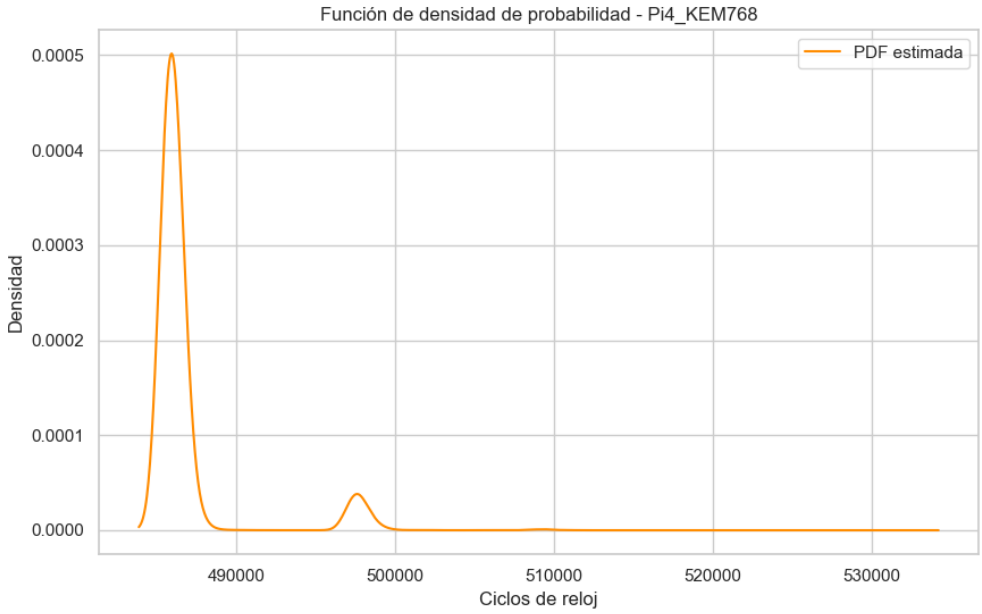


Figura 7: Densidad de probabilidad Raspberry Pi 4 con KEM768

3.3.3. Comparativa de rendimiento

A través del programa *compare_cycles.py* se realiza una comparación de rendimiento entre Raspberry Pi 3 y Raspberry Pi 4 para el nivel de seguridad KEM768, utilizando mediciones de 100.000 iteraciones por plataforma obtenidas con la API *perf_event* del kernel Linux, utilizada en el programa *analyze_cycles.py*. La figura 8 presenta los histogramas superpuestos de las distribuciones de ciclos de ambas plataformas, donde la Raspberry Pi 3 aparece en azul y la Raspberry Pi 4 en naranja, permitiendo la comparación visual directa de los ciclos de reloj y la frecuencia de estos. La figura 8 evidencia claramente la mejora de rendimiento de la Raspberry Pi 4 (distribución desplazada hacia la izquierda, menor número de ciclos) respecto a la Raspberry Pi 3, y mostrando también la menor variabilidad en la Raspberry Pi 4 mediante una distribución más estrecha.

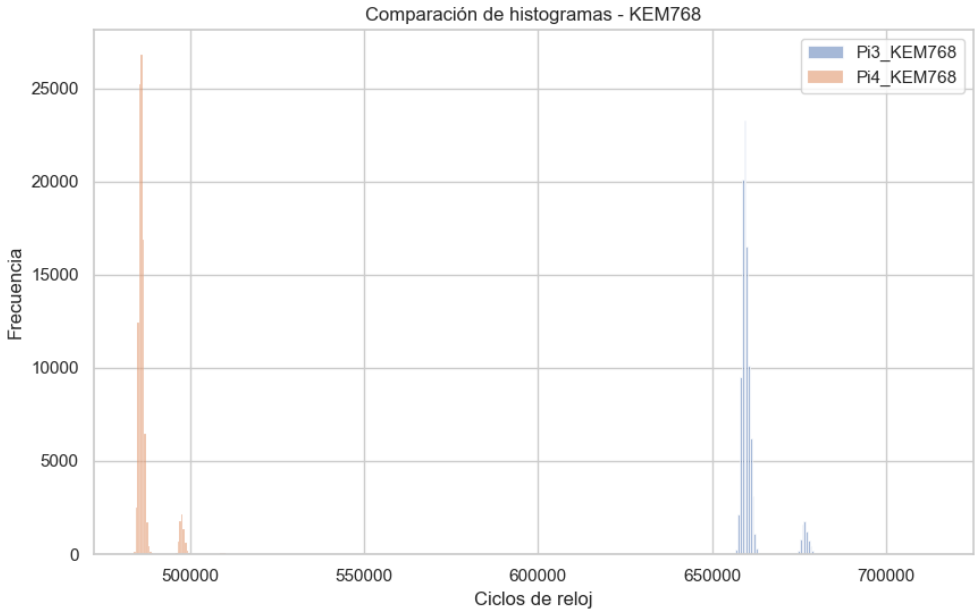


Figura 8: Comparación de histogramas para KEM768

3.3.4. Interpretación de los resultados

Los resultados del análisis comparativo se documentan automáticamente en un documento *Markdown* llamado *comparativa_estadisticas.md*, un reporte estructurado que presenta tablas con factores de escalado entre plataformas, diferencias absolutas en ciclos, y métricas de mejora de rendimiento que cuantifican, por ejemplo, que la Raspberry Pi 4 ejecuta KEM768 aproximadamente 1.36 veces más rápido que la Raspberry Pi 3, o lo que es lo mismo, la Raspberry Pi 3 es 0.73 veces más lenta respecto a la versión 4, como se puede comprobar en la figura 9.

KEM768

Estadistica	Pi3	Pi4	Diferencia
Media	660853.55	486856.03	-173997.52
Mediana	659538.00	486036.00	-173502.00
Desviacion estandar	4750.31	3255.16	-1495.15
Varianza	22565442.65	10596051.78	-11969390.87
Min	656231.00	483844.00	-172387.00
Max	713655.00	534215.00	-179440.00
Percentil 95	676222.00	497305.05	-178916.95
Percentil 99	678001.01	498597.01	-179404.00

Figura 9: Comparativa estadística con KEM 768

Los datos generados por estos scripts son fundamentales para el algoritmo de decisión de longitud de clave, ya que los factores de escalado calculados permiten estimar el rendimiento de operaciones criptográficas en tiempo real antes de ejecutarlas, optimizando la selección de parámetros según los recursos disponibles y los umbrales de latencia establecidos.

3.4. Diseño e implementación del protocolo cliente-servidor

Para llevar a ejecución los resultados obtenidos, se genera un protocolo cliente-servidor programado en el lenguaje de programación C, con la idea de probar qué nivel de seguridad se podría usar entre los dos modelos de Raspberry Pi sobre los que se han trabajado, de cara a comunicarse con garantías de seguridad frente a un ataque con basado en la computación cuántica.

3.4.1. Arquitectura general

La arquitectura del protocolo cliente servidor se apoya en cuatro componentes principales:

- Cliente: inicia la sesión, solicita un desafío de rendimiento y decide la longitud de clave.
- Servidor: ejecuta el algoritmo de ML-KEM y devuelve métricas de ejecución.
- Módulo benchmarking: mide en el servidor, los ciclos consumidos por las operaciones criptográficas.
- Lógica de decisión: Situada en el cliente. Interpreta las métricas del servidor y selecciona la longitud de clave (512, 768 o 1024) conforme a los criterios establecidos.

En la figura 10, se expone el diagrama de secuencia del protocolo cliente-servidor creado para la selección de longitud de clave.

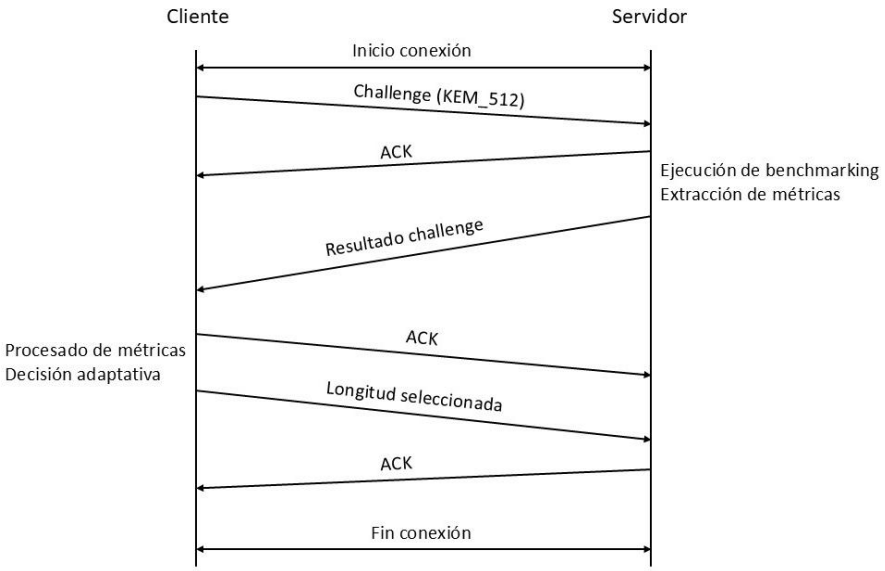


Figura 10: Diagrama de secuencia del protocolo

El establecimiento inicial de la comunicación requiere que el servidor configure un socket de escucha con las funciones *(socket(), bind(), listen())* en el puerto 8.000, permaneciendo en estado de espera hasta recibir conexiones entrantes. Por su parte, el cliente crea un socket TCP con la función *socket ()* e inicia la conexión hacia el servidor mediante la función *connect().* El flujo de datos está sincronizado mediante confirmaciones explícitas para evitar bloqueos: el cliente envía un mensaje de "*challenge*"; el servidor confirma recepción con un ACK y, a continuación, remite el resultado del benchmark. Tras recibirlo, el cliente envía un ACK del resultado, calcula el tiempo estimado a partir de los ciclos y la frecuencia efectiva y decide la longitud de clave. Esta decisión se comunica al servidor, que la confirma con un último ACK. Con esta secuencia, ambos extremos comparten un estado consistente y el canal puede cerrarse limpiamente.

3.4.2. Requisitos funcionales y no funcionales

Los requisitos funcionales abordan un intercambio de mensajes con estructuras de datos definidas como en la figura 11, la ejecución de un análisis integrado que devuelva métricas de ciclos, frecuencia y arquitectura, y una decisión fundamentada de la longitud de clave con notificación explícita al servidor.

```c
struct header {
    unsigned char packetType;
    unsigned char id;
    unsigned char len;
};

struct challenge_data {
    struct header header_info;
    int longitud_clave;
};

struct result_data {
    struct header header_info;
    uint64_t ciclos;
    double frecuencia_ghz;
    char arquitectura[64];
    int longitud_final;
};
```

Figura 11: Estructuras de paquete

En el plano no funcional, se aborda la latencia acotada (por ejemplo, por debajo del percentil 95 obtenido del análisis de resultados de Raspberry Pi 3 y 4), robustez ante fallos y portabilidad entre ARM y x64.

3.4.3. Modelado de mensajes y estructuras

El protocolo utiliza mensajes con una estructura común que facilita el intercambio ordenado de información, como se mostró en la figura 11. Cada mensaje incluye una cabecera fija *struct header* con tres campos: *packetType* para identificar el tipo de mensaje (1=CHALLENGE, 2=CONFIRMACION, 3=RESULTADO), *id* para correlacionar peticiones y respuestas, y *len* para validar la integridad del mensaje recibido.

Los mensajes CHALLENGE (*struct challenge_data*) incorporan un campo *longitud_clave* que especifica el nivel de seguridad a evaluar.

Los mensajes RESULTADO (*struct result_data*) contienen las métricas del benchmark, esto incluye los ciclos con el número exacto medido, el campo

frecuencia_ghz con la frecuencia del procesador, y la arquitectura con una descripción del hardware. Adicionalmente se añade el campo *longitud_final* para intercambiar la longitud de clave acordada.

Los mensajes *CONFIRMACION* actúan como reconocimientos (ACK) y utilizan únicamente la cabecera.
Para garantizar la portabilidad entre arquitecturas se valida que la longitud recibida coincide con la esperada, detectando errores de transmisión.

3.4.4. Implementación y sincronización

El protocolo implementa una secuencia de confirmaciones para evitar bloqueos:

1. El cliente envía un *challenge* solicitando los datos necesarios.
2. El servidor responde con un ACK de recepción.
3. El servidor envía el resultado con ciclos, frecuencia y arquitectura.
4. El cliente confirma con un ACK de resultado.
5. El cliente envía la longitud de clave decidida.
6. El servidor devuelve un ACK final confirmando la recepción.

Con ello, ambos extremos comparten un estado común y pueden cerrar la conexión de forma ordenada. En la figura 12, muestra un ejemplo del uso de acuses de recibo (*ACK*) en la comunicación cliente-servidor mediante sockets.

```c
// Recibir resultado del challenge (incluyendo longitud_final decidida por el servidor)
recv(sockfd, &resultado, sizeof(resultado), 0);
if (resultado.header_info.packetType == RESULTADO){
    printf("\n--- RESULTADO DEL CHALLENGE ---\n");
    printf("Ciclos de reloj: %llu\n", (unsigned long long)resultado.ciclos);
    printf("Frecuencia: %.2f GHz\n", resultado.frecuencia_ghz);
    printf("Arquitectura: %s\n", resultado.arquitectura);
    printf("Longitud de clave final decidida por el servidor: %d bits\n", resultado.longitud_final);
} else {
    printf("Respuesta inesperada del servidor\n");
}

// Enviar ACK de resultado recibido
struct header ack_resultado;
ack_resultado.packetType = CONFIRMACION;
ack_resultado.id = resultado.header_info.id;
ack_resultado.len = sizeof(ack_resultado);
send(sockfd, &ack_resultado, sizeof(ack_resultado), 0);
printf("ACK de resultado enviado al servidor.\n");
```

Figura12: Recepción de datos y envió de ACK

Tras recibir el resultado del *challenge* con la función *recv()*, el cliente comprueba que el paquete recibido corresponde efectivamente a un mensaje de tipo *RESULTADO*. Si es así, se procesan los campos enviados por el servidor (como ciclos de reloj, frecuencia, arquitectura o la longitud de clave final) y se imprimen en pantalla. Una vez validada esta información, el cliente construye un mensaje de confirmación, asignando al campo *packetType* el valor *CONFIRMACION* y copiando en id el identificador del mensaje original recibido. Este mensaje se envía al servidor con *send()*, utilizando el valor 0 en el campo de *flags* para indicar un envío estándar sin opciones adicionales. De esta forma, el ACK garantiza que el servidor sabe que el resultado del *challenge* ha sido correctamente recibido y procesado, cerrando así el ciclo de comunicación fiable entre ambas partes.

3.5. Algoritmo de decisión de longitud de clave

3.5.1. Objetivo y criterios

El objetivo práctico es garantizar que el tiempo total de establecimiento de clave y sesión permanezca por debajo de un umbral operativo establecido sin sacrificar más seguridad de la necesaria. La decisión equilibra latencia y robustez frente a atacantes, escalando a 768 o 1024 bits de longitud de clave solo cuando el dispositivo y la carga lo permitan.

3.5.2. Cálculo de decisión

Para calcular los umbrales y garantizar seguridad sin perder rendimiento del sistema se optó por utilizar el percentil 95 de las mediciones obtenidas en concreto sobre el nivel de seguridad más bajo. Esta decisión hace que, al ejecutar el nivel de seguridad más bajo, se ejecute un *challenge* con menor peso computacional, lo que resulta en una menor carga para el sistema y velocidad en la decisión de nivel de seguridad.

El percentil 95 se calcula para cada dispositivo y nivel de seguridad, asegurando que el 95% de las ejecuciones estén por debajo del umbral. En la figura 13 se muestran los percentiles para todos los casos posibles.

```
### Raspberry Pi 3 (1.4 GHz)
- KEM512: Percentil 95 = 395,322 ciclos → 395,322 / 1,400,000,000 ≈ 0.282 ms
- KEM768: Percentil 95 = 676,222 ciclos → 676,222 / 1,400,000,000 ≈ 0.483 ms
- KEM1024: Percentil 95 = 994,179 ciclos → 994,179 / 1,400,000,000 ≈ 0.710 ms

### Raspberry Pi 4 (1.5 GHz)
- KEM512: Percentil 95 = 293,978 ciclos → 293,978 / 1,500,000,000 ≈ 0.196 ms
- KEM768: Percentil 95 = 497,305 ciclos → 497,305 / 1,500,000,000 ≈ 0.332 ms
- KEM1024: Percentil 95 = 732,171 ciclos → 732,171 / 1,500,000,000 ≈ 0.488 ms
```

Figura 13: Percentiles 95

Estos resultados muestran que ambos dispositivos permiten seleccionar la longitud máxima (1024 bits) sin comprometer el rendimiento. Este análisis se puede extrapolar a otros sistemas embebidos con menor capacidad, seleccionando así en otro nivel de seguridad. En este caso, las Raspberry Pi son dispositivos que, aunque no sean muy potentes, pueden realizar este tipo de operaciones sin mucho coste computacional.

En la figura 14, se muestra el fragmento de código del servidor que implementa esta lógica. Los umbrales de 0.5 y 0.3 ms están basados en los resultados anteriormente comentados para garantizar una selección precisa y eficiente. Este enfoque asegura que la decisión sea consistente con los datos experimentales y los objetivos de rendimiento.

```
// Decidir longitud de clave final según p95 (umbrales expresados en milisegundos)
const double umbral_1024_s = 0.0003; // p95 <= 0.3 ms → 1024
const double umbral_768_s  = 0.0005; // 0.3 ms < p95 <= 0.5 ms → 768

if (tiempo_s < umbral_1024_s) {
    resultado.longitud_final = 1024;
} else if (tiempo_s < umbral_768_s) {
    resultado.longitud_final = 768;
} else {
    resultado.longitud_final = 512;
}
printf("Longitud de clave final decidida: %d bits\n", resultado.longitud_final);
```

Figura 14: Selección de nivel de seguridad

3.5.3. Justificación estadística y umbrales

Los umbrales seleccionados para decidir la longitud de clave final se basan en el análisis estadístico de los tiempos de ejecución mencionados anteriormente. El umbral de 0.3 ms se establece como el límite superior para seleccionar la longitud de clave más alta (1024 bits), garantizando que solo dispositivos con un rendimiento suficientemente alto puedan manejar esta configuración sin comprometer la experiencia del usuario.

El umbral intermedio de 0.5 ms permite seleccionar una longitud de clave de 768 bits, ofreciendo un equilibrio entre seguridad y rendimiento para dispositivos con capacidades moderadas.

Finalmente, cualquier tiempo superior a 0.5 ms conduce a la selección de la longitud mínima de 512 bits, asegurando la viabilidad operativa incluso en dispositivos con recursos limitados.

3.6. Validación y pruebas

En la implementación desarrollada, los umbrales empleados en el cliente se definieron a partir de un análisis estadístico de los resultados obtenidos en las plataformas Raspberry Pi 3 y Raspberry Pi 4, utilizando el percentil 95 como se ha mencionado anteriormente. En las figuras 15 y 16 se muestra el resultado de la ejecución del protocolo, actuando la Raspberry Pi 3 como servidor y la Raspberry Pi 4 como cliente.

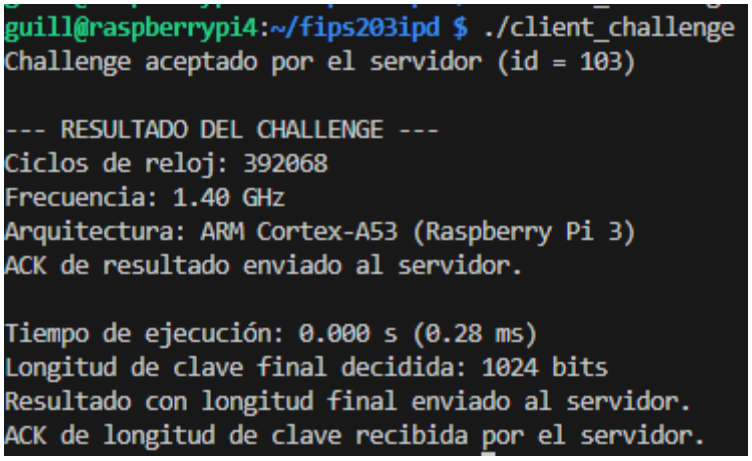


Figura 15: Raspberry Pi 4 como cliente

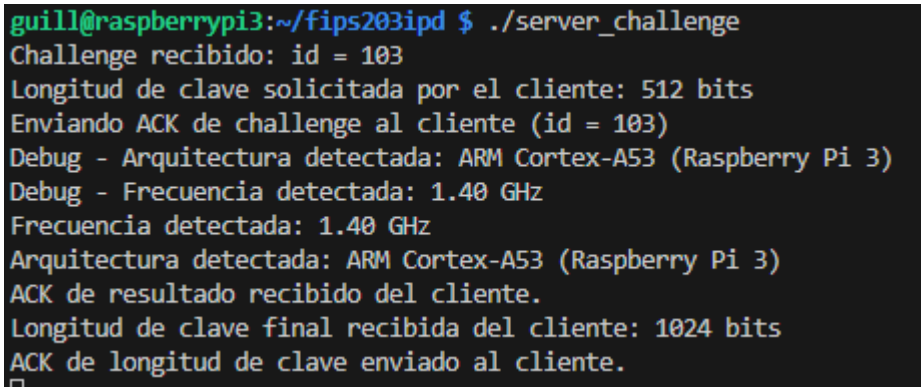


Figura 16: Raspberry Pi 3 como servidor

En este escenario, aunque el servidor cuenta con menos recursos de cómputo, el cliente tiene mayor capacidad y es quien aplica los umbrales previamente definidos y determinan el nivel de seguridad a emplear. En este caso, el resultado de la prueba determinó utilizar el nivel de seguridad más alto, dado por una longitud de clave de 1024. Esta elección viene dada por el resultado de la prueba, realizada en 0.28 ms cuando el umbral para bajar a un nivel más bajo de seguridad es de 0.3 ms. Este diseño experimental demuestra la flexibilidad del enfoque; la lógica de selección no depende del rol servidor/cliente, sino de la capacidad del dispositivo que realiza la evaluación y aplica los criterios en función de una posible sobrecarga en el servidor. En consecuencia, aunque esta arquitectura no responde al estándar habitual de los protocolos de seguridad en producción, ofrece un marco útil para experimentar con la adaptación dinámica de parámetros criptográficos, mostrando cómo la decisión basada en métricas empíricas permite equilibrar seguridad y eficiencia en sistemas de recursos limitados.

4. Conclusiones

El recorrido de este proyecto ha permitido explorar de manera integral la implementación de un sistema criptográfico resistente a ataques cuánticos en plataformas embebidas de bajo consumo, concretamente Raspberry Pi 3 y 4. Además, durante el desarrollo de este trabajo se refleja cómo la teoría y la práctica convergen para abordar los desafíos de la criptografía en la era cuántica. Lo que comenzó como un análisis de algoritmos y protocolos, se ha materializado en un sistema funcional de negociación de claves, capaz de adaptarse dinámicamente a las capacidades de los dispositivos y a las restricciones de tiempo de ejecución.

La implementación del protocolo cliente-servidor, junto con el análisis comparativo integrado, han permitido explorar las limitaciones y posibilidades de plataformas como Raspberry Pi y sistemas x64. Los resultados obtenidos demuestran que es posible equilibrar seguridad y rendimiento mediante decisiones fundamentadas en datos experimentales, utilizando métricas robustas como el percentil 95 para garantizar la consistencia y la fiabilidad del sistema.

Sin embargo, este trabajo también pone de manifiesto las limitaciones actuales. La dependencia de hardware específico y la necesidad de calibraciones precisas para cada plataforma subrayan la importancia de seguir investigando en soluciones más

generalizadas y escalables. Además, aunque el sistema es funcional en entornos controlados, su despliegue en escenarios reales requerirá abordar desafíos adicionales, como la variabilidad en las prestaciones de las redes y la interoperabilidad con otros sistemas criptográficos.

De cara al futuro, las líneas de desarrollo en criptografía post-cuántica deben centrarse en tres aspectos clave; la optimización de los algoritmos para reducir aún más los tiempos de ejecución, la ampliación de la compatibilidad con una mayor variedad de dispositivos y arquitecturas, y la integración en infraestructuras de comunicación existentes. Estos avances permitirán que los sistemas de negociación de claves resistentes a ataques cuánticos evolucionen desde implementaciones experimentales hacia soluciones prácticas y escalables para las comunicaciones del futuro.

En última instancia, este trabajo no solo contribuye al avance técnico, sino que también refuerza la importancia de la convergencia entre la teoría y la práctica, entre la criptografía y la ingeniería, para construir sistemas y entornos que sean tanto seguros como eficientes en un mundo cada vez más conectado y amenazado por la computación cuántica.

5. Lista de acrónimos

Wi-Fi: Wireless Fidelity

IoT: Internet of Things

NIST: National Institute of Standards and Technology

FIPS: Federal Information Processing Standards

ARPANET: Advanced Research Projects Agency Network

ML-KEM: Module Lattice-Based Key-Encapsulation Mechanism

VM: Virtual Machine

PMU: Performance Monitoring Units

ARM: Advance RISC Machine

RISC: Reduced Instruction Set Computer

CISC: Complex Instruction Set Computer

ACK: Acknowledgement

TSC: Time Stamp Counter

DES: Data Encryption Standard

AES: Advanced Encryption Standard

TLS: Transport Layer Security

RSA: Rivest-Shamir-Adleman

ECC: Elliptic Curve Cryptography

QFT: Transformada Cuántica de Fourier

PQC: Criptografía Post-Cuántica

KEM: Key Encapsulation Mechanism

CRYSTALS-Kyber: Cryptographic Suite for Algebraic Lattices-Keyber

PMU: Performance Monitoring Unit

CPU: Central Processing Unit

DH: Diffie-Hellman

LWE: Learning With Errors

RAM: Random Access Memory

ROM: Read Only Memory

RDTSC: Read Time-Stamp Counter

DF: Data Frame

CRYSTALS-Dilithium: Cryptographic Suite for Algebraic Lattices – Dilithium

6. Bibliografía

[1] T. Edison, Ed., *Appletons' Encyclopaedia of Applied Mechanics*, 1879, s.v. "The Telegraph." [Online]. Available: http://www.telegraph-history.org/edison/appletons/index.html. [Accessed: Jul. 22, 2025].

[2] J. G. Cruz and F. P. Corbacho, "La 'Compañía Trasatlántica Española', pionera de las radiocomunicaciones marítimas españolas: 'Siempre adelante'," *Llull: Revista de la Sociedad Española de Historia de las Ciencias y de las Técnicas*, vol. 37, no. 80, pp. 13–43, 2014.

[3] A. P. Yuste, "El primer cable sin hilos que cruzó el Atlántico," in *Historia de las ciencias y de las técnicas*, Universidad de La Rioja, 2004, pp. 787–794.

[4] E. S. Grosvenor and M. Wesson, *Alexander Graham Bell*. New Word City, 2016. [Online]. Available: https://acortar.link/Ave8In [Accessed: Jul. 22, 2025].

[5] C. Paloque-Bergès and V. Schafer, "Arpanet (1969–2019)," *Internet Histories*, vol. 3, no. 1, pp. 1–14, 2019. [Online]. Available: https://doi.org/10.1080/24701475.2018.1560921. [Accessed: Jul. 22, 2025].

[6] N. Al-Falahy and O. Y. Alani, "Technologies for 5G Networks: Challenges and Opportunities," *IT Professional*, vol. 19, no. 1, pp. 12-20, Jan.-Feb. 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7839836, doi: 10.1109/MITP.2017.9. [Accessed: Jul. 25, 2025].

[7] P. Gokhale, O. Bhat, and S. Bhat, "Introduction to IOT," *International Advanced Research Journal in Science, Engineering and Technology*, vol. 5, no. 1, pp. 41-44, 2018. [Online]. Available: https://acortar.link/nHZLge [Accessed: Jul. 25, 2025].

[8] H. Y. Wong, "Shor's Algorithm," in *Introduction to Quantum Computing*, Cham: Springer, 2024. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-36985-8_29, doi: 10.1007/978-3-031-36985-8_29. [Accessed: Jul. 22, 2025]

[9] L. Chen, L. Chen, S. Jordan, Y. K. Liu, D. Moody, R. Peralta, … & D. Smith-Tone, *Report on Post-Quantum Cryptography*, vol. 12. Gaithersburg, MD, USA: US Department of Commerce, National Institute of Standards and Technology, 2016. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf?ref=suddo.de [Accessed: Jul. 25, 2025].

[10] National Institute of Standards and Technology (NIST), *Module-Lattice-Based Key-Encapsulation Mechanism Standard*, Federal Information Processing Standards Publication (FIPS) NIST FIPS 203, Aug. 13, 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203 [Accessed: Jul. 26, 2025]

[11] M. Jammer, *The Conceptual Development of Quantum Mechanics*, 2nd ed. New York, NY, USA: American Institute of Physics, 1989.

[12] M. Planck, "Ueber das Gesetz der Energieverteilung im Normalspectrum," *Annalen der Physik*, vol. 309, no. 3, pp. 553–563, 1901.

[13] A. Einstein, "Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt," *Annalen der Physik*, vol. 322, no. 6, pp. 132–148, 1905.

[14] E. Schrödinger, "An Undulatory Theory of the Mechanics of Atoms and Molecules," *Physical Review*, vol. 28, no. 6, pp. 1049–1070, 1926.

[15] P. García González and J. J. García Sanz, *Física cuántica I*. Madrid, Spain: UNED – Universidad Nacional de Educación a Distancia, 2014. [Online]. Available: https://elibro.net/es/ereader/unican/48765?page=30 [Accessed: Mar. 26, 2025]

[16] G. Gamow, *Biografía de la Física*. Barcelona, Spain: Salvat Editores, 1985.

[17] A. Einstein, B. Podolsky, and N. Rosen, "Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?," *Physical Review*, vol. 47, no. 10, pp. 777–780, 1935.

[18] M. A. Nielsen y I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[19] J. Preskill, "Quantum Computing in the NISQ Era and Beyond," *Quantum*, vol. 2, p. 79, 2018.

[20] W. H. Zurek, "Decoherence and the transition from quantum to classical," *Physics Today*, vol. 44, no. 10, pp. 36–44, 1991.

[21] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*. Berlin, Germany: Springer, 2002.

[22] D. Mahto and D. Yadav, "RSA and ECC: A comparative analysis," *International Journal of Applied Engineering Research*, vol. 12, pp. 9053–9061, 2017.

[23] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Foundations of Computer Science (FOCS)*, Santa Fe, NM, USA, Nov. 1994, pp. 124–134. doi: 10.1109/SFCS.1994.365700.

[24] National Institute of Standards and Technology (NIST), *FIPS 204: ML-DSA — Lattice-Based Digital Signature Standard*. Gaithersburg, MD, USA: NIST, Aug. 13, 2024. [Online]. Available: https://csrc.nist.gov/pubs/fips/204/final [Accessed: Mar. 28, 2025]

[25] National Institute of Standards and Technology (NIST), *FIPS 205: SLH-DSA — Stateless Hash-Based Digital Signature Standard*. Gaithersburg, MD, USA: NIST, Aug. 13, 2024. [Online]. Available: https://csrc.nist.gov/pubs/fips/205/final [Accessed: Mar. 28, 2025]

[26] The Linux Foundation, *Perf Events*. [Online]. Available: https://perfwiki.github.io/main/ [Accessed: Aug. 03, 2025]

[27] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2017.