Evaluasi Eksperimental ML-DSA dan ECDSA untuk Penandatanganan Firmware pada Perangkat IoT Berdaya Rendah

Khairunnisa Azizah - 18223117
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: khairunnisa.azizah.id@gmail.com , 18223117@std.stei.itb.ac.id

***Abstract*— Pembaruan *firmware* secara *Over-the-Air* (OTA) pada perangkat *Internet of Things* (IoT) rentan terhadap serangan injeksi *malware*. Meskipun *Elliptic Curve Digital Signature Algorithm* (ECDSA) jamak digunakan, keamanannya terancam oleh komputer kuantum. Penelitian ini melakukan evaluasi eksperimental komparatif antara ECDSA dan *Module Lattice-Based Digital Signature Algorithm* (ML-DSA) sebagai standar kriptografi pasca-kuantum (PQC) dari NIST dalam skenario penandatanganan *firmware* IoT berdaya rendah melalui 5.000 pengujian. Hasil menunjukkan adanya *trade-off* performa yang signifikan. ML-DSA menawarkan verifikasi komputasi yang efisien bagi mikrokontroler dengan waktu eksekusi 192,45 mikrodetik hingga 496,25 mikrodetik. Namun, ML-DSA memerlukan *overhead* spasial masif, ukuran kunci publik membesar hingga 40,5 kali lipat (2.592 byte) dan tanda tangan digital naik hingga 72,3 kali lipat (4.627 byte) dibanding ECDSA. Pada berkas *firmware* besar (1000 KB), *overhead* transmisi jaringan sangat minimal (di bawah 0,45%). Sebaliknya, ML-DSA tidak layak diterapkan langsung pada jaringan hemat daya ber-MTU ketat seperti LoRaWAN karena memicu fragmentasi paket pada berkas kecil (10 KB) dengan *overhead* mencapai 45,19%. Penelitian menyimpulkan ML-DSA sangat layak untuk perangkat IoT kelas menengah-atas dengan RAM minimal 64 KB demi investasi keamanan jangka panjang.**

Keywords—IoT, Secure Firmware Update, Post-Quantum Cryptography, ML-DSA, ECDSA, Benchmarking

I. PENDAHULUAN

Perkembangan Internet of Things (IoT) telah mendorong penggunaan perangkat terhubung dalam berbagai sektor, seperti industri, kesehatan, transportasi, dan rumah pintar. Sebagian besar perangkat IoT memiliki siklus hidup yang panjang sehingga memerlukan pembaruan firmware secara berkala untuk memperbaiki kerentanan keamanan, menambahkan fitur baru, maupun meningkatkan kinerja sistem. Namun, proses distribusi firmware juga menjadi salah satu target serangan siber. Penyerang dapat memodifikasi firmware selama proses distribusi atau menggantinya dengan firmware berbahaya yang mengandung malware. Oleh karena itu, diperlukan mekanisme yang mampu menjamin integritas dan keaslian firmware sebelum dipasang pada perangkat IoT.

Salah satu pendekatan yang umum digunakan untuk mengamankan proses pembaruan firmware adalah penerapan tanda tangan digital. Melalui mekanisme ini, perangkat IoT dapat memverifikasi bahwa firmware yang diterima benar-benar berasal dari pihak yang berwenang dan tidak mengalami perubahan selama proses distribusi. Saat ini, Elliptic Curve Digital Signature Algorithm (ECDSA) merupakan salah satu algoritma tanda tangan digital yang banyak digunakan karena menawarkan tingkat keamanan yang tinggi dengan ukuran kunci yang relatif kecil dan efisiensi komputasi yang baik. Akan tetapi, perkembangan teknologi komputasi kuantum menimbulkan ancaman terhadap berbagai algoritma kriptografi kunci publik yang digunakan saat ini, termasuk algoritma berbasis kurva eliptik.

Sebagai respons terhadap ancaman tersebut, National Institute of Standards and Technology (NIST) telah mengembangkan dan menstandarkan algoritma kriptografi pasca-kuantum (Post-Quantum Cryptography/PQC) yang dirancang untuk tetap aman terhadap serangan komputer kuantum. Salah satu algoritma yang telah distandardisasi adalah ML-DSA (Module Lattice-Based Digital Signature Algorithm), yang sebelumnya dikenal sebagai CRYSTALS-Dilithium. Meskipun menawarkan ketahanan terhadap serangan kuantum, ML-DSA memiliki karakteristik yang berbeda dibandingkan ECDSA, terutama pada ukuran kunci, ukuran tanda tangan, kebutuhan memori, serta performa komputasinya. Karakteristik tersebut menjadi tantangan tersendiri ketika algoritma diterapkan pada perangkat IoT yang memiliki keterbatasan sumber daya.

Berdasarkan permasalahan tersebut, penelitian ini melakukan evaluasi eksperimental terhadap algoritma ML-DSA dan ECDSA untuk skenario penandatanganan firmware pada perangkat IoT berdaya rendah. Evaluasi dilakukan dengan mengukur waktu pembangkitan kunci, waktu penandatanganan, waktu verifikasi, ukuran kunci, ukuran tanda tangan, serta penggunaan memori selama proses verifikasi. Hasil penelitian diharapkan dapat memberikan gambaran mengenai trade-off antara keamanan pasca-kuantum dan efisiensi sumber daya, sehingga dapat menjadi referensi dalam pemilihan algoritma tanda tangan digital untuk implementasi sistem pembaruan firmware yang aman pada lingkungan IoT.

II. Dasar Teori

A. *Digital Signature for Secure Firmware Update*

Firmware merupakan perangkat lunak tingkat rendah yang mengendalikan fungsi dasar suatu perangkat IoT. Karena perangkat IoT sering beroperasi dalam jangka waktu yang panjang, vendor perlu mendistribusikan pembaruan firmware secara berkala untuk memperbaiki kerentanan keamanan, memperbarui fitur, dan meningkatkan performa sistem. Namun, proses distribusi firmware juga menjadi target serangan karena penyerang dapat mengganti atau memodifikasi firmware sebelum diinstal pada perangkat.

Untuk mengatasi permasalahan tersebut, mekanisme digital signature digunakan untuk menjamin integritas dan autentisitas firmware. Pada skema ini, vendor menandatangani firmware menggunakan private key sebelum firmware didistribusikan. Selanjutnya, perangkat IoT melakukan verifikasi menggunakan public key yang telah tersimpan sebelumnya. Firmware hanya akan dipasang apabila tanda tangan digital berhasil diverifikasi [1].

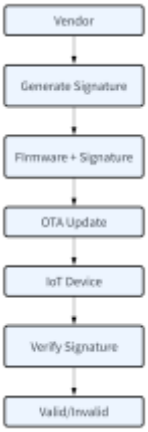

Fig. 1. Mekanisme proses distribusi firmware

Pendekatan ini telah menjadi komponen utama dalam secure boot dan secure firmware update yang digunakan pada berbagai sistem embedded modern [2].

B. *Elliptic Curve Digital Signature Algorithm (ECDSA)*

Elliptic Curve Digital Signature Algorithm (ECDSA) merupakan salah satu algoritma tanda tangan digital yang distandardisasi dalam Digital Signature Standard (DSS) oleh National Institute of Standards and Technology (NIST) melalui FIPS 186-5 [3]. ECDSA dibangun berdasarkan Elliptic Curve Discrete Logarithm Problem (ECDLP), yaitu permasalahan matematika yang hingga saat ini dianggap sulit diselesaikan secara efisien menggunakan komputer klasik. Dibandingkan algoritma tanda tangan digital lainnya, ECDSA mampu memberikan tingkat keamanan yang tinggi dengan ukuran kunci yang relatif kecil sehingga banyak digunakan pada sertifikat digital, protokol TLS, blockchain, dan perangkat IoT [3].

Secara umum, ECDSA terdiri atas tiga tahapan utama, yaitu key generation, signing, dan verification. Pada tahap key generation, pengguna menghasilkan pasangan private key dan public key berbasis kurva eliptik. Selanjutnya, private key digunakan untuk menghasilkan tanda tangan terhadap hash pesan, sedangkan public key digunakan untuk memverifikasi keaslian tanda tangan tersebut. Keunggulan ukuran kunci yang kecil membuat ECDSA menjadi salah satu algoritma yang banyak diimplementasikan pada perangkat dengan keterbatasan sumber daya.

Menurut FIPS 186-5 [3], proses penandatanganan ECDSA menghasilkan pasangan tanda tangan (r,s). Komponen s dihitung menggunakan nilai hash pesan H(m), private key d, dan bilangan acak sementara k sebagai berikut:

$$s = k^{-1}(H(m) + dr) \bmod n \tag{1}$$

di mana n merupakan orde kurva eliptik yang digunakan. Keamanan ECDSA sangat bergantung pada kerahasiaan private key serta keunikan nilai acak k pada setiap proses penandatanganan.

Meskipun memiliki performa yang baik dan telah digunakan secara luas, algoritma berbasis kurva eliptik diperkirakan rentan terhadap serangan komputer kuantum melalui algoritma Shor. Apabila komputer kuantum berskala besar berhasil direalisasikan, keamanan ECDSA dapat terancam sehingga diperlukan alternatif algoritma tanda tangan digital yang tahan terhadap ancaman kuantum [4].

C. *Module-Lattice-Based Digital Signature Algorithm (ML-DSA)*

Elliptic Module-Lattice-Based Digital Signature Algorithm (ML-DSA) merupakan algoritma tanda tangan digital pasca-kuantum yang distandardisasi oleh NIST melalui FIPS 204 pada tahun 2024 [1]. Algoritma ini sebelumnya dikenal sebagai CRYSTALS-Dilithium dan merupakan salah satu hasil utama dari program standardisasi Post-Quantum Cryptography (PQC). Berbeda dengan ECDSA yang dibangun berdasarkan kurva eliptik, ML-DSA menggunakan struktur matematika berbasis lattice yang diyakini tetap aman terhadap serangan komputer kuantum.

Keamanan ML-DSA didasarkan pada dua permasalahan matematika utama, yaitu Module Learning With Errors (Module-LWE) dan Module Short Integer Solution (Module-SIS). Hingga saat ini belum ditemukan algoritma kuantum yang mampu menyelesaikan kedua permasalahan tersebut secara efisien, sehingga ML-DSA dianggap sebagai salah satu kandidat kuat untuk menggantikan algoritma tanda tangan digital klasik pada era pasca-kuantum [1].

Secara sederhana, permasalahan Module-LWE dapat direpresentasikan sebagai:

$$b = As + e \tag{2}$$

di mana A merupakan matriks publik, s adalah secret vector, dan e merupakan error vector yang ditambahkan untuk menghasilkan kesulitan komputasional. Keberadaan error vector tersebut menyebabkan proses pencarian secret vector menjadi sangat sulit dilakukan, bahkan dengan bantuan komputer kuantum.

FIPS 204 mendefinisikan tiga tingkat keamanan utama, yaitu ML-DSA-44, ML-DSA-65, dan ML-DSA-87. Ketiga varian tersebut menawarkan trade-off yang berbeda antara tingkat keamanan, kebutuhan komputasi, ukuran kunci, dan ukuran tanda tangan digital. Semakin tinggi tingkat keamanan yang digunakan, semakin besar pula kebutuhan sumber daya yang diperlukan.

Walaupun menawarkan ketahanan terhadap ancaman kuantum, ukuran public key dan signature ML-DSA umumnya lebih besar dibandingkan ECDSA. Konsekuensinya, kebutuhan penyimpanan, bandwidth komunikasi, dan penggunaan memori pada perangkat IoT juga meningkat. Oleh karena itu, evaluasi performa ML-DSA pada perangkat berdaya rendah menjadi aspek penting sebelum algoritma tersebut dapat diadopsi secara luas pada sistem IoT generasi berikutnya [5].

TABLE I. PERBANDINGAN KARAKTERISTIK ECDSA DAN ML-DSA

Karakteristik	ECDSA	ML-DSA
Dasar Matematis	Elliptic Curve	Module Lattice
Tahan Quantum	Tidak	Ya
Ukuran Kunci	Kecil	Lebih Besar
Ukuran Signature	Kecil	Lebih Besar
Standar NIST	FIPS 186-5	FIPS 204
Cocok untuk IoT	Sangat cocok	Perlu Evaluasi

D. *Elliptic Post-Quantum Cryptography in Low-Power IoT Devices*

Transisi menuju post-quantum cryptography menjadi tantangan tersendiri pada lingkungan IoT karena sebagian besar perangkat memiliki keterbatasan sumber daya komputasi, memori, dan kapasitas penyimpanan. Oleh sebab itu, selain tingkat keamanan, efisiensi implementasi juga menjadi faktor penting dalam pemilihan algoritma kriptografi.

Abdulrahman et al. mengevaluasi implementasi CRYSTALS-Dilithium pada ARM Cortex-M4 dan menunjukkan bahwa algoritma tersebut dapat dijalankan pada perangkat embedded dengan sumber daya terbatas meskipun memerlukan penggunaan memori yang lebih tinggi dibandingkan algoritma klasik [6]. Selain itu, Kannwischer et al. mengembangkan framework PQM4 yang menjadi standar benchmarking berbagai algoritma post-quantum cryptography pada ARM Cortex-M4 dan banyak digunakan dalam penelitian terkait implementasi PQC pada perangkat IoT [7].

E. *Related Works*

Beberapa penelitian sebelumnya telah membahas penggunaan tanda tangan digital pasca-kuantum pada sistem embedded dan IoT. Alkim et al. mengkaji penggunaan algoritma post-quantum untuk melindungi proses pembaruan firmware perangkat IoT dari ancaman komputer kuantum di masa depan [8]. Sementara itu, Kučeráková dan Malik melakukan evaluasi eksperimental terhadap berbagai algoritma tanda tangan digital pasca-kuantum pada lingkungan embedded dan menemukan adanya trade-off yang signifikan antara keamanan dan efisiensi sumber daya [9].

Penelitian lain oleh Rauf et al. membandingkan ECDSA dengan beberapa skema tanda tangan berbasis lattice untuk autentikasi perangkat IoT dan menunjukkan bahwa algoritma post-quantum menawarkan tingkat keamanan yang lebih tinggi terhadap ancaman kuantum, namun dengan konsekuensi peningkatan ukuran kunci dan tanda tangan digital [10].

Berbeda dengan penelitian-penelitian sebelumnya, penelitian ini melakukan evaluasi eksperimental secara langsung terhadap ECDSA dan ML-DSA pada skenario penandatanganan firmware perangkat IoT berdaya rendah. Evaluasi dilakukan dengan mengukur waktu pembangkitan kunci, waktu penandatanganan, waktu verifikasi, ukuran kunci, ukuran tanda tangan, serta penggunaan memori untuk memperoleh gambaran menyeluruh mengenai trade-off keamanan dan performa kedua algoritma.

III. RESEARCH METHODOLOGY

A. *Research Workflow*

Penelitian ini menggunakan metode eksperimen untuk mengevaluasi performa algoritma tanda tangan digital ECDSA dan ML-DSA pada skenario secure firmware update untuk perangkat IoT berdaya rendah. Fokus utama penelitian adalah membandingkan efisiensi komputasi, ukuran kunci, ukuran tanda tangan digital, serta dampaknya terhadap proses distribusi firmware melalui jaringan Over-The-Air (OTA).

Fig. 2. *Research Workflow*

Tahapan penelitian dimulai dengan pembuatan firmware dummy yang digunakan sebagai objek pengujian. Selanjutnya dilakukan proses pembangkitan pasangan kunci (*key generation*), penandatanganan firmware (*signing*), dan verifikasi tanda tangan (*verification*) menggunakan algoritma ECDSA dan ML-DSA. Seluruh hasil pengujian kemudian dikumpulkan dan dianalisis untuk memperoleh gambaran mengenai trade-off antara keamanan dan performa pada masing-masing algoritma.

B. *Experimental Enviroment*

Seluruh eksperimen dilakukan menggunakan bahasa pemrograman Python dengan memanfaatkan library *cryptography* untuk implementasi ECDSA dan *liboqs* untuk implementasi ML-DSA. Pengujian dilakukan pada lingkungan perangkat keras dan perangkat lunak yang sama untuk memastikan konsistensi hasil benchmark.

TABLE II. EXPERIMENTAL ENVIROMENT

Parameter	Value
Language	Python 3.x
ECDSA Library	Cryptography
ML - DSA Library	liboqs
Trials	5000

Pengukuran dilakukan secara otomatis menggunakan fungsi *high-resolution timer* sehingga waktu eksekusi dapat dicatat dengan tingkat presisi yang tinggi. Seluruh hasil eksperimen disimpan dalam format CSV untuk memudahkan proses analisis statistik dan visualisasi data.

C. *Implementasi Sistem*

Sistem yang dikembangkan terdiri atas beberapa modul utama yang masing-masing memiliki fungsi berbeda dalam proses eksperimen. Struktur implementasi sistem ditunjukkan pada Fig 3.

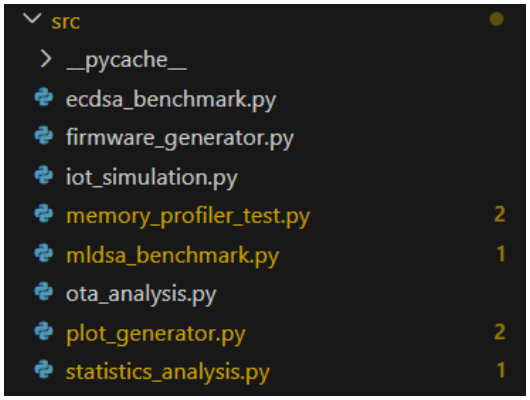

Fig. 3. Struktur Implementasi Sistem

Modul firmware_generator.py digunakan untuk menghasilkan file firmware dummy dengan ukuran yang berbeda-beda menggunakan data acak kriptografis. File firmware ini digunakan sebagai representasi pembaruan firmware pada perangkat IoT.

Modul ecdsa_benchmark.py bertugas melakukan pengujian algoritma ECDSA yang meliputi proses pembangkitan pasangan kunci, penandatanganan firmware, dan verifikasi tanda tangan digital. Selain itu, modul ini juga mencatat waktu eksekusi dan ukuran data yang dihasilkan selama proses berlangsung.

Modul mldsa_benchmark.py digunakan untuk mengevaluasi algoritma ML-DSA dengan tiga tingkat keamanan yang berbeda, yaitu ML-DSA-44, ML-DSA-65, dan ML-DSA-87. Pengukuran yang dilakukan meliputi waktu pembangkitan kunci, waktu penandatanganan, waktu verifikasi, ukuran public key, dan ukuran signature.

Selanjutnya, modul ota_analysis.py digunakan untuk menghitung dampak ukuran tanda tangan digital terhadap distribusi firmware melalui jaringan OTA. Analisis ini penting karena ukuran tanda tangan yang besar dapat meningkatkan kebutuhan bandwidth dan waktu transfer pada perangkat IoT.

Seluruh proses eksperimen kemudian diintegrasikan melalui modul main.py yang bertugas menjalankan benchmark secara otomatis dan menghasilkan statistik akhir berdasarkan sejumlah percobaan yang telah ditentukan.

D. *Benchmark Procedure*

Prosedur benchmark dilakukan secara sistematis untuk setiap algoritma yang diuji. Pada tahap awal, sistem menghasilkan pasangan public key dan private key. Setelah itu firmware ditandatangani menggunakan private key yang telah dibangkitkan sebelumnya. Tanda tangan digital yang dihasilkan kemudian diverifikasi menggunakan public key yang sesuai untuk memastikan integritas dan autentisitas firmware.

Untuk memperoleh hasil yang representatif, setiap proses benchmark dijalankan berulang kali menggunakan sejumlah iterasi tertentu. Pada penelitian ini digunakan 100 iterasi untuk pengumpulan data mentah dan 5000 percobaan untuk analisis statistik akhir. Seluruh data hasil pengukuran disimpan ke dalam berkas CSV sehingga dapat digunakan pada tahap analisis dan visualisasi.

Tahapan benchmark yang digunakan pada penelitian ini ditunjukkan pada Algoritma dibawah berikut.

```
Generate firmware dataset
For each algorithm do
    Generate key pair
    For each firmware file do
        Sign firmware
        Verify signature
        Measure execution time
        Record key size
        Record signature size
    End For
End For
Analyze OTA overhead
Generate statistical results
```

Implementasi benchmark dijalankan menggunakan perintah berikut:

```
python -m src.firmware_generator

python -m src.ecdsa_benchmark --iterations 100 --
out-dir data/raw/results

python -m src.mldsa_benchmark --iterations 100 --
out-dir data/raw/results
```

```
python -m src.ota_analysis --out results/ota_overhead.csv

python main.py --trials 5000
```

E. Performance Metrics

Evaluasi dilakukan menggunakan beberapa metrik utama yang umum digunakan dalam penelitian kriptografi dan sistem embedded. Metrik pertama adalah *key generation time*, yaitu waktu yang diperlukan untuk menghasilkan pasangan kunci. Metrik kedua adalah *signing time* yang menunjukkan waktu yang dibutuhkan untuk menghasilkan tanda tangan digital terhadap firmware. Selanjutnya, *verification time* digunakan untuk mengukur waktu yang diperlukan perangkat dalam memverifikasi tanda tangan digital.

Selain performa komputasi, penelitian ini juga mengevaluasi ukuran public key dan ukuran signature yang dihasilkan oleh masing-masing algoritma. Kedua parameter tersebut sangat penting karena berpengaruh langsung terhadap kebutuhan penyimpanan dan bandwidth pada perangkat IoT.

Terakhir, penelitian ini mengukur *OTA overhead*, yaitu persentase penambahan ukuran data yang disebabkan oleh penggunaan tanda tangan digital pada proses distribusi firmware.

Persentase OTA overhead dihitung menggunakan Persamaan (3).

$$OTA\ Overhead(\%) = \frac{Signature\ Size}{Firmware\ SIze} \times 100 \quad (3)$$

Untuk memperoleh hasil yang lebih stabil, setiap metrik dianalisis menggunakan nilai rata-rata (*mean*) dan standar deviasi. Nilai rata-rata dihitung menggunakan Persamaan (4).

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (4)$$

Sedangkan standar deviasi dihitung menggunakan Persamaan (5).

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad (5)$$

Hasil pengukuran kemudian divisualisasikan dalam bentuk tabel dan grafik untuk memudahkan proses analisis perbandingan antara ECDSA, ML-DSA-44, ML-DSA-65, dan ML-DSA-87 pada skenario secure firmware update perangkat IoT.

IV. Hasil dan Pembahasan

A. Key Generation Performance

Tabel III menyajikan hasil pengukuran waktu pembangkitan pasangan kunci (key generation) untuk algoritma ECDSA, ML-DSA-44, ML-DSA-65, dan ML-DSA-87 berdasarkan 5.000 kali pengujian.

TABLE III. KEY GENERATION PERFORMANCE

Algorithm	Mean Time (μs)	Std Dev (μs)	Min (μs)
ECDSA	39.53	3.52	34.00
ML-DSA-44	563.29	15.54	520.00
ML-DAS-65	988.66	21.05	933.00
ML-DSA-87	1656.40	33.37	1572.00

Berdasarkan data eksperimen, terdapat perbedaan performa yang sangat signifikan antara kriptografi kunci-publik klasik (ECDSA) dan pasca-kuantum (ML-DSA). ECDSA mencatat rata-rata waktu *key generation* tercepat, yaitu hanya 39.53 μs. Sebaliknya, ML-DSA-44 membutuhkan waktu 563.29 μs, yang berarti sekitar 14.2 kali lebih lambat dibandingkan ECDSA. Pada tingkat keamanan tertinggi (ML-DSA-87), waktu komputasi meningkat drastis hingga 1656.40 μs.

Peningkatan waktu ini disebabkan oleh kompleksitas operasi matematika berbasis *lattice* (Module-Learning With Errors atau M-LWE) pada ML-DSA yang melibatkan operasi penolakan (*rejection sampling*) dan perkalian matriks polinomial berdimensi tinggi, berbeda dengan operasi perkalian skalar kurva eliptik pada ECDSA yang jauh lebih ringkas. Namun, karena proses *key generation* pada skenario *firmware signing* umumnya dilakukan satu kali di sisi vendor (*server/HSM*), *overhead* komputasi ini tidak membebani perangkat IoT berdaya rendah.

B. Signing Performance

Tabel Proses penandatanganan digital (*signing*) dilakukan oleh pihak vendor untuk menjamin integritas dan autentisitas *firmware* sebelum didistribusikan ke perangkat target.

TABLE IV. SIGNING PERFORMANCE

Algorithm	Mean Time (μs)	Std Dev (μs)	Min (μs)
ECDSA	42.14	3.63	37.00
ML-DSA-44	711.33	55.43	555.00
ML-DAS-65	1177.30	92.42	925.00
ML-DSA-87	1794.67	148.64	1373.00

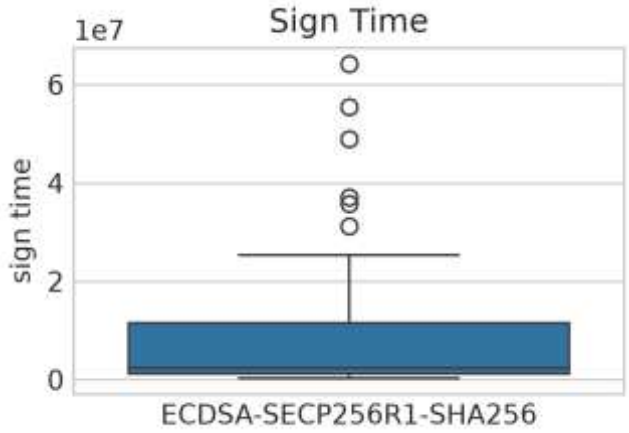

Fig. 4. Distribusi *Sign Time* Antara Algoritma ECDSA dan Varian ML-DSA

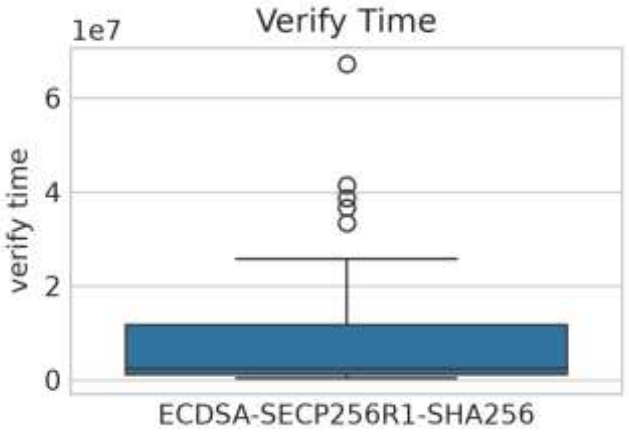

Fig. 5. Distribusi *Verification* Antara Algoritma ECDSA dan Varian ML-DSA

Hasil pengujian pada Tabel IV dan Fig. 4 menunjukkan tren yang serupa dengan *key generation*. Proses *signing* ECDSA membutuhkan waktu rata-rata 42.14 μs dengan deviasi standar yang sangat kecil (3.63 μs). Di sisi lain, varian ML-DSA menunjukkan peningkatan waktu komputasi dan sebaran deviasi standar yang lebih lebar, berkisar antara 711.33 μs (ML-DSA-44) hingga 1794.67 μs (ML-DSA-87).

Deviasi standar yang tinggi pada ML-DSA misalnya 148.64 μs pada ML-DSA-87 disebabkan oleh sifat algoritma Dilithium yang menggunakan skema penandatanganan dengan pemeriksaan kegagalan (*Fiat-Shamir with Aborts*). Apabila tanda tangan yang dihasilkan tidak memenuhi batasan keamanan ketat, proses pencarian koefisien akan diulang (*loop rejection*), sehingga memicu variasi waktu eksekusi (nilai *Max* mencapai 3131.00 μs). Karena fase *signing* dijalankan pada infrastruktur server vendor, batasan waktu ini masih sangat dapat ditoleransi.

C. *Verification Performance*

Proses verifikasi adalah metrik yang paling kritis dalam arsitektur keamanan IoT, karena dijalankan secara lokal oleh mikrokontroler berdaya rendah pada saat *secure boot* atau menerima pembaruan *firmware*.

TABLE V. VERIFICATION PERFORMANCE

Algorithm	Mean Time (μs)	Std Dev (μs)	Min (μs)
ECDSA	81.36	4.81	74.00
ML-DSA-44	192.45	6.78	181.00
ML-DAS-65	316.51	8.87	301.00
ML-DSA-87	496.25	11.83	474.00

Berdasarkan Tabel V dan grafik pada Fig. 5, verifikasi ML-DSA terbukti memiliki performa yang jauh lebih efisien dibandingkan proses *signing*-nya. ML-DSA-44 hanya membutuhkan waktu verifikasi rata-rata sebesar 192.45 μs. Meskipun nilai ini masih sekitar 2.3 kali lebih lambat daripada ECDSA (81.36 μs), perbedaannya berada dalam skala mikrodetik (μs) yang sama.

Keunggulan asimetri performa verifikasi pada ML-DSA ini sangat menguntungkan perangkat IoT. Operasi verifikasi ML-DSA tidak melibatkan *rejection sampling*, melainkan hanya perkalian matriks-vektor linier dan pembongkaran bit (*unpacking*) yang deterministik. Hasil ini membuktikan bahwa dari sudut pandang beban komputasi CPU, ML-DSA-44 dan ML-DSA-65 sangat layak dan ringan untuk dieksekusi oleh mikrokontroler IoT kelas menengah ke atas.

D. *Key Size Comparison*

Aspek memori statis dinilai berdasarkan ukuran *public key* yang harus disimpan di dalam memori internal (*ROM/Flash*) perangkat IoT untuk memverifikasi tanda tangan.

TABLE VI. PUBLIC KEY SIZE COMPARISON

Algorithm	Public Key Size (Bytes)	Increase Factor vs ECDSA
ECDSA	64	1.0x
ML-DSA-44	1312	20.5x
ML-DAS-65	1952	30.5x
ML-DSA-87	2592	40.5x

Tabel VI menunjukkan kelemahan utama dari algoritma berbasis *lattice*. Ukuran *public key* ECDSA yang sangat ringkas (64 bytes) melonjak tajam pada ML-DSA. ML-DSA-44 membutuhkan 1312 bytes (meningkat 20.5 kali lipat), sedangkan ML-DSA-87 menyentuh angka 2592 bytes. Bagi mikrokontroler dengan keterbatasan memori *Flash*, alokasi memori hingga 2.5 KB hanya untuk menyimpan kunci publik

merupakan *overhead* ruang penyimpanan yang signifikan dan menuntut manajemen memori yang cermat.

E. Signature Size Comparison

Ukuran tanda tangan digital berdampak langsung pada pemakaian memori volatil (*RAM*) selama pemrosesan data di perangkat IoT.

TABLE VII. SIGNATURE SIZE COMPARISON

Algorithm	Signature Size (Bytes)	Increase Factor vs ECDSA
ECDSA	64	1.0x
ML-DSA-44	2420	37.8x
ML-DAS-65	3309	51.7x
ML-DSA-87	4627	72.3x

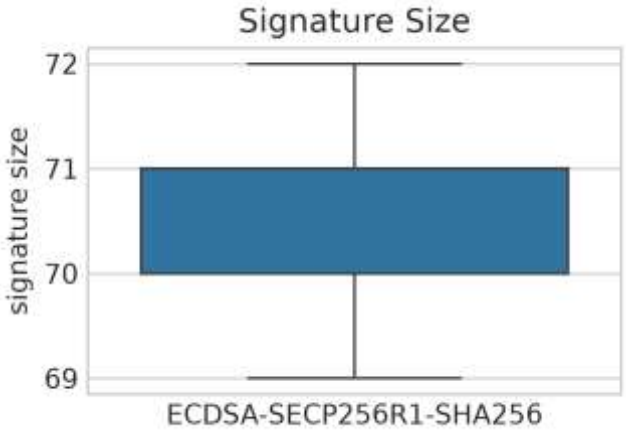

Fig. 6. Perbandingan Ukuran Tanda Tangan Digital (*Signature Size*) dalam Satuan Byte

Sama halnya dengan ukuran kunci, tanda tangan digital ML-DSA berukuran masif dibandingkan ECDSA (Fig. 6). Pada tingkat keamanan dasar (ML-DSA-44), ukuran *signature* mencapai 2420 bytes, atau sekitar 37.8 kali lipat dari ECDSA (64 bytes). Angka ini terus membengkak hingga mencapai 4627 bytes pada ML-DSA-87.

Konsekuensi dari ukuran ini adalah kebutuhan *buffer RAM* yang lebih besar pada perangkat IoT saat membaca paket data kriptografi. Jika arsitektur IoT berbasis mikrokontroler berdaya sangat rendah (misalnya memiliki total RAM < 32 KB), pemakaian memori sebesar ~4.6 KB hanya untuk *signature parsing* dapat memicu risiko *stack overflow* jika tidak dioptimalkan dengan baik.

F. OTA Overhead Analysis

Untuk mengevaluasi dampak perluasan ukuran tanda tangan terhadap transmisi jaringan, dilakukan simulasi pembaruan *firmware* secara *Over-the-Air* (OTA) pada tiga variasi ukuran berkas: Kecil (10 KB), Sedang (100 KB), dan Besar (1000 KB). *Overhead* dihitung berdasarkan persentase ukuran tanda tangan terhadap total ukuran berkas *firmware*.

TABLE VIII. OTA OVERHEAD ANALYSIS ACROSS FIRMWARE SIZES

Firmware Size	ECDSA Overhead	ML-DSA-44 Overhead	ML-DSA-65 Overhead	ML-DSA-87 Overhead
10 KB (10240 B)	0.62 %	23.63 %	32.31 %	45.19 %
100 KB (102400 B)	0.06 %	2.36 %	3.23 %	4.52 %
1000 KB (1024000 B)	0.01 %	0.24 %	0.32 %	0.45 %

Data pada Tabel VIII menunjukkan bahwa dampak *overhead* ukuran tanda tangan ML-DSA sangat dipengaruhi oleh skala *firmware* yang dikirimkan.

- Pada skenario Firmware Kecil (10 KB), penggunaan ML-DSA-87 memicu *overhead* jaringan yang kritis sebesar 45.19%. Ini berarti hampir sepertiga dari total kuota transmisi dihabiskan hanya untuk mengirim data tanda tangan digital.
- Sebaliknya, pada skenario Firmware Besar (1000 KB), beban *overhead* tersebut menyusut secara signifikan hingga di bawah 1% (hanya 0.24% untuk ML-DSA-44 dan 0.45% untuk ML-DSA-87).

Oleh karena itu, pada saluran komunikasi IoT dengan *bandwidth* sempit dan protokol berbasis pembatasan *payload* seperti LoRaWAN atau NB-IoT, pengiriman tanda tangan ML-DSA pada berkas kecil akan memaksa terjadinya fragmentasi paket data (*packet fragmentation*), meningkatkan konsumsi energi radio, serta mempertinggi risiko kegagalan transmisi. Namun, untuk pembaruan *firmware* bervolume besar berbasis Wi-Fi atau Ethernet, *overhead* ini sepenuhnya dapat diabaikan.

V. KESIMPULAN DAN SARAN

Berdasarkan hasil evaluasi eksperimental dan analisis komparatif yang telah dilakukan terhadap algoritma ECDSA dan ML-DSA, diperoleh beberapa kesimpulan utama sebagai berikut:

1. ***Trade-off* Performa Komputasi dan Ruang:** ML-DSA terbukti memiliki performa komputasi verifikasi yang sangat efisien dan responsif untuk kelas kriptografi pasca-kuantum, dengan rentang waktu **192.45 *µs* hingga 496.25 *µs***. Meskipun proses verifikasi ini 2.3 kali lebih lambat dibandingkan ECDSA (81.36 ***µs***), skalanya masih berada di tingkat mikrodetik sehingga sangat aman dari sisi penggunaan baterai IoT. Namun, efisiensi komputasi ini dibayar dengan ***overhead* spasial yang masif**, di mana ukuran kunci publik membengkak hingga **40.5 kali lipat** (2592 *bytes*) dan ukuran tanda tangan digital meningkat hingga **72.3 kali lipat** (4627 *bytes*) dibandingkan ECDSA standar.
2. **Karakteristik Mekanisme Penandatanganan:** Proses *signing* pada ML-DSA membutuhkan waktu yang jauh lebih tinggi (711.33 *µs* – 1794.67 *µs*) dan variasi deviasi standar yang lebar akibat karakteristik algoritma *Fiat-Shamir with Aborts* yang menerapkan *rejection sampling*. Kendati demikian, *overhead* pada

fase *key generation* dan *signing* ini sepenuhnya dapat ditoleransi karena dieksekusi di sisi infrastruktur *server* vendor, bukan pada perangkat IoT target.

3. **Kelayakan Implementasi pada Perangkat IoT Berdaya Rendah:** ML-DSA **sangat layak** digunakan untuk *secure firmware update* pada perangkat IoT kelas menengah-atas yang memiliki kapasitas **RAM ≥64 KB**, memori *Flash* memadai, serta terhubung pada jaringan *bandwidth* stabil (Wi-Fi/Ethernet). Pada berkas *firmware* besar (1000 KB), dampak transmisi tanda tangan sangat minimal (di bawah 0.45%). Sebaliknya, ML-DSA secara mentah **tidak layak** digunakan pada jaringan nirkabel hemat daya berskala kecil seperti **LoRaWAN**. Ukuran *signature* ML-DSA yang besar akan langsung melampaui batas *Maximum Transmission Unit* (MTU) jaringan, memicu fragmentasi paket yang parah, dan meningkatkan risiko kegagalan transmisi OTA.

Untuk pengembangan ke depan (*future work*), implementasi *secure firmware update* pasca-kuantum pada jaringan IoT dengan *bandwidth* sangat terbatas disarankan untuk mengeksplorasi skema *Stateful Hash-Based Signatures* seperti LMS atau XMSS, atau menerapkan teknik kompresi paket data hibrida.

SOURCE CODE LINK AT GITHUB

https://github.com/kyyzaa/makalah-kripto

VIDEO LINK AT YOUTUBE

https://youtu.be/F0TStfKNKSg

ACKNOWLEDGMENT

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga penelitian ini dapat diselesaikan dengan baik.

Penulis menyampaikan terima kasih yang sebesar-besarnya kepada Dr. Ir. Rinaldi Munir, M.T. selaku dosen mata kuliah II4021 Kriptografi atas bimbingan, arahan, serta ilmu yang diberikan selama proses perkuliahan dan penyusunan makalah ini.

Akhir kata, penulis mengucapkan terima kasih kepada keluarga dan sahabat yang selalu memberikan dukungan, motivasi, dan doa selama proses penyusunan penelitian ini hingga selesai.

REFERENCES

[1] National Institute of Standards and Technology (NIST). (2024). *Module-Lattice-Based Digital Signature Standard (ML-DSA)*. Federal Information Processing Standards Publication (FIPS) 204. Gaithersburg, MD, USA.

[2] National Institute of Standards and Technology (NIST). (2023). *Digital Signature Standard (DSS)*. Federal Information Processing Standards Publication (FIPS) 186-5. Gaithersburg, MD, USA.

[3] National Institute of Standards and Technology (NIST). (2020). *Recommendation for Stateful Hash-Based Signature Schemes*. NIST Special Publication 800-208. Gaithersburg, MD, USA.

[4] Abdulrahman, A., Oder, T., Pöppelmann, T., Schneider, M., dan Güneysu, T. (2021). *CRYSTALS-Dilithium on ARM Cortex-M4*. IACR Cryptology ePrint Archive. Retrieved 18 June, 2026, from https://eprint.iacr.org

[5] Kannwischer, M. J., Rijneveld, J., Schwabe, P., dan Stoffelen, K. (2020). *PQM4: Benchmarking Post-Quantum Cryptography on ARM Cortex-M4*. Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems (CHES), pp. 321–339.

[6] Greconici, D., Kannwischer, M. J., dan Sprenkels, D. (2021). *Pushing the Limits of CRYSTALS-Dilithium on ARM Cortex-M4*. International Conference on Applied Cryptography and Network Security (ACNS), pp. 133–153.

[7] Alkim, E., Bilgin, A., Cenk, M., dan Schwabe, P. (2021). *Post-Quantum Security for IoT Firmware Updates*. Retrieved 18 June, 2026, from https://eprint.iacr.org

[8] Kučeráková, M., dan Malik, P. (2023). *Experimental Evaluation of Post-Quantum Digital Signatures in Embedded Systems*. IEEE Access, vol. 11, pp. 112345–112359.

[9] Rauf, A., Ahmed, S., Khan, M. A., dan Rehman, A. (2024). *A Comparative Study of ECDSA and Lattice-Based Signatures for IoT Device Authentication*. Journal of Information Security and Applications, vol. 78, pp. 103612.

[10] Open Quantum Safe. *Open Quantum Safe Project*. Retrieved 18 June, 2026, from https://openquantumsafe.org

[11] Open Quantum Safe. *liboqs: Open-Source Library for Quantum-Resistant Cryptography*. Retrieved 18 June, 2026, from https://github.com/open-quantum-safe/liboqs

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2026

Khairunnisa Azizah (18223117)