XVII Jornadas de Ingeniería Telemática
JITEL 2025
Universidad de Extremadura

Actas de las XVII Jornadas de Ingeniería Telemática (JITEL 2025), Cáceres (España), 12-14 de noviembre de 2025.

Análisis de Algoritmos Post-Cuánticos de Cifrado en Plataformas Heterogéneas

José Alberto Benito-Corral*, Jorge E. López de Vergara*, Luis de Pedro*, Iván González*, Florina Almenares†, Luis Cruz-Piris‡, Andrés Marín§

*Dept. Tecnología Electrónica y de las Comunicaciones, Universidad Autónoma de Madrid
†Dept. Ingeniería Telemática, Universidad Carlos III de Madrid
‡Dept. Automática, Universidad de Alcalá
§Dept. Ing. Sistemas Telemáticos, Universidad Politécnica de Madrid
josea.benito@estudiante.uam.es, {jorge.lopez_vergara, luis.depedro, ivan.gonzalez}@uam.es, florina@it.uc3m.es, luis.cruz@uah.es, andres.mlopez@upm.es

En este artículo se presenta un estudio exhaustivo del rendimiento de algoritmos criptográficos post-cuánticos estandarizados por el NIST (Kyber/ML-KEM, Dilithium/ML-DSA, FALCON/FN-DSA y SPHINCS+/SLH-DSA), comparándolos con RSA y ECDSA. Se evaluaron tiempos de generación de claves, encapsulación/desencapsulación, firma/verificación y consumo de CPU/memoria en las plataformas de: Ordenador PC, Raspberry Pi 5 y teléfono móvil Android. Se utilizaron distintos lenguajes de programación en las dos primeras plataformas (C, Java, C#, Python) y Kotlin en la tercera. Los resultados indican que las implementaciones en C optimizado con extensiones vectoriales ofrecen el mejor rendimiento, destacando a Kyber y Dilithium como opciones prácticas en sistemas reales, con resultados comparables a los que se obtienen en algoritmos de cifrado clásicos.

***Palabras Clave*—Criptografía post-cuántica, Kyber, Dilithium, FALCON, SPHINCS+, RSA, ECDSA, rendimiento, recursos, plataformas, lenguajes de programación.**

I. Introducción

La inminente aparición de ordenadores cuánticos, capaces de ejecutar el algoritmo de Shor en tiempo polinómico, es decir, con una eficiencia mucho mayor que los actuales algoritmos clásicos, pone en jaque la criptografía clásica basada en la factorización (RSA) y el logaritmo discreto (curvas elípticas, ECDSA) [1]. Para garantizar la confidencialidad y autenticidad de la información a largo plazo, la comunidad científica y los organismos de estandarización (NIST), están promoviendo esquemas de criptografía post–cuántica (PQC, *Post-Quantum Cryptography*) basados en problemas matemáticos intratables hoy en día tanto para procesadores clásicos como cuánticos.

Aunque en agosto de 2024 el NIST (*National Institute of Standards and Technology*) publicó varios estándares PQC [2], su integración en sistemas reales afronta desafíos prácticos: incompatibilidades con infraestructuras heredadas, limitaciones de hardware en dispositivos IoT y la necesidad de soporte multi-lenguaje en entornos empresariales heterogéneos. Esta brecha entre especificación y adopción motiva un análisis detallado de rendimiento y consumo de recursos, para que las organizaciones puedan planificar migraciones graduales sin comprometer su actividad durante el periodo de transición.

Por ello, este artículo presenta una evaluación práctica de los algoritmos estandarizados y a estandarizar por el NIST —ML-KEM (Kyber), ML-DSA (Dilithium), FN-DSA (FALCON) y SLH-DSA (SPHINCS+)— comparándolos con RSA-2048 y ECDSA-256 en tres plataformas heterogéneas: PC (Intel x86), Raspberry Pi 5 (ARM Cortex A76) y teléfono móvil Android. Se utiliza C, Java, C#, Python en las dos primeras plataformas y Kotlin en la tercera. Con el lenguaje C se ha probado a aplicar optimizaciones de compilación que aprovechen el hardware disponible. A partir de esta combinación de algoritmos, plataformas y lenguajes, se han medido los tiempos de generación de claves, encapsulación/decapsulación, firma/verificación y consumo de CPU y memoria.

Las contribuciones de este trabajo son:

- Comparar el rendimiento de esquemas post–cuánticos frente a RSA y ECDSA en operaciones criptográficas.
- Analizar el uso de recursos (CPU y memoria) de estos nuevos algoritmos criptográficos en diferentes arquitecturas y lenguajes de programación.
- Ofrecer recomendaciones prácticas para la adopción de criptografía resistente a ataques cuánticos en entornos reales.

El resto del artículo se estructura de la siguiente manera: en primer lugar, se revisa el estado del arte y los trabajos relacionados; después, se describe la metodología empleada y el entorno de pruebas; a continuación, se presentan

los resultados obtenidos; y, por último, se exponen las conclusiones junto con las principales líneas de trabajo futuro.

II. Estado del Arte

En esta sección, se muestran estudios y trabajos relacionados. La criptografía post-cuántica ha evolucionado rápidamente en la última década para responder a la amenaza que suponen los ordenadores cuánticos a los algoritmos clásicos. El NIST inició en 2016 un proceso de estandarización que, tras la selección de cuatro esquemas principales, tres basados en retículas (Kyber para intercambio de claves, Dilithium y FALCON para firmas), y otro basado en *hashes* sin estado (SPHINCS+), culminó en 2024 con la estandarización de ML-KEM (Kyber) [3], ML-DSA (Dilithium) [4], SLH-DSA (SPHINCS+) [5] y la preparación del borrador para FN-DSA (FALCON).

Varios estudios han evaluado el rendimiento de estos algoritmos en diferentes entornos. Benny *et al*. [6] realizan un meta-análisis sobre los riesgos de la computación cuántica y la llegada de los algoritmos post-cuánticos Kyber, Dilithium, FALCON y SPHINCS+, donde examinan la seguridad, características y rendimiento de estos algoritmos y las relaciones entre tamaño de clave pública y privada así como el tiempo de cifrado. De manera similar, Iglesias Hernández *et al*. [7] realizan un análisis comparativo entre Dilithium y FALCON, donde evalúan la seguridad, eficiencia y rendimiento, comparando el tamaño de las claves y midiendo el tiempo de ejecución en la generación de claves, firma y verificación para las distintas versiones de estos algoritmos.

En ámbitos más específicos, Vidaković y Miličević [8] analizan el desempeño y la aplicabilidad de Dilithium, FALCON y SPHINCS+ en entornos con recursos limitados, como IoT, tarjetas inteligentes, blockchain y comunicación entre vehículos (V2V). Se evalúa el equilibrio entre seguridad (tamaño de clave pública y firma), eficiencia computacional y consumo de recursos, destacando las ventajas y limitaciones de cada algoritmo según su utilización. Fitzgibbon y Octtaviani [9] presentan un estudio de rendimiento de algoritmos KEM: Kyber, HQC, McElliece y BIKE, y de firma digital: Dilithium, FALCON y SPHINCS+, para los niveles de seguridad 3 y 5, en distintos dispositivos: un PC, un portátil y una Raspberry Pi 4. Se usa la librería `liboqs` y demuestra de nuevo, que los algoritmos basados en retículas son los más eficientes. Abassi *et al*. [10] analizan Kyber, NTRU, BIKE, Dilithium y FALCON, en tres entornos distintos: servidor, portátil y un dispositivo equivalente a una Raspberry Pi 3B+, utilizando también `liboqs` e implementaciones originales de los autores. Los resultados confirman que los algoritmos muestran un rendimiento similar, siendo BIKE el de mayor consumo.

Este artículo presenta similitudes con los trabajos previos para analizar los algoritmos post-cuánticos, pero se distingue de estos por su análisis simultáneo a lo largo de tres ejes: algoritmos, arquitecturas y lenguajes. Ofrece así una visión más completa, evaluando PC, SBC (*Single Board Computer*) y teléfono móvil con C, Java, C#, Python y Kotlin, proporcionando tiempos de operación y uso de CPU y memoria de las distintas variantes de los algoritmos en todas estas plataformas y lenguajes.

III. Metodología

En esta sección se detalla el comportamiento de los esquemas post-cuánticos en entornos reales. Para ello se diseñó un banco de pruebas basado en la ejecución repetida de operaciones criptográficas y la monitorización de consumo de recursos. A continuación se describen los principales elementos de la metodología empleada.

Sistemas de prueba. Se emplearon tres plataformas:

- **PC:** Procesador AMD Ryzen 5 5500U (6 núcleos / 12 hilos, hasta 4.0 GHz), 16 GB RAM, Ubuntu 24.04.2 LTS, GCC 13.3.0, .NET SDK 8.0.115, Java 21.0.2 y Python 3.12.3.
- **SBC (Raspberry Pi 5):** Procesador ARM Cortex-A76 con 4 núcleos y 64 bits, 2.4 GHz, 8 GB RAM, Debian GNU/Linux 12, GCC 12.2.0, .NET SDK 8.0.405, Java 17.0.13 y Python 3.11.2.
- **Móvil Android:** Procesador MediaTek Dimensity 700 (ARM Cortex-A76 + A55) con 8 núcleos, 2.2 GHz, 8 GB RAM, Android 13, Kotlin 1.9.24.

Bibliotecas criptográficas. Se seleccionaron implementaciones oficiales o de referencia y se emplearon las siguientes bibliotecas:

- *C:* `OpenSSL` (RSA, ECDSA y Kyber (NEON en Raspberry PI 5) y código de referencia en C para Kyber, Dilithium, SPHINCS+ (pq-crystals) y FALCON (benchmarks NIST-PQC Round 3).
- *Java:* `bouncycastle.pqc.crypto` (SPHINCS+, Dilithium, FALCON), `bouncycastle.pqc.jcajce` (Kyber) y `bouncycastle.jce` (RSA, ECDSA).
- *Python:* `cryptography.hazmat` (RSA, ECDSA)
- *C#:* `bouncycastle.pqc.crypto` (PQC), `System.Security.Cryptography` (RSA, ECDSA).
- *Kotlin:* `bouncycastle.pqc.jcajce` para todos los esquemas post-cuánticos.

Adaptación del código. A partir de las versiones anteriores, se realizó una instrumentación en cada lenguaje para medir de forma precisa los tiempos que supone la generación de claves, la encapsulación/desencapsulación (intercambio de claves) o la firma/verificación (firmas digitales). El código fuente, disponible en GitHub[1], incluye las modificaciones necesarias para obtener estas métricas.

Obtención de uso de recursos. El consumo de CPU y memoria se registró con:

- `/usr/bin/time -v` en PC y SBC, obteniendo tiempo de ejecución, uso máximo de memoria residente (RSS) y porcentaje de CPU.
- Android Studio Profiler en el móvil, midiendo memoria y CPU tras activar depuración USB y perfilado.

Procedimiento. Para cada operación criptográfica:

[1] https://github.com/Alberto1-2/TFG_CODE

1. Se ejecutaron 1000 iteraciones (100 en móvil). Se calcularon valores promedio y desviación típica de tiempos (ms), CPU (%) y memoria (MB).

Este diseño asegura comparaciones homogéneas y minimiza ruido de medición, proporcionando datos fiables sobre el coste real para la integración algoritmos PQC.

IV. Resultados

En esta sección se presentan los resultados de las pruebas que se muestran en la Tabla I con los tiempos medios de operación (generación + encapsulación/desencapsulación para intercambio de claves, y generación + firma + verificación para firmas) para PC y Raspberry: con C (con optimizaciones AVX2 y NEON y FPU), C#, Java, Python; y en Android: Kotlin.

Para Kyber (512, 768 y 1024 bits), Python ofrece los peores tiempos: 7.9 ms a 17.7 ms en PC y de 11.1 ms a 25 ms en Raspberry Pi 5, siendo hasta 300 veces más lento respecto a C con AVX2. En cambio, Java y C# ofrecen resultados mucho más competitivos, con valores del submilisegundo en todas las variantes, alcanzando entre 0.18 y 0.4 ms en PC y 0.3 a 0.65 ms en Raspberry. La Fig. 1 muestra la comparativa de tiempos para Kyber 512.

En Dilithium (niveles 2, 3 y 5), de forma similar, Python presenta tiempos elevados, (≈42–73 ms) en PC y hasta 137 ms en Raspberry, mientras que Java y C# se sitúan entre 0.7 y 1.5 ms y entre 1.3 y 3.1 ms en ambos entornos, con degradaciones que oscilan entre 5 y hasta más de 300 veces (python) frente a C (PC). Kotlin ofrece cifras similares, con valores (≈6–14 ms). La Fig. 2 muestra la comparativa de tiempos para Dilithium 2.

Para FALCON 512 y 1024, Python resulta impráctico en PC (más de 10 s por iteración en FALCON 1024) y en Raspberry (1.7-20 s), mientras que Java y C# reducen esos tiempos a 11–25 ms (512 bits) y 28–68 ms (1024 bits). La Fig. 3 muestra la comparativa de tiempos para FALCON 512.

SPHINCS+ en su esquema de firma SHA2-128f (*fast robust*) alcanza en PC 7.9 ms (C), hasta 117 ms (C#), 107 ms (Java) y 67 ms (Python). En Raspberry Pi 5, llega a 40 ms (C), mientras que en C#, Java y Python alcanza 205 ms, 145 ms y 83 ms. En Android, Kotlin obtiene 391 ms, descartándolo para aplicaciones interactivas. La Fig. 4 muestra la comparativa de tiempos para SPHINCS+ SHA2-128f.

El uso de memoria en C es bastante bajo en relación con el resto de lenguajes, en un orden de magnitud similar entre PC y Raspberry, de unidades; en Python del orden de decenas, y en Java, C# y Kotlin del orden de centenas de MBytes. En uso CPU, Java y C# se superan el 100 % de uso, lo que indica que se aprovecha la existencia de varios cores en el procesador. C y Python muestran un uso cercano al 100 %. Mientras, en Kotlin se observan usos de CPU que alcanzan hasta el 200 %.

En algoritmos clásicos, RSA-2048 es mucho más lento, debido a la generación de clave: sobre 181 ms (C) en PC, y similares en C# y Java (183-143 ms), y 43 ms en Python. En Raspberry Pi 5, supera los 500 ms en todos los lenguajes. Kotlin en Android requiere 325 ms. ECDSA-256, en cambio, se mantiene por debajo de 1.2 ms en todas las plataformas y lenguajes, destacando por su eficiencia.

Estas cifras muestran que Kyber y Dilithium ofrecen un compromiso óptimo entre latencia y recursos, manteniéndose en rangos de ms incluso en hardware limitado como Raspberry Pi o Android, mientras que SPHINCS+ y RSA quedan fuera de escenarios de alto rendimiento.

V. Conclusiones

En este trabajo se ha presentado una evaluación exhaustiva de los esquemas de criptografía post-cuántica propuestos para estandarizar en el NIST (Kyber, Dilithium, FALCON y SPHINCS+) frente a los algoritmos clásicos RSA-2048 y ECDSA-256, en tres plataformas heterogéneas (PC, Raspberry Pi 5 y Android) con implementaciones con distintos lenguajes. Los resultados revelan que:

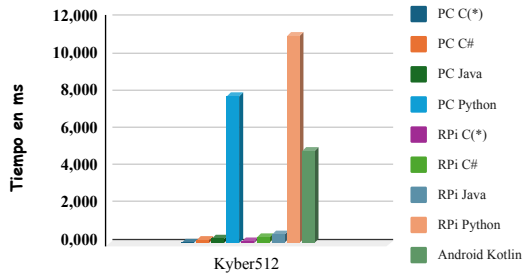

Fig. 1. Tiempos de ejecución para Kyber 512

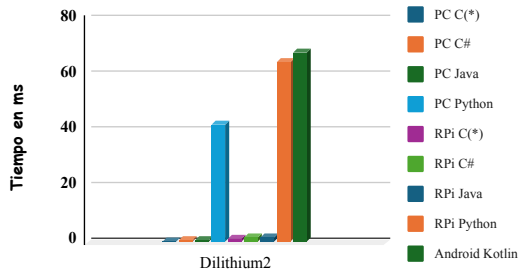

Fig. 2. Tiempos de ejecución para Dilithium 2

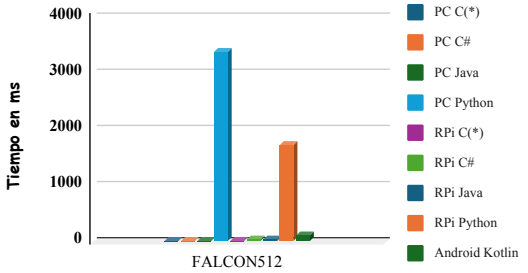

Fig. 3. Tiempos de ejecución para FALCON 512

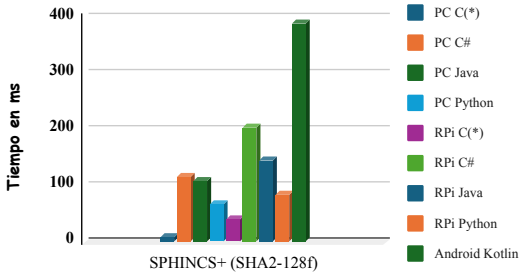

Fig. 4. Tiempos de ejecución para SPHINCS+ SHA2-128f

Tabla I
TIEMPO MEDIO TOTAL DE EJECUCIÓN Y DESVIACIÓN TÍPICA (MS) EN PC, RASPBERRY PI 5 Y ANDROID
C(*) INDICA OPTIMIZACIONES AVX2 EN C PARA PC, Y NEON Y FPU PARA RPI 5 EN C

Esquema	**PC**: C(*)	C#	Java	Python	**RPi 5**: C(*)	C#	Java	Python	**Android**: Kotlin
Intercambio de claves									
Kyber 512	0.027±0.003	0.185±0.028	0.265±0.128	7.900±0.071	0.073±0.002	0.290±0.179	0.447±0.332	11.111±0.180	4.909±7.204
Kyber 768	0.042±0.004	0.241±0.030	0.287±0.095	12.419±0.084	0.116±0.004	0.464±0.111	0.651±0.502	18.404±1.130	5.701±2.858
Kyber 1024	0.058±0.005	0.360±0.033	0.408±0.049	17.756±0.072	0.212±0.045	0.655±0.101	0.638±0.190	24.965±0.241	7.093±2.971
Firma digital									
Dilithium 2	0.124±0.039	0.678±0.312	0.776±0.268	42.254±20.532	0.810±0.359	1.329±0.631	1.345±0.542	64.722±29.253	6.810±5.265
Dilithium 3	0.203±0.062	1.186±0.559	1.244±0.481	68.721±35.142	1.316±0.631	2.336±1.135	2.160±0.913	112.230±59.474	10.624±8.095
Dilithium 5	0.283±0.066	1.543±0.554	1.555±0.475	88.771±37.510	1.649±0.555	3.128±1.185	2.806±0.919	136.899±54.810	14.086±9.285
SPHINCS+ (SHA2-128f)	7.914±0.149	116.600±0.555	106.922±8.019	66.621±5.194	40.182±0.057	204.739±10.919	145.073±9.440	83.469±0.095	391.044±25.067
FALCON 512	6.050±6.970	13.159±4.615	10.905±3.313	3.394±1.582 **seg**	10.953±2.918	25.790±9.378	22.767±6.263	1.709±1.113 **seg**	106.980±24.010
Algoritmos clásicos									
RSA-2048	181.449±96.890	183.340±100.645	146.266±93.880	43.265±25.652	648.012±329.217	646.377±271.578	538.567±347.396	241.216±111.766	325.385±143.890
ECDSA-256	0.105±0.012	0.338±0.036	0.675±0.261	0.145±0.005	0.342±0.072	1.107±0.344	1.188±0.577	0.373±0.025	0.642±0.049

- **Rendimiento.** Kyber 512 y Dilithium 2, optimizados con AVX2 en C, alcanzan latencias de intercambio de claves y firma/verificación inferiores a 0.3 ms en PC. En Raspberry Pi 5 con NEON, estos esquemas crecen entre 4 y 7 veces, y en Android se mantienen por debajo de 7 ms, superando claramente a RSA en velocidad de generación de claves y firmas. Además, con una desviación típica también mínima con valores por debajo de 5 milésimas y 7 décimas de milisegundo, respectivamente en C con AVX2, lo que proporciona una gran estabilidad de estos algoritmos en su ejecución en PC.
- **Consumo de recursos.** En Android, los esquemas post-cuánticos emplean un 140 %–200 % de CPU y 270–300 MB de memoria, exceso asumible en la mayoría de aplicaciones móviles. SPHINCS+ y FALCON presentan restricciones mayores (latencias de hasta cientos de ms) y, en el caso de SPHINCS+ es importante escoger un esquema ligero (SHA2-128f) para reducir los tiempos de proceso.
- **Portabilidad multi–lenguaje.** Las implementaciones en Java y C# ofrecen latencias submilisegundo para Kyber e inferior a los 2 ms para Dilithium, favoreciendo su integración en aplicaciones empresariales. Python resulta adecuado solo para prototipos por sus penalizaciones de hasta dos órdenes de magnitud.

De este modo, Kyber y Dilithium se perfilan como las opciones más equilibradas para una transición segura y eficiente hacia la era post-cuántica en entornos reales. A partir de los resultados obtenidos, como trabajo futuro se plantean las siguientes ideas:

- **Optimización de implementaciones:** mejorar el rendimiento de FALCON y SPHINCS+ mediante técnicas de bajo nivel y optimización de código para ampliar su accesibilidad en entornos de baja capacidad computacional.
- **Estudio en entornos reales a gran escala:** analizar el comportamiento de los algoritmos en sistemas con distinto número de iteraciones, alta concurrencia de usuarios, múltiples dispositivos y diversidad de arquitecturas, para evaluar su escalabilidad y robustez.
- **Evaluación de nuevos candidatos PQC:** aplicar la misma metodología a los esquemas de *NIST Round 4* (BIKE, Classic McEliece, HQC, o NTRU), con el fin de comparar sus características de rendimiento y eficiencia con los estándares actuales.
- **Esquemas híbridos clásico–post-cuántico:** diseñar y validar protocolos que combinen algoritmos tradicionales y post-cuánticos en un mismo flujo criptográfico, facilitando la compatibilidad y reforzando la seguridad durante la transición.

AGRADECIMIENTOS

Esta actuación ha sido financiada mediante el programa de actividades de I+D con referencia TEC-2024/COM-504 y acrónimo RAMONES-CM concedido por la Comunidad de Madrid a través de la Dirección General de Investigación e Innovación Tecnológica a través de la Orden 5696/2024.

REFERENCIAS

[1] D. J. Bernstein and T. Lange, "Post-quantum cryptography," *Nature*, vol. 549, pp. 188–194, 2017.

[2] National Institute of Standards and Technology, "Post-quantum cryptography standardization fips," https://csrc.nist.gov/pqc-standardization, NIST, Tech. Rep., 2024.

[3] ——, "FIPS 203 module-lattice-based key-encapsulation mechanism standard," https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf, NIST, Federal Information Processing Standards Publication FIPS 203, 2024.

[4] ——, "FIPS 204 module-lattice-based digital signature standard," https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf, NIST, Federal Information Processing Standards Publication FIPS 204, 2024.

[5] ——, "FIPS 205 stateless hash-based digital signature standard," https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf, NIST, Federal Information Processing Standards Publication FIPS 205, 2024.

[6] S. Benny, I. Desai, L. Uriarte, I. Tsai, and L. McMahan, "A meta-analysis on NIST post-quantum cryptographic primitive finalists," *Emerging Investigators*, 2024.

[7] E. Iglesias Hernández, L. Hernández Álvarez, L. Hernández Encinas, and J. Sánchez García, "Análisis comparativo de las firmas digitales postcuánticas basadas en retículos," in *IX Jornadas Nacionales de Investigación en Ciberseguridad (JNIC)*, 2024, pp. 404–411.

[8] M. Vidaković and K. Miličević, "Performance and applicability of post-quantum digital signature algorithms in resource-constrained environments," *Algorithms*, vol. 16, no. 11, p. 518, 2023.

[9] G. Fitzgibbon and C. Ottaviani, "Constrained device performance benchmarking with the implementation of post-quantum cryptography," *Cryptography*, vol. 8, no. 2, 2024.

[10] M. Abbasi, F. Cardoso, P. Váz, J. Silva, and P. Martin, "A practical performance benchmark of post-quantum cryptography across heterogeneous computing environments," *Cryptography*, vol. 9, no. 2, 2025.