INSTITUT TEKNOLOGI DEL

COMPARATIVE STUDY OF IOT RESOURCE CONSUMPTION FOR FIRMWARE UPDATE (CASE STUDY OVER-THE-AIR MECHANISM)

TUGAS AKHIR

13323020 NAOMI APRILIA BUTAR BUTAR

13323025 BASANTA ALFONSO HUTASOIT

13323031 HANDIKA PRATAMA NAINGGOLAN

FAKULTAS VOKASI

PROGRAM STUDI DIII TEKNOLOGI KOMPUTER

INSTITUT TEKNOLOGI DEL

2025

INSTITUT TEKNOLOGI DEL

COMPARATIVE STUDY OF IOT RESOURCE CONSUMPTION FOR FIRMWARE UPDATE (CASE STUDY OVER-THE-AIR MECHANISM)

TUGAS AKHIR

Diajukan sebagai salah satu syarat untuk memperoleh gelar
Ahli Madya
Teknik (A.Md.T)

13323020 NAOMI APRILIA BUTAR BUTAR
13323025 BASANTA ALFONSO HUTASOIT
13323031 HANDIKA PRATAMA NAINGGOLAN

FAKULTAS VOKASI
PROGRAM STUDI DIII TEKNOLOGI KOMPUTER

DAFTAR ISI

DAFTAR ISI 1
DAFTAR TABEL 4
DAFTAR GAMBAR 7
BAB I PENDAHULUAN 9
1.1 Latar Belakang 9
1.2 Rumusan Masalah 11
1.3 Tujuan 12
1.4 Batasan Penelitian 12
1.5 Hasil yang Diharapkan 12
1.6 Sistematika Penyajian 13
BAB II TINJAUAN PUSTAKA 15
2.1 Landasan Teori 15
2.1.1 Konsep Dasar Keamanan dan Efisiensi pada IoT 15
2.1.2 Kriptografi pada Perangkat Bersumber Daya Terbatas 16
2.2 Algoritma Keamanan yang Digunakan pada OTA 17
2.2.1 Hash (SHA-256) 17
2.2.2 Digital Token (HMAC-SHA256) 28
2.2.3 Enkripsi (AES) 34
2.3 Internet of Things (IoT) dan Firmware 46
2.3.1 Konsep IoT dan Peran Firmware 46
2.3.2 Keamanan Firmware IoT dan Risiko Serangan 46
2.4 Mekanisme Over-the-Air (OTA) Update 47
2.4.1 Konsep dan Tahapan OTA 47
2.4.2 Tantangan OTA pada Perangkat IoT Bersumber Daya Terbatas 49
2.5 Analisis Penggunaan Sumber Daya (Performance Resource Usage) 50
2.5.1 Penggunaan RAM dan Flash 50
2.5.2 Beban CPU dan Komputasi Kriptografi 54
2.6 Hardware 56
2.6.1 ESP-32 56
2.6.2 Sensor Suhu 63
2.7 Software 64

2.7.1 Arduino IDE 64
2.7.2 C++ 65
2.7.3 PHP 65
2.7.4 VS Code 65
2.8 Studi Terdahulu (Related Work) 66
2.8.1 Pengaplikasian Teknik Over The Air (OTA) untuk Smart Breaker ESP32 66
2.8.2 Optimization Methods Regarding Battery Life and Write Speed to an SD-Card 66
2.8.3 Pengembangan Perangkat Pembelajaran Mikrokontroler ESP32 Berbasis IoT & Bluetooth 67
2.8.4 Update Time Delay pada Peralihan Fase Lampu Lalu Lintas Menggunakan Teknik OTA 67
2.9 Gap Penelitian 68
2.10 Jadwal Penelitian 70
2.11 Estimasi Biaya Penelitian 70
BAB III ANALISIS DAN DESAIN 71
3.1.1 Analisis Masalah 71
3.1.2 Analisis Pemecahan Masalah 72
3.2 Analisis Kebutuhan Sistem 73
3.2.1 Analisis Kebutuhan Perangkat Keras (Hardware) 73
3.2.2 Analisis Kebutuhan Perangkat Lunak (Software) 75
3.3 Desain Sistem 77
3.3.1 Model Perancangan Sistem (Layer / Logical Architecture) 77
3.3.2 Desain Arsitektur Sistem 79
3.3.3 Diagram Blok Sistem 80
3.3.4 Model Alur Komunikasi OTA 83
3.3.5 Desain Antarmuka Web (Upload/Cek Update/Reboot) 84
3.4 Desain Proses (Flowchart OTA) 86
3.4.1 Flowchart Plain OTA 86
3.4.2 Flowchart OTA dengan SHA-256 88
3.4.3 Flowchart OTA dengan Digital Token (HMAC-SHA256) 90
3.4.4 Flowchart OTA dengan AES (Enkripsi) 91

3.5 Skenario Pengujian 93
3.5.1 Skenario Pengujian Fungsionalitas 93
3.5.2 Skenario Pengujian Komunikasi Data 94
3.5.3 Skenario Pengujian Resource Usage (CPU, RAM, Energi, Waktu OTA) 94
BAB IV HASIL DAN PEMABAHASAN 96
4.1 Implementasi Sistem OTA 96
4.1.1 Alur Proses Update OTA 96
4.1.2 Implementasi Algoritma Keamanan OTA 97
4.2 Hasil Implementasi Pengiriman OTA 115
4.2.1 Tahap Pengunggahan Firmware ke Server OTA 115
4.2.2 Tahap Pemeriksaan Ketersediaan Update (Check Update) 116
4.2.3 Tahap Pengunduhan Firmware ke ESP32 (Download OTA) 117
4.2.4 Aktivasi Firmware Baru (Reboot & Verifikasi Versi) 118
4.3 Hasil Pengukuran Resource Selama OTA 119
4.3.1 Durasi Waktu Update OTA 120
4.3.2 Kecepatan Unduh Firmware (kb/s) 122
4.3.3 Pengukuran Penggunaan CPU Rata-rata (CPU Avg %) 124
4.3.4 Penggunaan CPU Maksimum (CPU Max %) 126
4.3.5 Penggunaan RAM (MAX/MIN/AVG) 128
4.4 Hasil Hipotesa Algoritma 132
BAB V KESIMPULAN DAN SARAN 134
5.1 Kesimpulan 134
5.2 Saran 135
Daftar Pustaka 136
LAMPIRAN 141

DAFTAR TABEL

Table 1 Representasi byte pesan dan panjangnya 18
Table 2 Perhitungan padding secara konsep 19
Table 3 Hasil padding dalam bentuk 64 byte (512 bit) 19
Table 4 Parsing blok menjadi 16 word 32-bit 20
Table 5 Message Schedule 21
Table 6 Inisialisasi variabel a…h (initial hash values) 22
Table 7 Konstanta round SHA-256 ($K_0 \ldots K_{63}$) 22
Table 8 Definisi simbol untuk T_1 23
Table 9 Nilai komponen T_1 (sesuai contoh numerik) 23
Table 10 Penjumlahan T_1 (mod 2^{32}) 24
Table 11 Definisi simbol untuk T_2 24
Table 12 Nilai komponen T_2 (sesuai contoh) 24
Table 13 Penjumlahan T_2 (mod 2^{32}) 25
Table 14 Update a…h pada t = 0 25
Table 15 Ringkasan state setelah putaran pertama 26
Table 16 Penjumlahan intermediate hash dengan initial hash 26
Table 17 Penggabungan delapan word 32-bit 27
Table 18 Hasil Konversi Pesan 29
Table 19 Notasi Dasar yang Dipakai 36
Table 20 Konversi Plaintext dan Key ke Hex 36
Table 21 Penyusunan Plaintext ke State 4×4 (per kolom) 37
Table 22 Penyusunan Key ke State 4×4 37
Table 23 Contoh XOR per byte (Posisi [Row0,Col0] dan [Row0,Col1]) 37
Table 24 State setelah AddRoundKey Awal 38
Table 25 Contoh Lookup S-Box 38
Table 26 State setelah SubBytes 38
Table 27 Proses ShiftRows 39
Table 28 State setelah ShiftRows 39
Table 29 Perhitungan 02·C6 dan 03·C6 di GF(2^8) 40
Table 30 State setelah MixColumns 40
Table 31 Contoh XOR AddRoundKey (beberapa posisi) 41

Table 32 State setelah AddRoundKey (round ini) 41
Table 33 State setelah SubBytes (round berikutnya) 41
Table 34 ShiftRows (round berikutnya) 42
Table 35 State setelah ShiftRows (round berikutnya) 42
Table 36 Contoh XOR pada AddRoundKey final 43
Table 37 State akhir (Ciphertext blok contoh) 43
Table 38 Perbandingan urutan enkripsi vs dekripsi 43
Table 39 Penggunaan RAM 52
Table 40 Flah Terhadap Metode OTA 54
Table 41 Beban CPU 54
Table 42 Pengaruh Tiap Mekanisme Keamanan terhadap CPU 55
Table 43 Embedded Flash & Peripheral Interfaces 58
Table 44 Core & Memory 59
Table 45 Wi-Fi & Bluetooth Subsystem 59
Table 46 RF (Radio Frequency) Front-End 60
Table 47 Cryptographic Hardware Accelertion 60
Table 48 RTC & Low-Power Subsystem 60
Table 49 Konfigurasi dan Pemanfaatan FreeRTOS pada ESP32 61
Table 50 Spesifikasi Sensor DS18B20 63
Table 51 Gap Penelitian 68
Table 52 Jadwal Penelitian 70
Table 53 Estimasi Biaya Keseluruhan 70
Table 54 Server OTA 81
Table 55 ESP32 Transmitter dengan Algoritma 82
Table 56 Monitoring Layer 82
Table 57 Skenario Pengujian Fungsionalitas 93
Table 58 Skenario Pengujian Komunikasi Data 94
Table 59 Skenario Pengujian Resource Usage 94
Table 60 Perbandingan Durasi OTA (detik) 120
Table 61 Kecepatan Unduh Firmware 122
Table 62 Pengukuran Penggunaan CPU Rata rata 124
Table 63 Penggunaan CPU Maksimum 126

Table 64 Penggunaan RAM (MAX/MIN/AVG) 128
Table 65 Penggunaan RAM Rata rata 129
Table 66 Penggunaan RAM Maksimum..... 130
Table 67 Pembuktian Hipotesis Penggunaan Resource..... 132
Table 68 Pembuktian Hipotesis Efisiensi Algoritma..... 133

DAFTAR GAMBAR

Gambar 2. 2 Tabel ASCII 18
Gambar 2. 3 Alur Enkripsi algoritma AES 36
Gambar 2. 6 ESP32 56
Gambar 2. 7 Arsitektur ESP32 58
Gambar 2. 8 Sensor DS18B20 63
Gambar 2. 9 Arduino IDE 64
Gambar 2. 10 PHP 65
Gambar 2. 11 Visual Studio Code 65
Gambar 3. 1 Current System 71
Gambar 3. 2 Target Sytem 72
Gambar 3. 3 Model Perancangan Sistem 77
Gambar 3. 4 Desain Arsitektur Sistem OTA 79
Gambar 3. 5 Diagram Blok Sistem 80
Gambar 3. 6 Model Alur Komunikasi OTA 83
Gambar 3. 7 Desain Upload *Firmware* 84
Gambar 3. 8 Web Monitoring ESP32 85
Gambar 3. 9 Flowchart Plain OTA 87
Gambar 3. 10 Flowchart SHA-256 89
Gambar 3. 11 Tampilan *Flowchart* Digital Token 91
Gambar 3. 12 Tampilan *Flowchart* AES 92
Gambar 4. 2 Alur Proses Update OTA 96
Gambar 4. 3 Rangkaian ESP32 dengan Sensor Suhu 115
Gambar 4. 4 File Firmware yang tersimpan di Server Lokal 116
Gambar 4. 5 Website Upload Firmware 116
Gambar 4. 6 Tahap Pemeriksaan Ketersediaan Update 117
Gambar 4. 7 Tahap Pengunduhan Firmware ke ESP32 118
Gambar 4. 8 Aktivasi Firmware Baru 119
Gambar 4. 9 Grafik Durasi Waktu Update 121
Gambar 4. 10 Grafik Kecepatan Unduh Firmware 123
Gambar 4. 11 Grafik Penggunaan CPU Rata Rata 125
Gambar 4. 12 Grafik Penggunaan CPU Maksimum 126

Gambar 4. 13 Grafik Penggunaan RAM Minimum..... 128
Gambar 4. 14 Grafik Penggunaan RAM Rata Rata..... 130
Gambar 4. 15 Penggunaan RAM Maksimum..... 131

BAB I
PENDAHULUAN

Bab ini akan menjelaskan latar belakang masalah untuk memahami alasan dilaksanakannya tugas akhir ini. Tujuan dari tugas akhir akan dipaparkan, dan pertanyaan pertanyaan yang akan dijawab dalam proses pengerjaan akan dirumuskan. Batasan-batasan yang menjadi acuan juga akan dijelaskan, serta hasil yang diharapkan akan dijabarkan. Selain itu, sistematika penyajian tugas akhir akan diuraikan agar pembaca dapat memahami setiap bagian yang membentuk keseluruhan tugas ini.

1.1 Latar Belakang

Perkembangan teknologi Internet of Things (IoT) telah menghasilkan transformasi besar pada berbagai sektor seperti smart home, industri 4.0, kesehatan digital, sistem transportasi cerdas, hingga manajemen energi berbasis otomasi[1][2]. IoT memungkinkan perangkat saling terhubung dan bertukar data secara real-time melalui jaringan, sehingga memungkinkan terjadinya monitoring, analisis prediktif, serta pengendalian otomatis pada sistem terpadu [3][4][5]. Namun, meskipun memiliki peran penting dalam digitalisasi industri, sebagian besar perangkat IoT dirancang dengan keterbatasan sumber daya, seperti memori yang terbatas, kemampuan pemrosesan CPU rendah yang sangat dibatasi oleh karakteristik embedded hardware [6][7].

Salah satu persoalan mendasar yang muncul akibat keterbatasan ini terdapat pada proses pembaruan firmware. Firmware merupakan inti pengendali seluruh fungsi perangkat IoT, sehingga jika perangkat beroperasi menggunakan firmware lama, maka perangkat menjadi rentan terhadap eksploitasi keamanan, bug fungsional, serta penurunan performa sistem secara signifikan[8]. Kondisi ini semakin tinggi risikonya pada perangkat IoT yang terhubung ke jaringan publik, karena dapat terjadi pencurian data, manipulasi konfigurasi, dan penyusupan perangkat lunak berbahaya seperti malware[9][10]. Oleh karena itu, pembaruan firmware bukan hanya bersifat tambahan fungsional, tetapi merupakan bagian penting dalam menjaga lifecycle keamanan perangkat IoT, terutama pada era serangan siber yang semakin berkembang [10][11].

Metode pembaruan tradisional seperti flashing melalui kabel USB, serial programmer, atau SD card, dinilai tidak efektif karena membutuhkan intervensi manual, tidak scalable, serta tidak cocok untuk deployment perangkat IoT dalam jumlah besar [7][12]. Penelitian Supriyanto et al. (2024) menjelaskan bahwa bahkan metode update berbasis MQTT sekalipun masih menuntut arsitektur server-client khusus agar tidak menimbulkan bottleneck jaringan dan peningkatan beban pemrosesan pada perangkat IoT [8]. Hal ini menunjukkan bahwa pembaruan firmware membutuhkan mekanisme distribusi yang terpusat, efisien, dan dapat dilakukan tanpa kontak fisik.

Melalui perkembangan tersebut, pembaruan firmware Over-The-Air (OTA) muncul sebagai solusi yang relevan. OTA memungkinkan perangkat memperoleh pembaruan melalui jaringan menggunakan server terpusat, sehingga mampu meningkatkan skalabilitas, efisiensi manajemen, serta keamanan sistem secara kolektif [9]. Namun, implementasi OTA pada perangkat IoT bukan hanya tentang distribusi firmware, tetapi juga melibatkan aspek kriptografi seperti hashing, autentikasi, dan enkripsi yang pada praktiknya memengaruhi penggunaan sumber daya perangkat. Penelitian Jin (2022) menunjukkan bahwa penggunaan algoritma SHA-2 dan AES pada ESP32 menghasilkan perbedaan signifikan dalam latensi eksekusi dan konsumsi memori, meskipun ESP32 telah memiliki akselerasi kriptografi hardware [10][13]. Selain itu, studi On Misconception of Hardware and Cost in IoT Security and Privacy mengungkap bahwa fitur keamanan bawaan pada ESP32 tetap dibatasi oleh manajemen heap memory, cache CPU, serta latensi proses kriptografi ketika digunakan secara intensif [11].

Beberapa penelitian menunjukkan bahwa proses OTA dengan mekanisme keamanan dapat menambah beban penggunaan resource pada perangkat IoT. Penelitian Bauwens et al. (2020) menjelaskan bahwa proses enkripsi, verifikasi hash, dan manajemen distribusi firmware berkontribusi terhadap peningkatan penggunaan memori dan waktu eksekusi pada perangkat IoT bersumber daya terbatas[14]. Hal ini menunjukkan bahwa penambahan lapisan keamanan bukan hanya meningkatkan proteksi data, tetapi juga memengaruhi performa sistem secara keseluruhan[15]. Selain itu, ESP32 umumnya menjalankan FreeRTOS sebagai sistem operasi real-time yang mengatur proses parallel seperti pembacaan sensor dan koneksi Wi-Fi bersamaan dengan tugas OTA. Studi oleh Arm et al.

(2022) menjelaskan bahwa multitasking pada FreeRTOS meningkatkan konsumsi CPU dan memori karena aktivitas context switching dan inter-task communication, terutama saat task kriptografi berjalan[16]. Dengan demikian, pemilihan metode keamanan OTA perlu mempertimbangkan trade-off antara keamanan dan efisiensi pemanfaatan resource perangkat IoT seperti CPU, RAM, dan durasi update.

Berdasarkan permasalahan tersebut, penelitian ini melakukan kajian komparatif terhadap penggunaan resource ESP32 selama proses OTA menggunakan empat mekanisme, yaitu Plain OTA, SHA-256, token digital berbasis HMAC-SHA256, serta enkripsi AES. Masing-masing teknik memiliki perbedaan fungsi: SHA-256 menjamin integritas, HMAC-SHA256 menambahkan autentikasi sumber firmware, dan AES memastikan kerahasiaan firmware melalui enkripsi simetris. Penelitian ini bertujuan memberikan pemahaman mendalam terkait trade-off keamanan terhadap performa perangkat, khususnya pada parameter durasi update, kecepatan transfer, penggunaan CPU, dan konsumsi RAM. Dengan hasil penelitian ini, diharapkan kontribusi yang diberikan bukan hanya sebatas peningkatan keamanan, tetapi juga penyusunan rekomendasi praktis bagi pengembang sistem IoT dalam memilih mekanisme OTA yang tepat, efisien, dan sesuai dengan keterbatasan sumber daya perangkat embedded seperti ESP32.

1.2 Rumusan Masalah

Berdasarkan latar belakang dan tujuan yang telah dijelaskan, rumusan masalah dalam penelitian ini dapat dirumuskan sebagai berikut:

1. Bagaimana perbandingan penggunaan sumber daya (RAM, CPU, serta waktu pembaruan) pada perangkat ESP32 selama proses firmware update OTA dengan menggunakan mekanisme keamanan yang berbeda (SHA-256, HMAC SHA-256, dan AES)?
2. Di antara mekanisme OTA tersebut, mekanisme manakah yang paling efisien untuk diterapkan pada perangkat IoT dengan keterbatasan sumber daya, ditinjau dari sudut pandang penggunaan RAM, CPU, dan waktu proses pembaruan?

1.3 Tujuan

Tujuan dari penelitian ini adalah menganalisis penggunaan sumber daya pada perangkat IoT berbasis ESP32 selama proses pembaruan firmware secara Over-The-Air (OTA). Penelitian difokuskan pada pengukuran dan perbandingan parameter performa, seperti penggunaan RAM, beban CPU, dan waktu pembaruan firmware, ketika perangkat menjalankan proses update dengan beberapa mekanisme keamanan yang berbeda, yaitu Plain OTA, SHA-256, digital token berbasis HMAC-SHA256, dan enkripsi AES.Melalui analisis tersebut diharapkan dapat diperoleh gambaran yang jelas mengenai seberapa besar pengaruh kompleksitas masing-masing mekanisme keamanan terhadap efisiensi penggunaan sumber daya pada perangkat dengan keterbatasan hardware seperti ESP32.

1.4 Batasan Penelitian

1. Penelitian difokuskan pada perangkat IoT berbasis ESP32.
2. Evaluasi hanya mencakup empat mekanisme *update* berbasis OTA: Plain OTA, SHA-256, digital token, dan enkripsi AES.
3. Pengukuran terbatas pada penggunaan sumber daya (RAM, CPU), waktu proses *update*.
4. Tidak membahas aspek keamanan lanjutan seperti serangan fisik atau analisis protokol komunikasi mendalam.
5. Keterbatasan waktu, perangkat, dan alat ukur membatasi ruang lingkup penelitian.
6. Kajian hanya pada performa dan efisiensi, tanpa evaluasi keamanan secara menyeluruh.

1.5 Hasil yang Diharapkan

Hasil yang diharapkan dari penelitian ini adalah tercapainya pemahaman yang komprehensif mengenai pengaruh mekanisme keamanan OTA terhadap performa dan penggunaan sumber daya pada perangkat ESP32. Secara khusus, penelitian ini menargetkan beberapa luaran berikut:

1. Analisis Komparatif Penggunaan Sumber Daya

   Tersusunnya laporan analisis yang menjelaskan perbandingan penggunaan RAM, beban CPU, serta waktu pembaruan firmware di antara empat mekanisme OTA (Plain OTA, SHA-256, digital token, dan enkripsi AES) pada perangkat ESP32.

2. Model Evaluasi Resource-Efficiency

   Terbentuknya sebuah model atau kerangka evaluasi yang menggambarkan keterkaitan antara kompleksitas mekanisme OTA (Plain, integritas, autentikasi, enkripsi) dengan efisiensi penggunaan sumber daya perangkat, termasuk pengaruhnya terhadap ukuran firmware, waktu update.
3. Rekomendasi Mekanisme OTA yang Paling Efisien

   Tersusunnya rekomendasi praktis mengenai mekanisme OTA yang paling sesuai untuk perangkat IoT dengan keterbatasan hardware, berdasarkan keseimbangan antara efisiensi sumber daya, kecepatan pembaruan, dan keandalan sistem pembaruan firmware.
4. Dokumentasi Teknis dan Data Eksperimen

   Tersedianya dokumentasi teknis berupa rancangan sistem, konfigurasi perangkat keras dan perangkat lunak, skenario pengujian, serta hasil pengukuran performa yang dapat dijadikan rujukan oleh pengembang IoT atau peneliti lain yang ingin merancang dan menguji mekanisme OTA serupa pada perangkat sejenis.

1.6 Sistematika Penyajian

Laporan Tugas Akhir ini disusun dalam beberapa bab dengan sistematika sebagai berikut:

1. Bab I – Pendahuluan

   Bab ini memuat latar belakang masalah, pentingnya pembaruan firmware pada perangkat IoT, serta tantangan keterbatasan sumber daya pada perangkat seperti ESP32. Selain itu, bab ini juga menjelaskan tujuan penelitian, rumusan masalah, batasan penelitian, hasil yang diharapkan, dan gambaran umum sistematika penulisan laporan.
2. Bab II – Tinjauan Pustaka

   Bab ini berisi landasan teori yang mendukung penelitian, meliputi konsep Internet of Things (IoT), peran firmware dalam perangkat IoT, konsep dan tahapan Over-The-Air (OTA) update, serta penjelasan teori mengenai algoritma keamanan yang digunakan (SHA-256, HMAC-SHA256 sebagai digital token, dan AES). Bab ini juga memuat ringkasan penelitian terdahulu yang relevan serta gap penelitian yang menjadi dasar dilakukannya studi ini.
3. Bab III – Analisis dan Desain Sistem

Bab ini menjelaskan analisis kebutuhan sistem, baik perangkat keras maupun perangkat lunak, model perancangan sistem OTA berbasis ESP32, arsitektur sistem, diagram blok, alur komunikasi data, serta desain antarmuka dan flowchart mekanisme OTA untuk setiap skema keamanan (Plain, SHA-256, digital token, dan AES). Bab ini menjadi jembatan antara landasan teori dan implementasi sistem yang akan dibangun.

4. Bab IV – Implementasi dan Hasil Pengujian

   Bab ini berisi penjelasan mengenai implementasi sistem OTA pada ESP32, konfigurasi server OTA, serta langkah-langkah pengujian yang dilakukan. Selain itu, bab ini menyajikan hasil pengukuran penggunaan RAM, CPU, dan waktu pembaruan untuk setiap mekanisme OTA, beserta analisis komparatif untuk menilai trade-off antara tingkat keamanan dan efisiensi sumber daya.

5. Bab V – Kesimpulan dan Saran

   Bab ini merangkum temuan utama penelitian, memberikan jawaban atas rumusan masalah, serta menyampaikan kesimpulan terkait perbandingan penggunaan sumber daya pada berbagai mekanisme OTA. Bab ini juga memuat saran untuk pengembangan lebih lanjut, misalnya pengujian pada jenis perangkat IoT lain, penggunaan protokol komunikasi yang berbeda, atau eksplorasi mekanisme keamanan OTA yang lebih kompleks.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi uraian pustaka yang menjadi landasan teori dalam merancang dan membangun sistem pada Tugas Akhir ini. Kajian yang ditampilkan diambil dari sumber-sumber yang relevan sehingga dapat mendukung analisis serta proses pengembangan yang dilakukan, khususnya terkait keamanan dan efisiensi pembaruan firmware Over-the-Air (OTA) pada perangkat ESP32.

2.1 Landasan Teori

2.1.1 Konsep Dasar Keamanan dan Efisiensi pada IoT

Dalam sistem Internet of Things (IoT), keamanan dan efisiensi merupakan dua aspek yang saling berkaitan erat namun sering kali saling bertentangan. IoT terdiri dari jutaan perangkat yang saling terhubung dan saling bertukar data melalui jaringan nirkabel. Karena komunikasi berlangsung secara terbuka melalui internet, setiap perangkat berpotensi menjadi target serangan seperti manipulasi data, penyusupan firmware, atau injeksi kode berbahaya. Oleh karena itu, diperlukan mekanisme yang dapat menjamin integritas, autentikasi, dan kerahasiaan data selama proses komunikasi maupun pembaruan sistem[17][18].

Namun, tidak seperti komputer atau server dengan sumber daya besar, perangkat IoT seperti ESP32 memiliki keterbatasan signifikan dari sisi CPU, RAM, kapasitas penyimpanan. Mekanisme keamanan seperti hashing, autentikasi digital, atau enkripsi memang mampu meningkatkan perlindungan data, tetapi di sisi lain menambah beban pemrosesan dan memperlambat kinerja sistem. Setiap operasi kriptografi yang dijalankan membutuhkan siklus CPU tambahan, penggunaan memori sementara, dan konsumsi energi yang lebih besar. Akibatnya, penerapan algoritma keamanan yang terlalu berat bisa menyebabkan perangkat menjadi lambat, cepat panas, atau boros daya[19].

Oleh karena itu, dalam konteks penelitian ini, keamanan tidak semata-mata dilihat dari tingkat proteksi kriptografis**,** melainkan dari dampaknya terhadap efisiensi sistem**.** Fokus utama bukan pada seberapa kuat algoritma menahan serangan, tetapi pada seberapa besar pengaruhnya terhadap performa dan penggunaan sumber daya perangkat saat melakukan

proses Over-the-Air (OTA) firmware update**.** Dengan kata lain, keamanan diukur dari biaya komputasi (computational cost) yang timbul meliputi waktu proses, penggunaan CPU, RAM, dan energi[20].

Pendekatan ini relevan untuk perangkat IoT berskala kecil seperti ESP32, di mana efisiensi sumber daya menjadi prioritas utama agar sistem dapat beroperasi stabil dalam jangka panjang. Analisis seperti ini juga penting untuk menentukan trade-off terbaik antara keamanan dan performa, sehingga sistem tetap aman tetapi tidak mengorbankan kecepatan maupun daya tahan perangkat.

2.1.2 Kriptografi pada Perangkat Bersumber Daya Terbatas

Penerapan algoritma kriptografi pada perangkat Internet of Things (IoT) seperti ESP32 menghadirkan tantangan tersendiri karena keterbatasan sumber daya yang dimiliki. Mikrokontroler umumnya memiliki kapasitas RAM kecil, prosesor dengan frekuensi rendah, serta pasokan daya yang terbatas. Kondisi ini menuntut setiap proses kriptografi, seperti hashing atau enkripsi, dilakukan seefisien mungkin agar tidak mengganggu fungsi utama perangkat[21]. Keterbatasan memori (RAM) menyebabkan ruang penyimpanan sementara (buffer) yang digunakan saat proses hashing atau enkripsi harus dioptimalkan. Algoritma yang menggunakan blok data besar dapat mengakibatkan fragmentasi memori atau bahkan menyebabkan stack overflow jika tidak dikendalikan dengan baik. Pada ESP32, setiap byte RAM sangat berharga karena sistem juga harus menjalankan proses lain seperti komunikasi Wi-Fi, manajemen sensor, dan tugas-tugas sistem[22].

Sementara itu, kapasitas CPU yang terbatas membuat efisiensi perhitungan menjadi faktor utama. Proses kriptografi seperti SHA-256 dan AES memerlukan operasi matematis intensif yang berulang-ulang dalam jumlah besar. Jika tidak dioptimalkan, proses tersebut dapat memperlambat eksekusi program lain, memperpanjang waktu update firmware, dan menimbulkan lonjakan konsumsi energi[23]. Selain itu juga meningkat seiring bertambahnya kompleksitas algoritma. Operasi kriptografi yang panjang dan sering dilakukan akan mempercepat habisnya daya baterai, yang menjadi masalah serius pada perangkat IoT yang beroperasi secara nirkabel atau di lokasi tanpa sumber listrik permanen.

Oleh karena itu, keseimbangan antara keamanan dan efisiensi energi menjadi hal yang sangat penting dalam desain sistem IoT.

Dalam konteks penelitian ini, algoritma SHA-256, HMAC-SHA256, dan AES tidak semata-mata diuji untuk menilai tingkat keamanan datanya, melainkan untuk mengamati beban komputasi yang ditimbulkannya terhadap sumber daya perangkat selama proses Over-the-Air (OTA) firmware update pada ESP32. Fokus utama pengujian meliputi durasi update, kecepatan transfer, rata-rata penggunaan CPU, dan pemakaian RAM sepanjang proses berlangsung[24]. Dengan pendekatan ini, penelitian tidak hanya menilai bagaimana algoritma menjaga integritas atau kerahasiaan data, tetapi juga sejauh mana penerapan mekanisme tersebut memengaruhi performa dan efisiensi sistem secara keseluruhan. Hal ini penting untuk menentukan algoritma mana yang paling sesuai diterapkan pada perangkat IoT bersumber daya terbatas tanpa mengorbankan stabilitas sistem.

2.2 Algoritma Keamanan yang Digunakan pada OTA

2.2.1 Hash (SHA-256)

Secure Hash Algorithm 256-bit (SHA-256) merupakan salah satu anggota keluarga SHA-2 yang dikembangkan oleh National Institute of Standards and Technology (NIST) sebagai fungsi kriptografi satu arah (one-way hash function). Algoritma ini bekerja dengan memproses pesan menjadi nilai hash berukuran tetap, yaitu 256 bit (32 byte) yang unik dan tidak dapat dikembalikan ke bentuk semula. Sifat one-way ini menjadikan SHA-256 sangat efektif digunakan dalam proses verifikasi integritas data, di mana data asli tidak perlu dikirim ulang, melainkan cukup dibandingkan nilai hash-nya saja[25].

Dalam konteks keamanan firmware atau pengiriman data di perangkat IoT, SHA-256 digunakan untuk memastikan bahwa file yang diterima tidak mengalami perubahan, manipulasi, atau serangan pihak ketiga selama proses transmisi. Algoritma ini menghasilkan nilai hash yang berbeda meskipun hanya terdapat perubahan satu bit pada data sumber (dikenal sebagai efek avalanche), sehingga sangat sulit bagi penyerang memodifikasi firmware tanpa terdeteksi. Hal ini menjadikan SHA-256 sangat ideal untuk Over-The-Air Update (OTA) pada perangkat mikro seperti ESP32, karena tidak menambah beban memori yang besar serta dapat dijalankan dalam proses streaming tanpa menyimpan

keseluruhan berkas firmware ke RAM. Selain itu, SHA-256 juga banyak digunakan sebagai fungsi keamanan dasar dalam sistem autentikasi digital, blockchain, dan proteksi data berbasis token. Dukungan terhadap akselerasi perangkat keras seperti pada ESP32 semakin meningkatkan efisiensi eksekusi hashing, menjadikan algoritma ini mampu memberikan keamanan tinggi tanpa mengorbankan performa perangkat yang memiliki sumber daya terbatas[25].

Contoh perhitungan SHA-256

Misalkan pesan yang akan dihitung hash-nya adalah: abc

1. Konversi ke ASCII dan biner

Untuk mengkonversi ke bentuk hexa dan biner, dapat langsung melihat tabel ASCII dibawah ini:

Char	ASCII	Decimal	Bits	Char	ASCII	Decimal	Bits	Char	ASCII	Decimal	Bits
0	48	0	000000	F	70	22	010110	d	100	44	101100
1	49	1	000001	G	71	23	010111	e	101	45	101101
2	50	2	000010	H	72	24	011000	f	102	46	101110
3	51	3	000011	I	73	25	011001	g	103	47	101111
4	52	4	000100	J	74	26	011010	h	104	48	110000
5	53	5	000101	K	75	27	011011	i	105	49	110001
6	54	6	000110	L	76	28	011100	j	106	50	110010
7	55	7	000111	M	77	29	011101	k	107	51	110011
8	56	8	001000	N	78	30	011110	l	108	52	110100
9	57	9	001001	O	79	31	011111	m	109	53	110101
:	58	10	001010	P	80	32	100000	n	110	54	110110
;	59	11	001011	Q	81	33	100001	o	111	55	110111
<	60	12	001100	R	82	34	100010	p	112	56	111000
=	61	13	001101	S	83	35	100011	q	113	57	111001
>	62	14	001110	T	84	36	100100	r	114	58	111010
?	63	15	001111	U	85	37	100101	s	115	59	111011
@	64	16	010000	V	86	38	100110	t	116	60	111100
A	65	17	010001	W	87	39	100111	u	117	61	111101
B	66	18	010010	'	96	40	101000	v	118	62	111110
C	67	19	010011	a	97	41	101001	w	119	63	111111
D	68	20	010100	b	98	42	101010				
E	69	21	010101	c	99	43	101011				

Gambar 2. 1 Tabel ASCII

a) Tahap 1 – Representasi Pesan dan Panjang

Table 1 Representasi byte pesan dan panjangnya

Langkah	Keterangan	Nilai
1	Isi pesan (3 byte)	3, 4, 7 (dalam desimal)
2	3 dalam biner	00000011
3	4 dalam biner	00000100
4	7 dalam biner	00000111
5	Pesan biner M	00000011 00000100 00000111

6	Panjang pesan (byte)	3 byte
7	Panjang pesan (bit)	3 × 8 = 24 bit
8	Notasi panjang	l = 24

Keterangan:

- SHA-256 selalu bekerja pada bit, jadi langkah pertama adalah mengubah tiap byte pesan ke bentuk biner.
- Panjang pesan **l** (dalam bit) nanti digunakan di bagian padding, dan disimpan di 64 bit terakhir blok.

b) Tahap 2 – Padding Pesan sampai 512 bit

Aturan padding SHA-256:

1. Tambahkan bit 1 setelah pesan → dalam praktik: tambahkan satu byte 10000000 (hex 0x80).
2. Tambahkan k bit 0 sampai panjang total ≡ 448 (mod 512).
3. Tambahkan 64 bit terakhir yang berisi panjang pesan asli l dalam big-endian.

Table 2 Perhitungan padding secara konsep

Langkah	**Rumus / Keterangan**	**Hasil**
1	Panjang awal pesan (bit)	l = 24
2	Struktur blok: l + 1 + k + 64 ≡ 0 (mod 512)	total bit = 512
3	24 + 1 + k + 64 = 512 → k = 512 − 89 = 423	k = 423 bit 0
4	Setelah pesan (24 bit) tambahkan 1 bit ‘1’	byte 10000000 (0x80)
5	Tambah 423 bit ‘0’ → diwakili deretan byte 00 sampai posisi 448	semua 0 sampai bit ke-447
6	Panjang l = 24 = 0x18	64-bit big-endian: … 00 18

Table 3 Hasil padding dalam bentuk 64 byte (512 bit)

Byte 0–7
00000011 00000100 00000111 10000000 00000000 00000000 00000000 00000000
Byte 8–15
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Byte 16–23
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Byte 24–31
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Byte 32–39
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Byte 40–47
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Byte 48–55
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Byte 56–63
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011000

Keterangan:

- Byte 0–2 berisi pesan (3,4,7), byte 3 berisi 0x80 (1 bit 1 + 7 bit 0).
- Byte 4–55 semuanya nol (padding).
- Byte 56–63 adalah panjang pesan **l** dalam 64-bit big-endian, yaitu … 00 18. Di sini, hanya byte terakhir yang non-nol (00011000 = 24).

c) Tahap 3 – Parsing Menjadi 16 Word 32-bit (M0…M15)

Blok 512 bit dibagi menjadi 16 word 32-bit (4 byte per word).

Table 4 Parsing blok menjadi 16 word 32-bit

Word	**4 byte (biner)**	**Hex**
M0(0)	00000011 00000100 00000111 10000000	0x03040780
M0(1)	00000000 00000000 00000000 00000000	0x00000000
M0(2)	00000000 00000000 00000000 00000000	0x00000000
M0(3)	00000000 00000000 00000000 00000000	0x00000000
M0(4)	00000000 00000000 00000000 00000000	0x00000000
M0(5)	00000000 00000000 00000000 00000000	0x00000000
M0(6)	00000000 00000000 00000000 00000000	0x00000000
M0(7)	00000000 00000000 00000000 00000000	0x00000000
M0(8)	00000000 00000000 00000000 00000000	0x00000000
M0(9)	00000000 00000000 00000000 00000000	0x00000000
M0(10)	00000000 00000000 00000000 00000000	0x00000000
M0(11)	00000000 00000000 00000000 00000000	0x00000000

M0(12)	00000000 00000000 00000000 00000000	0x00000000
M0(13)	00000000 00000000 00000000 00000000	0x00000000
M0(14)	00000000 00000000 00000000 00000000	0x00000000
M0(15)	00000000 00000000 00000000 00011000	0x00000018

Keterangan:

- M0(0) berisi 4 byte pertama (3,4,7,0x80) yang digabung jadi 32 bit → 0x03040780.
- M0(15) berisi panjang pesan (24 bit) di ujung → 0x00000018.
- Word lain di contoh ini nol karena isinya padding nol.

d) Tahap 4 – Message Schedule $W_0 \ldots W_{63}$

Secara teori:

- Untuk t = 0 … 15 → $W_t = M0(t)$.
- Untuk t = 16 … 63 →

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} (mod\ 2^{32})$$

dengan:

- $\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus (x \gg 3)$
- $\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus (x \gg 10)$

Table 5 Message Schedule

t	W_t (hex)	t	W_t (hex)	t	W_t (hex)	t	W_t (hex)
0	03040780	16	00000000	32	00000000	48	00000000
1	00000000	17	00000000	33	00000000	49	00000000
2	00000000	18	00000000	34	00000000	50	00000000
3	00000000	19	00000000	35	00000000	51	00000000
4	00000000	20	00000000	36	00000000	52	00000000
5	00000000	21	00000000	37	00000000	53	00000000
6	00000000	22	00000000	38	00000000	54	00000000
7	00000000	23	00000000	39	00000000	55	00000000
8	00000000	24	00000000	40	00000000	56	00000000
9	00000000	25	00000000	41	00000000	57	00000000
10	00000000	26	00000000	42	00000000	58	00000000

11	00000000	27	00000000	43	00000000	59	00000000
12	00000000	28	00000000	44	00000000	60	00000000
13	00000000	29	00000000	45	00000000	61	00000000
14	00000000	30	00000000	46	00000000	62	00000000
15	00000018	31	00000000	47	00000000	63	00000000

Keterangan:

- Simbol W_t : input per putaran ke fungsi kompresi SHA-256.
- Di implementasi nyata, $W_{16}…W_{63}$ dihitung menggunakan σ_0 dan σ_1 untuk menyebarkan (difusi) bit dari pesan ke seluruh 64 round.
- Di contoh ini, W_t setelah t=15 ditampilkan sebagai nol untuk menyederhanakan penjelasan (titik fokus adalah cara kerja T_1/T_2 dan update a..h).

e) Tahap 5 – Inisialisasi Variabel a…h dan Konstanta K_t

SHA-256 punya delapan nilai awal (IV) $H_0…H_7$, diturunkan dari akar prima (spesifikasi standar).

Table 6 Inisialisasi variabel a…h (initial hash values)

Variabel	Nilai hex	Keterangan
a	6A09E667	$H_0(0)$
b	BB67AE85	$H_0(1)$
c	3C6EF372	$H_0(2)$
d	A54FF53A	$H_0(3)$
e	510E527F	$H_0(4)$
f	9B05688C	$H_0(5)$
g	1F83D9AB	$H_0(6)$
h	5BE0CD19	$H_0(7)$

Table 7 Konstanta round SHA-256 ($K_0…K_{63}$)

Indeks	K_t (hex)	Indeks	K_t (hex)
K_0	428A2F98	K_1	71374491
K_2	B5C0FBCF	K_3	E9B5DBA5
…	…	…	…
K_{62}	BEF9A3F7	K_{63}	C67178F2

Keterangan:

- a…h adalah state internal 256 bit yang akan diupdate 64 kali dalam satu blok.
- K_t adalah konstanta tetap (hasil fungsi matematis dari bilangan prima), tujuannya menambah non-linearitas dan mencegah serangan struktur sederhana.

f) Tahap 6 – Hash Computation (Round t = 0)

Pada setiap round t (0…63), SHA-256 menghitung:

$$T_1 = h + \Sigma_1(e) + Ch(e,f,g) + K_t + W_t$$

$$T_2 = \Sigma_0(a) + Maj(a,b,c)$$

Lalu memperbarui:

- $h' = g$
- $g' = f$
- $f' = e$
- $e' = d + T_1$
- $d' = c$
- $c' = b$
- $b' = a$
- $a' = T_1 + T_2$

Semua operasi mod 2^{32}.

Komponen T_1 pada t = 0

Table 8 Definisi simbol untuk T_1

Simbol	Rumus / Arti
$\Sigma_1(e)$	$ROTR^6(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e)$
Ch(e,f,g)	$(e \wedge f) \oplus ((\neg e) \wedge g) \rightarrow$ fungsi *choose*, memilih f atau g per bit
K_t	Konstanta round (di sini K_0 = 428A2F98)
W_t	Word dari message schedule (di sini W_0 = 03040780)
h	Nilai h dari state sebelumnya

Table 9 Nilai komponen T_1 (sesuai contoh numerik)

Komponen	Nilai hex	Keterangan
h	5BE0CD19	dari inisialisasi
$\Sigma_1(e)$	3587272B	hasil ROTR & XOR pada e = 510E527F

Ch(e,f,g)	1F85C98C	hasil $(e \,\&\, f) \oplus ((\neg e) \,\&\, g)$
K_0	428A2F98	konstanta round 0
W_0	03040780	word pertama dari pesan

Perhitungan T_1:

$$T_1 = h + \Sigma_1(e) + Ch(e,f,g) + K_0 + W_0$$

Table 10 Penjumlahan T_1 (mod 2^{32})

Langkah	Operasi	Hasil sementara (hex)
1	$h + \Sigma_1(e)$	5BE0CD19 + 3587272B = 9147F444
2	+ Ch(e,f,g)	9147F444 + 1F85C98C = B0CDB5D0
3	+ K_0	B0CDB5D0 + 428A2F98 = F357E568
4	+ W_0	F357E568 + 03040780 = F67BF4E8

Jadi:

Simbol	Nilai
T_1	F67BF4E8

Keterangan:

- $\Sigma_1(e)$ dan Ch(e,f,g) dirancang agar bergantung kuat pada bit-bit e,f,g dan membuat bit keluarannya menyebar (difusi & konfusi).
- Penjumlahan dilakukan **mod 2^{32}**, artinya kalau overflow 32 bit, bagian lebihnya dibuang.

Komponen T_2 pada t = 0

Table 11 Definisi simbol untuk T_2

Simbol	Rumus / Arti
$\Sigma_0(a)$	$ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$
Maj(a,b,c)	$(a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \rightarrow$ *majority* (mayoritas bit)

Table 12 Nilai komponen T_2 (sesuai contoh)

Komponen	Nilai hex	Keterangan
a	6A09E667	dari inisialisasi
b	BB67AE85	dari inisialisasi
c	3C6EF372	dari inisialisasi
$\Sigma_0(a)$	CE20B47E	hasil ROTR & XOR pada a
Maj(a,b,c)	3A6FE667	mayoritas bit dari a,b,c

Perhitungan T_2:

$$T_2 = \Sigma_0(a) + Maj(a,b,c)$$

Table 13 Penjumlahan T_2 (mod 2^{32})

Langkah	Operasi	Hasil sementara (hex)
1	Σ_0(a) + Maj(a,b,c)	$CE20B47E + 3A6FE667 = 08909AE5$

Jadi:

Simbol	Nilai
T_2	08909AE5

Keterangan:

- Maj memberikan 1 jika minimal 2 dari 3 bit (a,b,c) bernilai 1. Ini menyatukan informasi ketiga register.
- Σ_0(a), seperti Σ_1(e), adalah kombinasi rotasi untuk membuat setiap bit output dipengaruhi banyak bit input.

Update register a…h (round t = 0)

Rumus:

- $a' = T_1 + T_2$
- $b' = a$
- $c' = b$
- $d' = c$
- $e' = d + T_1$
- $f' = e$
- $g' = f$
- $h' = g$

Table 14 Update a…h pada t = 0

Variabel	Rumus	Perhitungan	Nilai baru (hex)
a′	$T_1 + T_2$	F67BF4E8 + 08909AE5	FF0C8FCD
b′	a	–	6A09E667
c′	b	–	BB67AE85
d′	c	–	3C6EF372
e′	$d + T_1$	A54FF53A + F67BF4E8	FA2A4622
f′	e	–	510E527F

g′	f	–	9B05688C
h′	g	–	1F83D9AB

Table 15 Ringkasan state setelah putaran pertama

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**
init	6A09E667	BB67AE85	3C6EF372	A54FF53A	510E527F	9B05688C	1F83D9AB	5BE0CD19
t=0	FF0C8FCD	6A09E667	BB67AE85	3C6EF372	FA2A4622	510E527F	9B05688C	1F83D9AB

Keterangan:

- Inilah state intermediate setelah 1 round.
- Proses yang sama diulang untuk t = 1 sampai 63, dengan W_t dan K_t berbeda-beda.
- Intinya: setiap round "mengaduk" a…h dengan cara yang sama sehingga perubahan kecil di pesan berpengaruh besar di hasil akhir.

g) Tahap 7 – Penjumlahan Akhir dengan Initial Hash

Setelah 64 round, kita punya a_{64} … h_{64} (di sini dicontohkan sebagai nilai seperti b87fcbba, f82e573e, dst).

Kemudian:

$$H0' = H0 + a64,\ H1' = H1 + b64, \dots, H7' = H7 + h64$$

Table 16 Penjumlahan intermediate hash dengan initial hash

Indeks	**H awal**	**a_{64}…h_{64}**	**Penjumlahan**	**H′ (hex)**
H_0	6A09E667	B87FCBBA	B87FCBBA + 6A09E667 = 2289B221	2289B221
H_1	BB67AE85	F82E573E	F82E573E + BB67AE85 = B39605C3	B39605C3
H_2	3C6EF372	0CDF7F1E	0CDF7F1E + 3C6EF372 = 494E7290	494E7290
H_3	A54FF53A	E01223AF	E01223AF + A54FF53A = 856218E9	856218E9
H_4	510E527F	E0B1B876	E0B1B876 + 510E527F = 31C00AF5	31C00AF5
H_5	9B05688C	BBCA117B	BBCA117B + 9B05688C = 56CF7A07	56CF7A07

H_6	1F83D9AB	62ED2E6E	62ED2E6E + 1F83D9AB = 82710819	82710819
H_7	5BE0CD19	DF4697F8	DF4697F8 + 5BE0CD19 = 3B276511	3B276511

Keterangan:

- Ini langkah "finalisasi" satu blok: state a…h dijumlahkan dengan nilai awal.
- Hasilnya H_0'…H_7' adalah **hash 256-bit** untuk blok ini.

h) Tahap 8 – Penggabungan H_0' … H_7' dan Nilai Hash Akhir

Table 17 Penggabungan delapan word 32-bit

Urutan	Nilai (hex)
H_0'	2289B221
H_1'	B39605C3
H_2'	494E7290
H_3'	856218E9
H_4'	31C00AF5
H_5'	56CF7A07
H_6'	82710819
H_7'	3B276511

Digabung:

$$Hash = H0' \parallel H1' \parallel \cdots \parallel H7'$$

Nilai hash akhir (256 bit):

2289b221b39605c3494e7290856218e931c00af556cf7a07827108193b276511

Keterangan:

- Simbol || berarti *concatenate* (sambung biner/hex).
- Inilah nilai SHA-256 untuk contoh pesan 3-byte yang kamu gunakan.
- Semua rumus di atas dirancang supaya:
  - Setiap bit input memengaruhi banyak bit output (*avalanche effect*).
  - Sulit menebak pesan dari hash (preimage resistance).
  - Sulit mencari dua pesan berbeda dengan hash sama (collision resistance).

2.2.2 Digital Token (HMAC-SHA256)

Hash-based Message Authentication Code (HMAC) dengan algoritma SHA-256 merupakan metode autentikasi kriptografi yang digunakan untuk menjamin integritas sekaligus keaslian suatu data. Tidak hanya menghitung hash dari data, HMAC–SHA256 juga menggunakan secret key (kunci rahasia) sehingga nilai autentikasi tidak dapat dihasilkan tanpa mengetahui kunci tersebut. Hal ini membedakan HMAC dari fungsi hash biasa, yang hanya memastikan integritas tetapi tidak memverifikasi sumber data[26].

Dalam implementasinya, HMAC bekerja dengan menggabungkan kunci rahasia dan data secara terstruktur, kemudian diproses menggunakan algoritma hash SHA-256 untuk menghasilkan nilai token unik. Token ini akan divalidasi oleh pihak penerima menggunakan kunci yang sama. Apabila data atau kunci telah dimanipulasi, nilai HMAC yang diperoleh akan berbeda sehingga proses autentikasi otomatis gagal. Karena sifatnya yang mengikat data dengan kunci, HMAC–SHA256 sering digunakan dalam sistem komunikasi IoT untuk mencegah firmware palsu, serangan man-in-the-middle, dan modifikasi paket data oleh pihak yang tidak sah[26].

Pada perangkat IoT seperti ESP32, HMAC–SHA256 dapat dijalankan secara streaming, artinya perhitungan token dilakukan bersamaan dengan proses penerimaan data tanpa memerlukan penyimpanan file secara penuh di RAM. Pendekatan ini sangat sesuai untuk perangkat dengan memori terbatas, termasuk ketika mekanisme keamanan diberikan pada proses Over-The-Air (OTA) update. Firmware yang dikirim server akan dilengkapi token HMAC–SHA256, kemudian perangkat menghitung ulang token dari data yang diterima. Hanya firmware dengan token valid yang dapat ditulis ke memori flash, sehingga mencegah instalasi firmware palsu atau berbahaya.

Rumus Dasar HMAC–SHA256

Secara umum, HMAC didefinisikan:

$$HMAC(K, M) = H((K' \oplus opad) \parallel H((K' \oplus ipad) \parallel M))$$

dengan:

- K : secret key (kunci rahasia)
- M : message (pesan)

- K' : kunci yang sudah dinormalisasi menjadi 1 blok (untuk SHA-256 = 64 byte)
- H : fungsi hash SHA-256
- ipad : byte 0x36 diulang 64 kali
- opad : byte 0x5C diulang 64 kali
- $\oplus$: XOR per byte (*bitwise XOR*)
- $\|$: konkatenasi (gabung biner)

Khusus untuk SHA-256:

- Ukuran blok internal B = 64 byte (512 bit)
- Output hash H(x) selalu 32 byte (256 bit)

Intuisinya:

- **Inner hash**: hash dari (K' $\oplus$ ipad) yang digabung dengan pesan MMM.
- **Outer hash**: hash dari (K' $\oplus$ opad) yang digabung dengan *inner hash*.

1. Data Contoh: Kunci dan Pesan

Gunakan Test Case 1 RFC 4231**.**

a. Kunci K

Kunci terdiri dari 20 byte**,** masing-masing bernilai 0x0B:

$$K = \{0B\ 0B\ 0B\ \ldots\ 0B\}20\ byte$$

Jadi:

- |K|=20 byte

b. Pesan M

Pesan adalah string "Hi There".

Konversi ASCII → heksadesimal:

Table 18 Hasil Konversi Pesan

Karakter	**ASCII (hex)**
H	48
i	69
(spasi)	20
T	54
h	68
e	65
r	72

e	65

Sehingga:

$$M = 48\ 69\ 20\ 54\ 68\ 65\ 72\ 65$$

Panjang pesan:

- |M|=8 byte
- Dalam bit: l=8×8=64 bit

2. Normalisasi Kunci: Membentuk K′ (64 Byte)

Aturan normalisasi HMAC:

- Jika |K|>B: hash dulu K dengan SHA-256 (jadi 32 byte), kemudian pad dengan 0x00 sampai 64 byte.
- Jika |K|≤B : langsung pad K dengan 0x00 sampai panjangnya 64 byte.

Pada contoh ini:

- |K|=20 byte
- B=64 byte

  → |K| ≤ B, jadi kita cukup menambah nol**.**

$$K' = K \ \| \ \{00\ 00\ \ldots\ 00\}\ 44\ byte$$

Secara eksplisit:

- 20 byte pertama: 0x0B
- 44 byte terakhir: 0x00

Total panjang: 64 byte**.**

K′ adalah versi "blok penuh" dari kunci, supaya bisa dioperasikan dengan ipad dan opad yang juga 64 byte.

3. Membentuk ipad dan opad (64 Byte)

Didefinisikan:

- ipad = 0x36 diulang 64 kali
- opad = 0x5C diulang 64 kali

Per byte:

- 0x36 = $0011\ 0110_2$
- 0x5C = $0101\ 1100_2$

Kedua *pad* ini nanti di-XOR dengan K′ untuk membuat inner key dan outer key**.**

4. Bagian Dalam (Inner Part HMAC)

a. Menghitung Inner Key: K_{inner}=K′⊕ipad

Kita XOR byte per byte antara K′ dan ipad:

- **20 byte pertama: 0x0B ⊕ 0x36**

  Contoh 1 byte:

  - 0x0B = $0000\ 1011_2$
  - 0x36 = $0011\ 0110_2$

  XOR:

  0000 1011 (0B)

  ⊕ 0011 0110 (36)

  = 0011 1101 (3D)

  Jadi 20 byte pertama hasil XOR = 0x3D semuanya.

- **44 byte terakhir: 0x00 ⊕ 0x36**

  - 0x00 = $0000\ 0000_2$
  - 0x36 = $0011\ 0110_2$

  XOR:

  0000 0000 (00)

  ⊕ 0011 0110 (36)

  = 0011 0110 (36)

  Jadi 44 byte terakhir hasil XOR = 0x36 semuanya.

- **Bentuk lengkap K_{inner}**

  $$K_{inner} = \{3D\ 3D\ \ldots\ 3D\}\ 20\ byte\ ;\ \{36\ 36\ \ldots\ 36\}\ 44\ byte$$

  Total tetap 64 byte.

  Kita "mencampur" kunci dengan ipad untuk membuat kunci khusus yang hanya dipakai di bagian *inner*.

b. Menyusun Inner Message: (K′⊕ipad) ‖ M

Inner message:

$$InnerMsg = K_{inner} \parallel M$$

- 64 byte pertama: K_{inner}

  (20 × 3D, lalu 44 × 36)

- Diikuti 8 byte pesan M:

48 69 20 54 68 65 72 65

Secara skematis:

$\{3D\ 3D \ldots 3D\ 36\ 36 \ldots 36\}\ 64\ byte\ inner\ key;\ \ \{48\ 69\ 20\ 54\ 68\ 65\ 72\ 65\}\ 8\ byte\ M$

Panjang sebelum padding SHA-256:

$$| InnerMsg | = 64 + 8 = 72\ byte$$

c. Menghitung Inner Hash: H_{inner}=SHA256(InnerMsg)

Pada tahap ini dijalankan **proses SHA-256 lengkap** terhadap InnerMsg:

1. Konversi InnerMsg ke deretan bit.
2. Tambah padding SHA-256:
   - Tambah 1 bit '1' (dalam implementasi: byte 0x80).
   - Tambah nol hingga panjang ≡ 448 (mod 512).
   - Tambah 64 bit yang berisi panjang pesan asli (72 byte = 576 bit).
3. Bagi jadi blok 512 bit (masing-masing 64 byte).
4. Untuk tiap blok:
   - Parse jadi 16 word awal W0…W15
   - Perluas jadi W0…W63 dengan fungsi σ_0,σ_1
   - Jalankan 64 *round* dengan fungsi Σ_0,Σ_1,Ch,Maj,T_1,T_2.
   - Update register a,b,c,d,e,f,g,ha,b,c,d,e,f,g,ha,b,c,d,e,f,g,h.

   Melakukan semua 64 *round* secara manual akan sangat panjang, jadi biasanya di laporan cukup dijelaskan rumus + langkah umum, lalu nilai akhirnya diambil dari implementasi program.

   Untuk test vector ini, hasil resmi:

$$H_{inner} = SHA256(InnerMsg) =$$
$$B8\ 55\ B6\ 08\ 82\ 19\ 35\ 63\ 45\ EB\ 59\ BC\ 4A\ 38\ DB\ 9C$$
$$44\ C2\ B4\ E9\ A7\ 09\ 21\ 0B\ 5E\ 05\ 27\ 37\ C8\ 2C\ 24\ 17$$

   Itu adalah 32 byte (256 bit). Nilai ini menjadi "pesan" untuk hash bagian luar.

5. Bagian Luar (Outer Part HMAC)

a. Menghitung Outer Key: K_{outer}=K′⊕opad

Sekarang kita XOR K′ dengan opad.

- 20 byte pertama: 0x0B ⊕ 0x5C

- 0x0B = 0000 1011_2
- 0x5C = 0101 1100_2

XOR:

0000 1011 (0B)

⊕ 0101 1100 (5C)

= 0101 0111 (57)

Jadi **20 byte pertama** = 0x57 semua.

❖ 44 byte terakhir: 0x00 ⊕ 0x5C

- 0x00 = 0000 0000_2
- 0x5C = 0101 1100_2

XOR:

0000 0000 (00)

⊕ 0101 1100 (5C)

= 0101 1100 (5C)

Jadi 44 byte terakhir = 0x5C semua.

❖ Bentuk lengkap K_{outer}

$$K_{outer} = \{57\ 57\ \dots\ 57\}\ 20\ byte\ ;\ \{5C\ 5C\ \dots\ 5C\}\ 44\ byte$$

Total 64 byte.

Ini adalah "kunci luar" yang dipakai untuk membungkus hasil inner hash. Ide HMAC: *pesan* selalu "diselimuti" kunci di bagian dalam dan luar.

b. Menyusun Outer Message: (K′⊕opad) ‖ H_{inner}

Outer message:

$$OuterMsg = K_{outer} \parallel H_{inner}$$

- 64 byte pertama: 20 × 57, lalu 44 × 5C
- 32 byte berikutnya: nilai H_{inner} di atas

Panjang sebelum padding SHA-256:

$$|\ OuterMsg\ | = 64 + 32 = 96\ byte$$

c. Hasil Akhir: HMAC(K,M)=SHA256(OuterMsg)

Terakhir, jalankan lagi fungsi SHA-256 penuh terhadap OuterMsg (padding, pembentukan blok, 64 *round*, dst.).

Hasil akhirnya adalah:

$$HMAC_{SHA-256}(K, M) =$$
$$B0\ 34\ 4C\ 61\ EE\ F7\ 35\ B1\ 36\ 7E\ 97\ 8C\ 55\ 89\ 89\ DF$$
$$1A\ 04\ 15\ DB\ 6B\ 8B\ 42\ 3D\ 1F\ 31\ 41\ 02\ 83\ 10\ DA\ 8D$$

Jika ingin ditulis sebagai satu string hex 64 digit:

$B0344C61EEF735B1367E978C558989DF1A0415DB6B8B423D1F3141028310DA8D$

Itulah nilai **HMAC–SHA-256** untuk:

- Kunci: 20 byte 0x0B
- Pesan: "Hi There"

2.2.3 Enkripsi (AES)

Advanced Encryption Standard (AES) merupakan algoritma kriptografi simetris yang digunakan secara luas pada sistem komunikasi modern, termasuk perangkat IoT. AES bekerja dengan menggunakan kunci rahasia (*secret key*) untuk melakukan proses enkripsi dan dekripsi data. Salah satu varian yang paling banyak digunakan dalam dunia IoT adalah AES-128, karena ukuran kuncinya (128 bit) dinilai paling efisien dalam hal keamanan dan kebutuhan komputasi. AES-128 memiliki struktur ronde yang relatif ringan namun tetap mampu memberikan tingkat keamanan tinggi terhadap serangan brute-force, cryptanalysis, maupun manipulasi data[27]. Karena bentuknya simetris, kunci yang digunakan untuk mengamankan data juga digunakan untuk membukanya kembali, sehingga sangat sesuai untuk aplikasi tertutup seperti komunikasi firmware antara server dan perangkat IoT. Hal ini menjadikan AES-128 sering diterapkan pada sistem *Over-The-Air (OTA) firmware update*, dengan tujuan menjaga kerahasiaan firmware agar tidak disalin, dimodifikasi, atau dianalisis oleh pihak yang tidak berwenang[28].

Pada perangkat IoT bersumber daya rendah seperti ESP32, AES-128 memiliki keuntungan tambahan yaitu dukungan akselerasi hardware. ESP32 memiliki modul kriptografi internal yang dapat mempercepat proses enkripsi/dekripsi AES tanpa membebani CPU secara signifikan. Implementasi ini juga dapat dilakukan secara *streaming*, sehingga firmware tidak perlu disimpan penuh di RAM saat didekripsi, melainkan langsung diolah per blok data kemudian ditulis ke memori flash. Dengan demikian, AES-128 tidak hanya menjaga

kerahasiaan firmware pada OTA, tetapi juga efisien dari sisi pemanfaatan memori, CPU, dan waktu eksekusi[27].

1). Konsep Singkat AES-128

AES-128 adalah algoritma kriptografi simetris berbasis blok dengan:

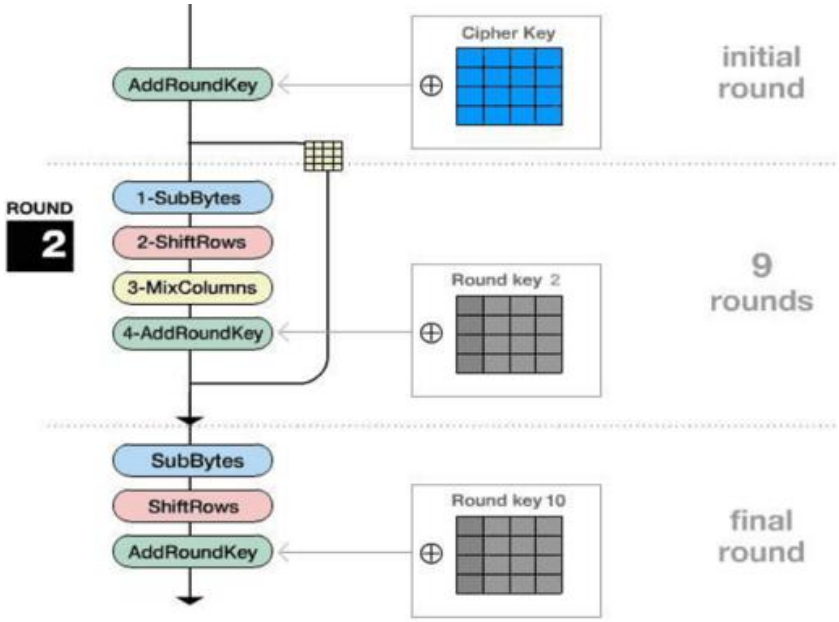
- Ukuran blok tetap: 128 bit = 16 byte
- Kunci: 128 bit (pada AES-128)
- Representasi data: 16 byte disusun dalam matriks 4×4 yang disebut *state*
- Semua operasi dilakukan dalam medan hingga GF(2^8)
  (setiap byte dianggap elemen polinomial derajat < 8)

Transformasi utama per *round*:

1. AddRoundKey – XOR state dengan round key
2. SubBytes – substitusi non-linier per byte menggunakan S-box
3. ShiftRows – rotasi baris secara siklik
4. MixColumns – operasi linier kolom di GF(2^8)

Pada AES-128:

- Terdapat 10 round.
- Round 0: hanya AddRoundKey.
- Round 1–9: SubBytes → ShiftRows → MixColumns → AddRoundKey.
- Round 10 (terakhir): SubBytes → ShiftRows → AddRoundKey (tanpa MixColumns).



Gambar 2. 2 Alur Enkripsi algoritma AES

2). Contoh Enkripsi AES-128

Table 19 Notasi Dasar yang Dipakai

Simbol / Istilah	Arti Singkat
State	Matriks 4×4 byte (16 byte) yang berisi data intermediate AES
⊕	XOR bitwise (penjumlahan mod 2 per bit)
SubBytes	Substitusi tiap byte memakai S-Box AES
ShiftRows	Pergeseran baris secara siklik ke kiri
MixColumns	Operasi linier per kolom di GF(2^8)
AddRoundKey	XOR state dengan round key (kunci pada round tersebut)
GF(2^8)	Medan hingga 256 elemen, tiap byte dianggap polinomial mod $x^8+x^4+x^3+x+1$

A. Data Contoh dan Pembentukan State

Table 20 Konversi Plaintext dan Key ke Hex

Jenis	Karakter	Hex
Plaintext	**A B C D**	**41 42 43 44**
	E F G H	45 46 47 48
	I J K L	49 4A 4B 4C
	M N O P	4D 4E 4F 50

Key	Q R S T	51 52 53 54
	U V W X	55 56 57 58
	Y Z a b	59 5A 61 62
	c d e f	63 64 65 66

Keterangan:

- Setiap karakter ASCII dikonversi ke representasi heksadesimal (1 byte).
- AES bekerja pada 16 byte (128 bit) untuk satu blok.

Table 21 Penyusunan Plaintext ke State 4×4 (per kolom)

	Col0	**Col1**	**Col2**	**Col3**
Row0	41	42	43	44
Row1	45	46	47	48
Row2	49	4A	4B	4C
Row3	4D	4E	4F	50

Table 22 Penyusunan Key ke State 4×4

	Col0	**Col1**	**Col2**	**Col3**
Row0	51	52	53	54
Row1	55	56	57	58
Row2	59	5A	61	62
Row3	63	64	65	66

Keterangan:

- Data AES selalu dipresentasikan sebagai **state** 4×4 byte.
- Pengisian dilakukan **per kolom** (bukan per baris).

B. Tahap 0 – AddRoundKey (Initial Round)

Operasi:

$$State = PlaintextState \oplus KeyState$$

Table 23 Contoh XOR per byte (Posisi [Row0,Col0] dan [Row0,Col1])

Posisi	**Plain (hex)**	**Plain (biner)**	**Key (hex)**	**Key (biner)**	**XOR (biner)**	**Hasil (hex)**
(0,0)	41	0100 0001	51	0101 0001	0001 0000	10

(0,1)	42	0100 0010	52	0101 0010	0001 0000	10

XOR dilakukan sama untuk semua posisi byte.

Table 24 State setelah AddRoundKey Awal

	Col0	**Col1**	**Col2**	**Col3**
Row0	10	10	10	10
Row1	10	10	10	10
Row2	10	10	2A	2E
Row3	2E	2A	2A	36

Keterangan:

- ⊕ adalah XOR bit-wise: 1⊕1=0, 1⊕0=1, 0⊕1=1, 0⊕0=0.
- AddRoundKey mencampur plaintext dengan kunci sehingga state awal sudah "terikat" dengan key.

C. Tahap 1 – SubBytes

SubBytes:

Setiap byte b diganti dengan SBox[b].

Table 25 Contoh Lookup S-Box

Input (hex)	**Baris (x)**	**Kolom (y)**	**SBox[x][y]**	**Output (hex)**
2A	2	A	E5	E5
10	1	0	C6	C6

State sebelum SubBytes = Tabel 5.

Setelah semua byte melewati S-Box, diperoleh:

Table 26 State setelah SubBytes

	Col0	**Col1**	**Col2**	**Col3**
Row0	C6	C6	C6	C6
Row1	C6	C6	C6	C6
Row2	C6	C6	E5	FB
Row3	FB	E5	E5	42

Keterangan:

- S-Box didesain untuk memberikan non-linearitas dan keamanan terhadap serangan kripto-analisis.
- Tidak ada operasi aritmetika eksplisit di sini, hanya tabel substitusi.

D. Tahap 2 – ShiftRows

Aturan ShiftRows (kiri siklik):

- Row0: tidak digeser
- Row1: geser kiri 1 byte
- Row2: geser kiri 2 byte
- Row3: geser kiri 3 byte

Table 27 Proses ShiftRows

Baris	Sebelum	Operasi	Sesudah
Row0	C6 C6 C6 C6	tidak digeser	C6 C6 C6 C6
Row1	C6 C6 C6 C6	kiri 1	C6 C6 C6 C6
Row2	C6 C6 E5 FB	kiri 2	E5 FB C6 C6
Row3	FB E5 E5 42	kiri 3 (kanan 1)	E5 E5 42 FB

Table 28 State setelah ShiftRows

	Col0	Col1	Col2	Col3
Row0	C6	C6	C6	C6
Row1	C6	C6	C6	C6
Row2	E5	FB	C6	C6
Row3	E5	E5	42	FB

Keterangan:

- ShiftRows menggeser data antar kolom → membuat efek difusi (bit-bit menyebar ke posisi berbeda pada round selanjutnya).

E. Tahap 3 – MixColumns

MixColumns bekerja per kolom. Matriks tetap:

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

Untuk kolom $[a_0,a_1,a_2,a_3]^T$:

- Row1 = 02·a0 ⊕ 03·a1 ⊕ a2 ⊕ a3
- Row2 = a0 ⊕ 02·a1 ⊕ 03·a2 ⊕ a3
- Row3 = a0 ⊕ a1 ⊕ 02·a2 ⊕ 03·a3
- Row4 = 03·a0 ⊕ a1 ⊕ a2 ⊕ 02·a3

a. Contoh Kolom [C6, C6, E5, E5]^T

Anggap salah satu kolom seperti ini, kita hitung 02·C6 dan 03·C6.

Table 29 Perhitungan 02·C6 dan 03·C6 di GF(2^8)

Langkah	Keterangan	Nilai
C6	Byte awal	C6
C6 (biner)	$1100\ 0110_2$	–
02·C6	kali x → $x^8 + x^7 + x^3 + x^2\ mod\ (x^8 + x^4 + x^3 + x + 1)$	97
97 (biner)	$1001\ 0111_2$	–
03·C6	02·C6 ⊕ C6 = 97 ⊕ C6	51

b. Kombinasi dalam satu baris (Row1 kolom ini)

$$Row1 = 02 \cdot a0 \oplus 03 \cdot a1 \oplus a2 \oplus a3$$
$$= 97 \oplus 51 \oplus E5 \oplus E5$$
$$= 97 \oplus 51 \oplus 00$$
$$= C6$$

Dengan langkah serupa untuk semua baris dan semua kolom, hasil akhirnya (sesuai contohmu) adalah:

Table 30 State setelah MixColumns

	Col0	Col1	Col2	Col3
Row0	C6	D8	42	FB
Row1	80	A2	42	FB
Row2	E5	D9	51	81
Row3	A3	BD	D5	BC

Keterangan:

- MixColumns adalah transformasi linier di GF(2^8) yang mencampur byte satu kolom, menambah difusi antar baris.
- Operasi 02·x, 03·x, dll. dilakukan dengan aturan GF(2^8) dan polinomial reduksi AES.

F. Tahap 4 – AddRoundKey (Round Berikutnya)

Untuk ilustrasi, round key di contoh ini = key awal (asumsi sederhana).

$$State = State\_MixColumns \oplus Key$$

Table 31 Contoh XOR AddRoundKey (beberapa posisi)

Posisi	State sebelum	Key	XOR (hasil)
(0,0)	C6	51	97
(0,1)	D8	52	8A
(0,2)	42	53	11
(0,3)	FB	54	AF

XOR dilakukan ke seluruh 16 byte.

Table 32 State setelah AddRoundKey (round ini)

	Col0	Col1	Col2	Col3
Row0	97	8A	11	AF
Row1	D5	F4	15	A3
Row2	BC	83	30	E3
Row3	C0	D9	B0	DA

Keterangan:

- AddRoundKey dijalankan di setiap round dengan round key yang berbeda (hasil key schedule).
- Karena contoh ingin fokus ke alur, angka key disederhanakan tetap sama.

G. Round Berikutnya: SubBytes → ShiftRows → AddRoundKey

a. SubBytes (Round Berikutnya)

Setiap byte pada Tabel 13 masuk ke S-Box lagi.

Hasil (sesuai data contoh):

Table 33 State setelah SubBytes (round berikutnya)

	Col0	Col1	Col2	Col3
Row0	88	7E	82	79
Row1	03	BF	59	0A
Row2	65	EC	04	11
Row3	BA	35	E7	57

Keterangan:

- Proses lookup S-Box sama seperti di Tahap 1, tapi nilai input-nya sekarang berasal dari state hasil round sebelumnya.

b. ShiftRows (Round Berikutnya)

Aturan sama:

- Row0: tetap
- Row1: geser kiri 1
- Row2: geser kiri 2
- Row3: geser kiri 3

Table 34 ShiftRows (round berikutnya)

Baris	Sebelum	Sesudah (kiri siklik)
Row0	88 7E 82 79	88 7E 82 79
Row1	03 BF 59 0A	BF 59 0A 03
Row2	65 EC 04 11	04 11 65 EC
Row3	BA 35 E7 57	57 BA 35 E7

Table 35 State setelah ShiftRows (round berikutnya)

	Col0	Col1	Col2	Col3
Row0	88	7E	82	79
Row1	BF	59	0A	03
Row2	04	11	65	EC
Row3	57	BA	35	E7

c. AddRoundKey (Final dalam Contoh)

XOR dengan key:

	Col0	Col1	Col2	Col3
Row0	51	52	53	54
Row1	55	56	57	58
Row2	59	5A	61	62
Row3	63	64	65	66

Table 36 Contoh XOR pada AddRoundKey final

Posisi	State (ShiftRows)	Key	XOR (hasil)
(0,0)	88	51	D9
(0,1)	7E	52	2C
(2,2)	65	61	04

Table 37 State akhir (Ciphertext blok contoh)

	Col0	Col1	Col2	Col3
Row0	D9	2C	D1	2D
Row1	EA	0F	5D	5B
Row2	5D	4B	04	8E
Row3	34	DE	50	81

Keterangan:

- Matriks di atas adalah ciphertext untuk blok plaintext "ABCDEFGHIJKLMNOP" dengan kunci "QRSTUVWXYZabcdef" dalam contoh yang dipotong ini (hanya sebagian round yang ditulis eksplisit).
- Di AES-128 asli ada total 10 round (Round 0 + 9 full round + 1 final tanpa MixColumns).

H. Contoh Pola Dekripsi AES-128 (Ringkas + Detail 1 Kolom)

Table 38 Perbandingan urutan enkripsi vs dekripsi

Enkripsi (per round)	Dekripsi (urutan kebalikan)
Round 0: AddRoundKey	–
Round 1–9: SubBytes → ShiftRows → MixColumns → AddRoundKey	Round 10: InvAddRoundKey → InvShiftRows → InvSubBytes
Round 10: SubBytes → ShiftRows → AddRoundKey	Round 9–2: InvAddRoundKey → InvMixColumns → InvShiftRows → InvSubBytes
	Round 1: InvAddRoundKey → InvShiftRows → InvSubBytes → AddRoundKey[0]

Keterangan:

- Dekripsi memakai operasi invers: InvSubBytes, InvShiftRows, InvMixColumns tapi tetap pakai XOR (AddRoundKey/InvAddRoundKey) dengan round key yang berbeda urutan.

a. InvAddRoundKey

InvAddRoundKey = AddRoundKey, karena:

$$(x \oplus k) \oplus k = x$$

Contoh (dari state setelah MixColumns):

Tabel 20. InvAddRoundKey (mengembalikan state sebelum AddRoundKey)

	Col0	**Col1**	**Col2**	**Col3**
Row0	C6	D8	42	FB
Row1	80	A2	42	FB
Row2	E5	D9	51	81
Row3	A3	BD	D5	BC

XOR dengan key → hasil:

	Col0	**Col1**	**Col2**	**Col3**
Row0	97	8A	11	AF
Row1	D5	F4	15	A3
Row2	BC	83	30	E3
Row3	C0	D9	B0	DA

Keterangan:

- Ini adalah state yang sama dengan hasil AddRoundKey pada tahap 4 enkripsi.

b. InvMixColumns – Contoh Lengkap 0E·97

InvMixColumns memakai matriks invers:

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

Untuk konstanta 02, 04, 08 digunakan fungsi xtime(x):

- Jika MSB(x) = 0 → xtime(x) = x << 1

- Jika MSB(x) = 1 → xtime(x) = (x << 1) ⊕ 0x1B

Lalu:

- $09 \cdot x = 08 \cdot x \oplus x$
- $0B \cdot x = 08 \cdot x \oplus 02 \cdot x \oplus x$
- $0D \cdot x = 08 \cdot x \oplus 04 \cdot x \oplus x$
- $0E \cdot x = 08 \cdot x \oplus 04 \cdot x \oplus 02 \cdot x$

Tabel 21. Perhitungan detail 0E·97

Langkah	Nilai (hex)	Biner / Keterangan
x	97	$1001\ 0111_2$ (MSB=1)
02·97	35	xtime(97): shift kiri + XOR 1B
04·97	6A	xtime(35)
08·97	D4	xtime(6A)
0E·97	D4 ⊕ 6A ⊕ 35	= 8B

Keterangan:

- Semua operasi ini terjadi di GF(2^8), sehingga setiap hasil selalu direduksi ke 1 byte lewat aturan xtime dan XOR.

c. InvShiftRows dan InvSubBytes

- **InvShiftRows**: kebalikan ShiftRows
  - Row1: geser kanan 1
  - Row2: geser kanan 2
  - Row3: geser kanan 3
- **InvSubBytes**: menggunakan tabel Inverse S-Box (InvSBox), lookup sama seperti SubBytes tapi dengan tabel invers.

Dengan mengulang pola:

- Round 10: InvAddRoundKey → InvShiftRows → InvSubBytes
- Round 9–2: InvAddRoundKey → InvMixColumns → InvShiftRows → InvSubBytes
- Round 1: InvAddRoundKey → InvShiftRows → InvSubBytes → AddRoundKey[0]

maka state akan kembali ke plaintext "ABCDEFGHIJKLMNOP".

2.3 Internet of Things (IoT) dan Firmware

2.3.1 Konsep IoT dan Peran Firmware

Internet of Things (IoT) adalah konsep di mana berbagai perangkat fisik seperti sensor, mikrokontroler, dan actuator, saling terhubung melalui jaringan internet untuk bertukar data dan menjalankan fungsi tertentu secara otomatis. Dalam arsitektur IoT, terdapat tiga komponen utama: perangkat keras (hardware**)** sebagai pengumpul data dan eksekutor, firmware sebagai pengendali logika kerja perangkat, dan server atau cloud sebagai pusat penyimpanan dan pengolahan data.

Firmware memiliki peran krusial karena berfungsi sebagai "otak" dari perangkat IoT yang menghubungkan antara sistem fisik dengan platform digital[29]. Ia bertanggung jawab mengatur komunikasi dengan jaringan, membaca sensor, mengirimkan data ke server, serta menerima pembaruan instruksi dari sistem pusat. Tanpa firmware yang optimal, perangkat tidak akan mampu beroperasi secara mandiri dan sinkron dengan sistem backend.

Dalam konteks penelitian ini, firmware pada mikrokontroler ESP32 berfungsi mengatur proses pembaruan kode melalui Over-the-Air (OTA)[30]. Proses OTA memungkinkan pengembang untuk memperbarui program dari jarak jauh tanpa memerlukan koneksi fisik. Dengan demikian, firmware berperan sebagai komponen yang tidak hanya menjalankan sistem, tetapi juga memastikan perangkat dapat terus diperbarui untuk peningkatan performa maupun keamanan. Hal ini sangat penting terutama pada sistem IoT berskala besar yang tersebar di berbagai lokasi, di mana pembaruan manual tidak efisien lagi dilakukan[17][31].

2.3.2 Keamanan Firmware IoT dan Risiko Serangan

Meskipun OTA menawarkan kemudahan, mekanisme ini juga memperkenalkan risiko keamanan baru jika tidak dilindungi dengan baik. Firmware yang dikirimkan melalui jaringan berpotensi dimodifikasi atau diganti oleh pihak yang tidak berwenang. Jika firmware palsu berhasil diinstal, perangkat IoT dapat sepenuhnya diambil alih, menyebabkan kebocoran data, gangguan sistem, bahkan digunakan untuk aktivitas berbahaya seperti botnet[9][8].

Salah satu contoh serangan paling terkenal yang menyoroti kerentanan firmware IoT adalah serangan botnet Mirai pada tahun 2016. Dalam kasus ini, jutaan perangkat IoT seperti kamera CCTV dan router diretas melalui celah keamanan pada firmware, kemudian digunakan untuk melakukan Distributed Denial of Service (DDoS) berskala besar terhadap berbagai layanan internet global[9]. Serangan tersebut menunjukkan bahwa tanpa mekanisme autentikasi dan verifikasi firmware yang memadai, perangkat IoT dapat menjadi titik lemah dalam ekosistem jaringan yang luas[32].

Namun, dalam konteks penelitian ini, fokusnya tidak terletak pada peningkatan keamanan semata, melainkan pada analisis beban komputasi (computational cost) dari mekanisme keamanan yang digunakan. Setiap algoritma seperti SHA-256**,** HMAC-SHA256, dan AES menambah lapisan perlindungan berbeda, tetapi juga menimbulkan konsekuensi terhadap performa perangkat, seperti peningkatan penggunaan CPU, RAM, serta waktu proses OTA. Dengan demikian, tujuan penelitian ini adalah menilai efisiensi sumber daya ESP32 ketika ketiga algoritma tersebut diterapkan, sehingga dapat ditentukan mekanisme keamanan yang paling seimbang antara keamanan dan efisiensi operasional.

2.4 Mekanisme Over-the-Air (OTA) Update

2.4.1 Konsep dan Tahapan OTA

Over-the-Air (OTA) update merupakan mekanisme pembaruan firmware yang memungkinkan perangkat diperbarui secara jarak jauh melalui jaringan tanpa perlu koneksi fisik seperti kabel atau media eksternal. Dalam sistem Internet of Things (IoT), metode ini menjadi komponen penting untuk menjaga konsistensi dan keberlanjutan operasional berbagai perangkat yang tersebar di lapangan. Proses OTA pada penelitian ini diterapkan secara push-based, di mana server atau host secara langsung mengirimkan kode pembaruan ke perangkat ESP32 ketika pembaruan dibutuhkan, bukan melalui mekanisme request dari perangkat. Pendekatan ini dipilih agar pembaruan dapat dilakukan secara cepat dan terkontrol tanpa harus menunggu perangkat melakukan pemeriksaan berkala terhadap versi firmware di server[29][30].

Ketika pembaruan dimulai, server secara langsung mengirimkan firmware baru kepada perangkat ESP32 melalui koneksi jaringan Wi-Fi menggunakan protokol komunikasi HTTP. Firmware yang dikirimkan merupakan file biner hasil kompilasi program terbaru yang siap dijalankan pada perangkat target. Selama proses pengiriman, firmware tersebut melewati tahapan verifikasi atau dekripsi, tergantung pada mekanisme keamanan yang diterapkan. Dalam penelitian ini, tiga pendekatan berbeda digunakan: SHA-256, HMAC-SHA256, dan AES. SHA-256 digunakan untuk memverifikasi integritas firmware agar data yang diterima sesuai dengan aslinya, HMAC-SHA256 digunakan untuk memastikan firmware dikirim dari sumber resmi dengan autentikasi digital token, sedangkan AES digunakan untuk mendekripsi firmware terenkripsi yang dikirim oleh server guna menjaga kerahasiaan konten selama transmisi[8].

Setelah data diterima dan diverifikasi, ESP32 menulis firmware baru tersebut ke partisi flash sekunder menggunakan skema dua partisi (OTA_0 dan OTA_1). Mekanisme ini memungkinkan sistem untuk tetap berjalan dengan firmware lama sambil menyimpan firmware baru di partisi lain. Setelah penulisan selesai dan hasil verifikasi sesuai, perangkat akan mengatur ulang pointer boot ke partisi baru dan melanjutkan ke proses reboot. Pada saat reboot, firmware baru diinisialisasi dan dijalankan. Jika sistem mendeteksi ketidaksesuaian hash atau kegagalan eksekusi, ESP32 secara otomatis kembali ke firmware sebelumnya melalui mekanisme rollback, sehingga stabilitas perangkat tetap terjamin[31].

Seluruh proses tersebut mulai dari pengiriman hingga reboot, dianalisis secara kuantitatif dalam penelitian ini untuk mengukur dampaknya terhadap penggunaan sumber daya perangkat. Parameter yang diamati meliputi durasi pembaruan, kecepatan transfer data, penggunaan CPU (rata-rata dan maksimum), serta penggunaan RAM selama proses OTA berlangsung. Tujuan utama bukan untuk menguji efektivitas keamanan dari algoritma yang digunakan, tetapi untuk mengevaluasi beban komputasi dan efisiensi performa sistem OTA ketika mekanisme kriptografi diterapkan. Dengan demikian, penelitian ini memfokuskan perhatian pada bagaimana proses OTA memengaruhi performa perangkat bersumber daya terbatas seperti ESP32 dari sisi computational overhead, bukan dari sisi proteksi data murni[23][17].

2.4.2 Tantangan OTA pada Perangkat IoT Bersumber Daya Terbatas

Implementasi pembaruan firmware secara Over-the-Air (OTA) pada perangkat IoT seperti ESP32 menghadirkan tantangan besar karena keterbatasan sumber daya perangkat keras yang dimiliki. Mikrokontroler ESP32 hanya dilengkapi dengan 520 KB RAM, prosesor dual-core Xtensa LX6 dengan kecepatan hingga 240 MHz, serta kapasitas memori flash yang relatif kecil. Dalam kondisi seperti ini, setiap mekanisme tambahan yang diterapkan untuk meningkatkan keamanan seperti hashing dengan SHA-256, autentikasi dengan HMAC-SHA256, atau enkripsi dengan AES yang akan meningkatkan beban kerja pada CPU, memperbesar penggunaan RAM, memperpanjang durasi pembaruan firmware, serta meningkatkan secara keseluruhan. Oleh karena itu, efisiensi menjadi aspek krusial yang perlu diperhatikan dalam proses OTA di perangkat dengan sumber daya terbatas[17][33].

Selama pelaksanaan OTA, ESP32 menjalankan beberapa proses secara multitasking di bawah manajemen FreeRTOS (Real-Time Operating System). Sistem operasi ini membagi aktivitas ke dalam beberapa task yang berjalan secara paralel, seperti task komunikasi jaringan (Wi-Fi stack) untuk mempertahankan konektivitas dengan server, task transfer data firmware, task verifikasi hash atau dekripsi data, serta task sistem inti seperti idle task dan watchdog monitoring. Ketika algoritma kriptografi diaktifkan, terutama pada tahapan verifikasi dan instalasi, CPU harus melakukan perhitungan matematis kompleks seperti rotasi bit, operasi XOR, substitusi byte, dan permutasi blok data. Proses-proses tersebut menambah tekanan kerja pada CPU dan dapat menyebabkan peningkatan temperatur prosesor serta fluktuasi penggunaan RAM, karena ESP32 harus memproses data dalam buffer memori yang sangat terbatas[8][23].

Selain itu, selama proses OTA, ESP32 juga harus menyimpan dua salinan firmware secara bersamaan — yaitu firmware lama dan firmware baru — di dua partisi flash yang berbeda (misalnya OTA_0 dan OTA_1). Hal ini meningkatkan kebutuhan penyimpanan sementara selama proses pembaruan berlangsung. Pada perangkat dengan kapasitas flash kecil, kondisi ini dapat mempersempit ruang untuk fungsi lain dan berpotensi menyebabkan kegagalan update apabila ukuran firmware melebihi kapasitas yang disediakan. Masalah ini semakin signifikan apabila firmware yang dikirimkan sudah dalam bentuk terenkripsi, karena ukuran file biasanya sedikit lebih besar dibandingkan file tanpa enkripsi[31].

Dalam penelitian ini, sistem FreeRTOS digunakan untuk memantau dan mencatat statistik runtime yang berkaitan dengan penggunaan CPU dan memori selama proses OTA berlangsung. Data yang diambil mencakup nilai CPU usage rata-rata dan maksimum (per core), serta penggunaan RAM minimum, rata-rata, dan maksimum pada setiap mekanisme OTA. Hasil pengukuran ini memberikan gambaran yang akurat tentang bagaimana setiap algoritma kriptografi memengaruhi performa keseluruhan sistem. Misalnya, implementasi SHA-256 cenderung memberikan overhead CPU yang moderat dengan penggunaan memori rendah, sedangkan HMAC-SHA256 membutuhkan waktu pemrosesan lebih lama karena melibatkan dua tahap hashing dan penggunaan kunci rahasia. Sementara itu, AES menunjukkan peningkatan konsumsi RAM akibat kebutuhan buffer tambahan untuk proses dekripsi blok data, tetapi memiliki efisiensi yang relatif lebih baik dalam hal kecepatan dibandingkan HMAC-SHA256[33].

Dengan pendekatan tersebut, penelitian ini tidak hanya menilai keberhasilan proses pembaruan firmware, tetapi juga mengevaluasi trade-off antara aspek keamanan dan performa sistem. Hasil analisis membantu menentukan algoritma yang paling efisien untuk diterapkan pada pembaruan firmware OTA di perangkat bersumber daya terbatas seperti ESP32. Dengan demikian, fokus penelitian bukan pada peningkatan lapisan keamanan semata, melainkan pada analisis empiris dampak algoritma kriptografi terhadap efisiensi penggunaan sumber daya sistem, yang mencerminkan keseimbangan antara kebutuhan keamanan dan kemampuan perangkat keras di lingkungan IoT.

2.5 Analisis Penggunaan Sumber Daya (Performance Resource Usage)

2.5.1 Penggunaan RAM dan Flash

2.5.1.1 Penggunaan RAM

Selama proses pembaruan firmware Over-the-Air (OTA) pada perangkat IoT seperti ESP32, penggunaan RAM menjadi salah satu faktor penentu keberhasilan sistem. Hal ini disebabkan karena proses OTA tidak hanya melakukan transfer data, tetapi juga membutuhkan ruang memori sementara untuk menampung data firmware, melakukan perhitungan kriptografi, serta menjalankan berbagai task sistem operasi secara paralel[34]. ESP32 menggunakan FreeRTOS sebagai sistem operasi, sehingga selama OTA, RAM juga

dialokasikan untuk task Wi-Fi, task penulisan flash, task kriptografi, task sistem inti (idle, timer, scheduler), serta buffer komunikasi jaringan (HTTP/TCP)[17].

Pada mekanisme Plain OTA, penggunaan RAM relatif kecil karena perangkat hanya membutuhkan buffer OTA untuk menyimpan potongan firmware yang diunduh melalui HTTP sebelum ditulis ke partisi flash[35]. Tidak adanya proses hashing, autentikasi, maupun dekripsi membuat metode ini menjadi yang paling efisien dalam hal alokasi memori. Sebaliknya, ketika menggunakan SHA-256, perangkat harus menyediakan ruang tambahan untuk konteks hashing yang menyimpan state perhitungan hash sebelum diverifikasi dengan nilai hash dari server. Implementasi HMAC-SHA256 membutuhkan RAM lebih besar karena selain menyimpan konteks hash, perangkat juga harus menyimpan kunci rahasia serta buffer untuk proses inner dan outer hash.

Pada metode AES, penggunaan RAM menjadi lebih signifikan karena proses enkripsi/dekripsi harus dilakukan blok-per-blok menggunakan buffer dengan ukuran 16 byte beserta Initialization Vector (IV) atau nonce yang digunakan sebagai nilai awal counter. Selain itu, sebagian library kriptografi (misalnya mbedTLS) membutuhkan buffer internal tambahan untuk menyimpan key schedule, sehingga alokasi memori meningkat seiring dengan ukuran firmware dan intensitas operasi kriptografi. Kondisi ini memperlihatkan bahwa semakin kompleks mekanisme keamanan, semakin besar pula kebutuhan RAM selama OTA berlangsung[17][14].

Tantangan ini sejalan dengan temuan penelitian “Secure Over-the-Air Firmware Update Mechanism for Wireless Sensor Networks”, yang melaporkan bahwa mekanisme OTA aman pada perangkat IoT menimbulkan rata-rata overhead tambahan sekitar 3 KB RAM dan ~14 KB flash untuk fungsi keamanan berbasis kriptografi. Penelitian lain berjudul “Over-the-Air Firmware Updates for Constrained NB-IoT Devices” menyatakan bahwa keterbatasan kapasitas RAM menyebabkan beberapa algoritma tidak dapat diimplementasikan secara penuh pada perangkat IoT skala kecil, sehingga perlu pemilihan metode yang efisien agar perangkat tidak mengalami crash selama pembaruan[34]. Dengan demikian, penting untuk memastikan adanya headroom RAM, yaitu sisa minimum RAM

yang tersedia selama OTA, guna menjamin stabilitas sistem dan mencegah kegagalan pembaruan.

Table 39 Penggunaan RAM

Metode OTA	Komponen Tambahan RAM	Keterangan Beban RAM
Plain OTA	Buffer download firmware	Paling ringan, tidak ada kriptografi
SHA-256	Buffer + konteks hash	Beban sedang, hanya 1 tahap hashing
HMAC-SHA256	Buffer + konteks hash + secret key	Beban lebih tinggi, hashing ganda
AES-CTR	Buffer 16B + IV + internal key schedule	Beban tertinggi, dekripsi blok-per-blok

2.5.1.2 Penggunaan Flash

Selain RAM, flash memory merupakan komponen krusial dalam proses pembaruan firmware Over-the-Air (OTA) pada perangkat IoT seperti ESP32. Flash digunakan sebagai ruang penyimpanan permanen bagi firmware, konfigurasi sistem, serta file pendukung seperti manifest dan metadata keamanan[34]. Proses OTA yang aman mengharuskan perangkat memiliki cukup kapasitas flash untuk menyimpan dua versi firmware sekaligus serta library kriptografi tambahan, sehingga pengelolaan penggunaan flash harus diperhitungkan sejak tahap perancangan sistem.

Sebagian besar arsitektur OTA modern menggunakan skema dual-partition yang membagi flash menjadi dua image firmware, misalnya OTA_0 dan OTA_1. Firmware aktif dijalankan dari salah satu partisi, sementara firmware baru yang diunduh disimpan pada partisi cadangan. Setelah proses unduh dan validasi selesai (misalnya verifikasi hash SHA-256 atau verifikasi token HMAC), pointer boot loader akan diarahkan ke partisi yang baru diperbarui. Mekanisme ini memungkinkan rollback otomatis apabila proses pembaruan gagal, sehingga perangkat tetap dapat beroperasi dengan firmware lama tanpa korupsi sistem. Walau aman, pendekatan ini membutuhkan kapasitas flash minimal dua kali ukuran firmware ditambah overhead metadata, yang menjadi tantangan pada perangkat dengan flash kecil (misalnya 2–4 MB)[34].

Pada sistem OTA dengan keamanan, penggunaan flash meningkat karena adanya library kriptografi tambahan**,** terutama saat menggunakan hashing (SHA-256), autentikasi token (HMAC-SHA256), atau enkripsi firmware (AES-CTR). Library seperti mbedTLS menambahkan kode enkripsi, tabel substitusi (S-Box), vector inisialisasi/nonce, fungsi hash, dan struktur konteks ke dalam file binary. Hal ini menyebabkan ukuran final firmware meningkat signifikan dibanding metode Plain OTA. Selain itu, metadata manifest OTA dalam format JSON juga harus disimpan pada server dan sebagian dipasang pada flash perangkat, berisi informasi seperti nomor versi firmware, ukuran file, URL server, nilai hash, token, initialization vector (IV), dan flag metode keamanan yang digunakan. Walaupun ukurannya relatif kecil (beberapa ratus byte), metadata ini tetap perlu dipertimbangkan dalam keseluruhan alokasi flash[34][17].

Penelitian "Secure Over-the-Air Firmware Update Mechanism for Wireless Sensor Networks" melaporkan bahwa fitur OTA aman menimbulkan overhead flash rata-rata 14 KB hingga 22 KB tergantung algoritma keamanan yang digunakan. Laporan teknis Espressif (2023) juga menyatakan bahwa pengaktifan secure boot dan flash encryption pada ESP32 dapat meningkatkan ukuran firmware hingga 6%–15% akibat penambahan library enkripsi dan modul validasi bootloader[36]. Sementara itu, artikel "Innovative Firmware Update Method to Microcontrollers" (MDPI, 2021) menemukan bahwa efisiensi pembaruan firmware dapat ditingkatkan dengan meminimalkan blok data yang ditransfer serta mengoptimalkan partition switching untuk mengurangi penulisan flash yang tidak diperlukan. Temuan-temuan ini menunjukkan bahwa perancangan OTA yang aman harus mempertimbangkan keseimbangan antara kebutuhan ruang flash dan tingkat keamanan yang diterapkan.Untuk itu, perancangan OTA pada ESP32 memerlukan formula perhitungan sebagai berikut:

$$Firmware\ aktif\ +\ Firmware\ cadangan\ +\ Library\ keamanan\ +\ Metadata\ manifest$$

Jika salah satu komponen melebihi kapasitas flash yang tersedia, proses OTA berisiko gagal atau rollback tidak dapat dilakukan. Karena itu, pemilihan mekanisme keamanan harus disesuaikan dengan kapasitas flash perangkat agar pembaruan berjalan aman namun tetap efisien.

Table 40 Flah Terhadap Metode OTA

Metode OTA	Komponen Flash Tambahan	Keterangan Beban Flash
Plain OTA	Dua partisi firmware	Ukuran firmware terkecil, tidak ada library kriptografi
SHA-256	Library hash + konteks hashing	Ukuran firmware bertambah sedang
HMAC-SHA256	Library hash + penyimpanan kunci & token	Ukuran firmware lebih besar dari SHA-256
AES-CTR	Library AES + IV/nonce + key schedule	Peningkatan ukuran firmware signifikan

2.5.2 Beban CPU dan Komputasi Kriptografi

Selama proses pembaruan firmware OTA, perangkat IoT seperti ESP32 harus menjalankan beberapa operasi secara paralel di bawah manajemen FreeRTOS, termasuk koneksi jaringan Wi-Fi, pengunduhan firmware, penulisan data ke flash, dan eksekusi algoritma kriptografi (jika diterapkan). Ketika fitur keamanan aktif, beban CPU meningkat karena proses hashing, autentikasi token, atau dekripsi firmware membutuhkan perhitungan matematis kompleks dan pengolahan blok data dalam jumlah besar.

Evaluasi beban CPU pada OTA dilakukan menggunakan statistik runtime FreeRTOS, yang membagi beban kerja berdasarkan waktu eksekusi setiap task. Dua indikator utama digunakan:

Table 41 Beban CPU

Indikator CPU	Deskripsi
CPU Avg	Rata-rata penggunaan CPU selama keseluruhan proses OTA
CPU Max	Beban puncak (peak load) pada saat task kriptografi berjalan

Pengukuran ini penting karena peningkatan CPU tidak hanya memperlambat proses OTA, tetapi juga dapat mengganggu task penting lain seperti Wi-Fi stack atau watchdog timer. Dalam kondisi ekstrem, lonjakan CPU dapat menyebabkan disconnect Wi-Fi, kegagalan penulisan flash, hingga crash sistem**.**

Secara konseptual, pengaruh tiap mekanisme keamanan terhadap CPU dapat dibandingkan sebagai berikut:

Table 42 Pengaruh Tiap Mekanisme Keamanan terhadap CPU

Metode OTA	Beban CPU	Alasan Teknis
Plain OTA	Rendah	Tidak ada hashing atau enkripsi tambahan
SHA-256	Sedang	Hashing linear terhadap ukuran firmware; intensif tetapi stabil
HMAC-SHA256	Tinggi	Melakukan hashing ganda (inner + outer), serta memproses secret key
AES-CTR	Tinggi (spike)	Dekripsi blok per 16 byte; beban tinggi tetapi lebih cepat untuk data besar

Perbedaan utama antara HMAC-SHA256 dan AES-CTR terletak pada pola beban CPU**.** HMAC menghasilkan beban CPU tinggi yang merata dalam durasi lebih lama**,** karena hashing ganda harus dilakukan terhadap seluruh file firmware. Sebaliknya, AES-CTR menghasilkan lonjakan CPU singkat (spike) ketika memproses blok 16 byte, tetapi proses streaming dekripsinya lebih cepat jika data berukuran besar. Kondisi ini membuat AES lebih efisien untuk file OTA berukuran besar, namun berpotensi menjadi bottleneck pada perangkat dengan clock rendah atau jaringan tidak stabil.

Penelitian *"Securization of a FOTA Process for ESP32"* (Arjona et al., 2024) membuktikan bahwa HMAC-SHA256 menimbulkan overhead CPU tertinggi pada mikrokontroler karena melibatkan dua tahap hashing dan pengolahan kunci rahasia. Sementara AES menghasilkan lonjakan CPU yang lebih besar tetapi berlangsung lebih singkat**,** sehingga lebih efisien dalam throughput total pembaruan. Hal ini konsisten dengan temuan Pyrgas & Konstantinou (2023), yang menyatakan bahwa kriptografi simetris seperti AES lebih efisien daripada metode berbasis hashing untuk file besar, tetapi tetap memerlukan optimasi buffer streaming agar tidak menghambat task lain. Secara umum, dampak beban CPU terhadap sistem OTA dapat dirumuskan sebagai computational cost:

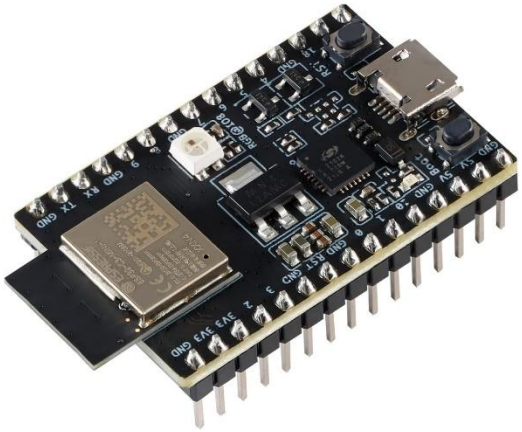
$$Computational\ Cost = CPU\ Avg + CPU\ Max\ Spike + Durasi\ Task\ Kriptografi$$

Semakin besar nilai cost, semakin berat mekanisme keamanan terhadap performa sistem OTA pada perangkat IoT. Oleh karena itu, pemilihan algoritma OTA harus mempertimbangkan keseimbangan antara keamanan, kapasitas perangkat, dan efektivitas waktu pembaruan.

2.6 Hardware

Pada sub-bab ini, akan dijelaskan mengenai perangkat keras yang digunakan dalam pengerjaan Tugas Akhir ini. Perangkat keras yang digunakan meliputi beberapa komponen utama yang berperan penting dalam keseluruhan sistem.

2.6.1 ESP-32



Gambar 2. 3 ESP32

(Sumber : https://m.media-amazon.com/images/I/61xIBhl8t-L._AC_SL1500_.jpg)

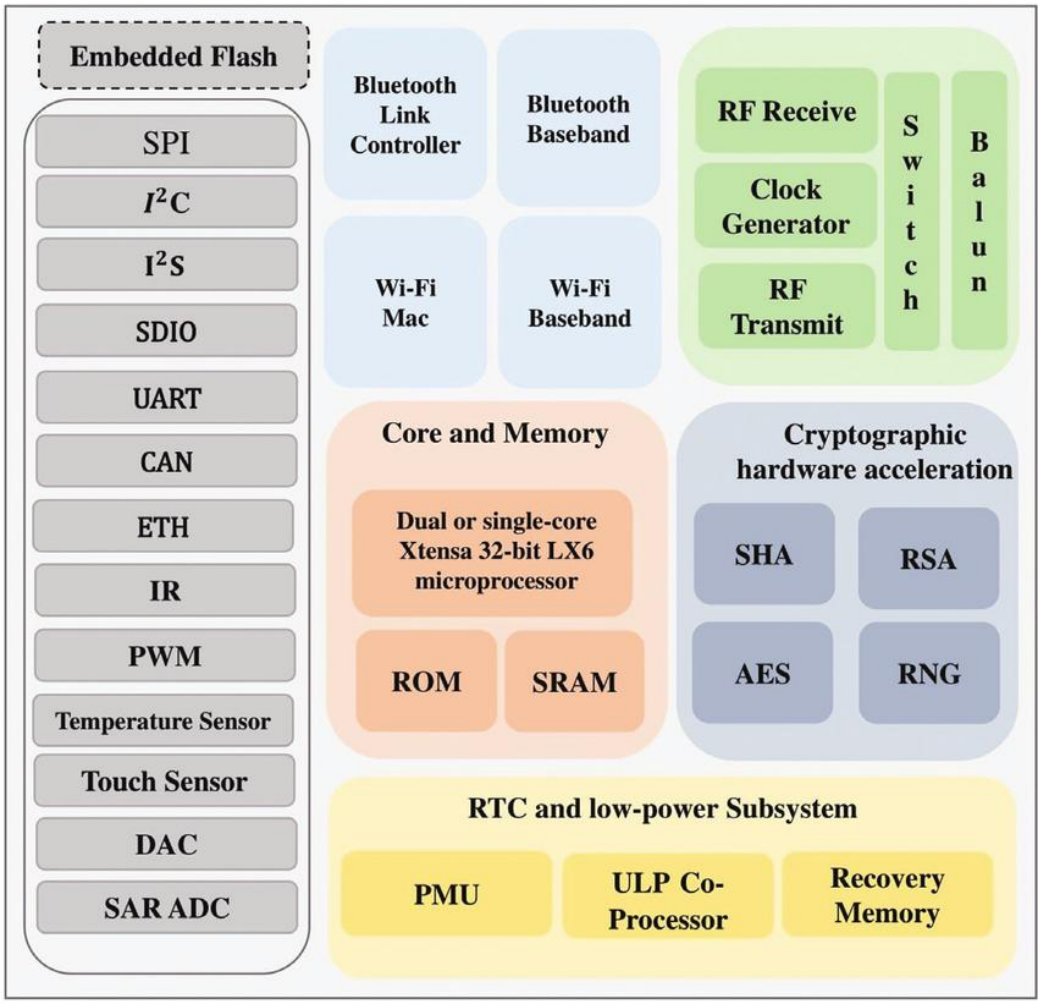
ESP32 merupakan mikrokontroler keluaran Espressif Systems yang sudah terintegrasi dengan modul Wi-Fi dan Bluetooth, sehingga sangat populer digunakan pada berbagai aplikasi *Internet of Things* (IoT). Salah satu keunggulan utama ESP32 adalah kemampuannya untuk melakukan pembaruan firmware secara *Over-the-Air* (OTA) menggunakan framework Arduino maupun ESP-IDF. Fitur ini memungkinkan perangkat yang sudah terpasang di lapangan untuk mendapatkan pembaruan firmware tanpa perlu koneksi fisik melalui kabel, sehingga lebih efisien dan praktis, terutama ketika perangkat tersebar dalam jumlah besar. Dengan adanya dukungan OTA, ESP32 dapat membantu pengembang memastikan perangkat tetap mendapatkan perbaikan bug, peningkatan performa, serta kompatibilitas dengan sistem terbaru. Hal ini menjadikan ESP32 bukan hanya sebagai mikrokontroler dengan konektivitas yang andal, tetapi juga sebagai platform yang mendukung siklus hidup perangkat IoT agar lebih terjaga dan berkelanjutan[37][38].

Tabel 2. 1 Spesifikasi Hardware

Kategori	Spesifikasi ESP32
Prosesor	Dual-core Xtensa LX6, hingga 240 MHz
Memori Internal	520 KB SRAM

Kategori	Spesifikasi ESP32
Memori Eksternal	Flash hingga 16 MB (tergantung modul)
Konektivitas	Wi-Fi 802.11 b/g/n (2.4 GHz), Bluetooth v4.2 & BLE
GPIO	± 34 pin GPIO (tergantung modul board)
Antarmuka	SPI, I2C, I2S, UART, PWM, ADC (12-bit), DAC (8-bit), CAN bus
Sensor Internal	Touch sensor, Hall effect sensor, sensor suhu
Keamanan	Akselerasi hardware untuk AES, SHA, HMAC-SHA
Mode Daya	Deep-sleep (<10 µA), Light-sleep, Active mode
Tegangan Operasi	2.2 – 3.6 V (umumnya 3.3 V)
Fitur Tambahan	Mendukung OTA update, enkripsi flash, secure boot

2.5.1.1 Arsitektur ESP32



Gambar 2. 4 Arsitektur ESP32

ESP32 merupakan mikrokontroler kelas IoT generasi terbaru yang memiliki kemampuan Wi-Fi + Bluetooth + akselerasi kriptografi di dalam satu chip. Arsitektur umumnya terdiri dari beberapa blok utama, yaitu komunikasi RF**,** subsistem prosesor & memori**,** modul komunikasi**,** serta perangkat kriptografi untuk keamanan tinggi.

1. Embedded Flash & Peripheral Interfaces

Pada sisi kiri diagram terlihat blok peripheral interface**,** yang berfungsi untuk komunikasi dengan perangkat eksternal:

Table 43 Embedded Flash & Peripheral Interfaces

Peripheral	**Fungsi**
SPI	Komunikasi data berkecepatan tinggi; dapat digunakan untuk flash external.
I²C	Komunikasi sensor (contoh: sensor suhu, kelembaban).
I²S	Digunakan untuk perangkat audio digital.

Peripheral	Fungsi
UART	Komunikasi serial (misalnya debugging, modul lain).
CAN	Digunakan dalam sistem otomotif.
Ethernet (ETH)	Koneksi jaringan kabel.
PWM	Kontrol kecepatan motor/LED brightness.
SAR ADC	Analog ke digital; membaca sensor analog.
DAC	Menghasilkan output analog.

Peripheral ini membuat ESP32 sangat fleksibel untuk aplikasi IoT, otomasi, sensor, dan sistem embedded lainnya.

2. Core and Memory

Bagian paling penting adalah Core Xtensa LX6, yaitu prosesor 32-bit yang tersedia dalam dual-core atau single-core**.**

Table 44 Core & Memory

Komponen	Fungsi
Xtensa LX6	CPU utama yang menjalankan program aplikasi dan kontrol sistem.
ROM	Menyimpan bootloader dan kode dasar sistem.
SRAM	Memory utama untuk program dan variabel selama runtime.

Kelebihan ESP32 adalah dual-core, sehingga dapat memisahkan tugas (misalnya satu core untuk Wi-Fi, satu untuk aplikasi).

3. Wi-Fi & Bluetooth Subsystem

ESP32 memiliki dua teknologi komunikasi nirkabel dalam satu chip**:**

Table 45 Wi-Fi & Bluetooth Subsystem

Teknologi	Fungsi
Wi-Fi MAC & Baseband	Bertanggung jawab untuk koneksi Wi-Fi 802.11 b/g/n.
Bluetooth Link Controller & Baseband	Mendukung Bluetooth Classic dan BLE (Low Energy).

Dengan combo Wi-Fi + BLE, ESP32 mendukung aplikasi hemat daya sekaligus konektivitas tinggi seperti IoT handheld, smart home, dan wearable device.

4. RF (Radio Frequency) Front-End

Blok RF menangani proses sinyal radio fisik, termasuk:

Table 46 RF (Radio Frequency) Front-End

Komponen	Fungsi
RF Transmit/Receive	Pemrosesan sinyal nirkabel untuk Wi-Fi/Bluetooth.
Clock Generator	Menghasilkan clock sinkronisasi RF.
Switch & Balun	Mendiferensiasikan jalur sinyal dan impedansi antena.

Ini memungkinkan ESP32 bekerja tanpa memerlukan modul RF eksternal.

5. Cryptographic Hardware Accelertion

Salah satu keunggulan ESP32 adalah hardware acceleration untuk enkripsi, seperti:

Table 47 Cryptographic Hardware Accelertion

Algoritma	Fungsi
SHA	Hashing data untuk integritas dan autentikasi (digunakan di OTA).
AES	Enkripsi simetris cepat (AES-128/192/256).
RSA	Kriptografi kunci publik (misalnya untuk SSL/TLS).
RNG (Random Number Generator)	Membuat bilangan acak yang aman untuk kriptografi.

Adanya modul kriptografi hardware membuat proses OTA, HTTPS, dan keamanan IoT menjadi jauh lebih cepat dan aman dibanding mikrokontroler tanpa akselerator kripto.

6. RTC & Low-Power Subsystem

Blok ini mendukung mode hemat energi:

Table 48 RTC & Low-Power Subsystem

Komponen	Fungsi
ULP Co-Processor	Prosesor kecil yang aktif saat CPU utama tidur (mengakses sensor rendah daya).
Recovery Memory	Menyimpan data saat mode low-power.

ESP32 sangat efisien untuk aplikasi IoT yang membutuhkan pemantauan sensor dengan baterai jangka panjang**.**

2.6.1.2 Freertos

Pada ESP32, FreeRTOS berfungsi sebagai sistem operasi waktu-nyata (RTOS) yang menjalankan tugas multitasking, manajemen memori, penjadwalan CPU, dan sinkronisasi antar proses. Dengan memori yang relatif besar dan dukungan dual-core pada ESP32, FreeRTOS memungkinkan Anda untuk membagi beban kerja antara berbagai tugas (tasks), interrupt service routines (ISRs), dan event handling secara efisien. Dalam konteks proyek OTA aman Anda, FreeRTOS memastikan bahwa proses pengunduhan, verifikasi hash (SHA-256 atau HMAC), dan dekripsi (AES-CTR) dijalankan dengan prioritas yang tepat tanpa mengganggu fungsi utama perangkat seperti koneksi WiFi atau komunikasi server.

FreeRTOS pada ESP32 menggunakan penjadwalan preemptive berdasarkan prioritas tugas: tugas dengan prioritas lebih tinggi dapat mengambil alih CPU dari tugas prioritas lebih rendah ketika siap. Ini memungkinkan proses kriptografi yang membutuhkan waktu (misalnya compute SHA-256) untuk diberikan prioritas cukup tinggi agar selesai dalam batas waktu yang wajar, sementara tugas background seperti monitoring atau komunikasi tetap berjalan. Selain itu, FreeRTOS pada ESP32 menyediakan mekanisme *tick* dan *idle task* yang memungkinkan CPU masuk ke mode penghematan daya ketika tidak ada tugas aktif, sehingga meningkatkan efisiensi total perangkat.

Dengan kehadiran FreeRTOS, pengukuran penggunaan CPU yang Anda lakukan pada OTA menjadi lebih bermakna: misalnya ketika Anda mencatat rata-rata CPU (%) dan maksimum CPU (%) selama pembaruan firmware, angka-angka tersebut mencerminkan seberapa besar beban CPU dari task FreeRTOS aktif (pengunduhan, kriptografi, flash write) dibanding task lain. Oleh sebab itu, memahami bagaimana FreeRTOS mengalokasikan CPU dan men-scheduling tugas sangat penting agar Anda bisa menafsirkan mengapa algoritma keamanan tertentu (misalnya HMAC-SHA256) memiliki penggunaan CPU rata-rata lebih tinggi.

Table 49 Konfigurasi dan Pemanfaatan FreeRTOS pada ESP32

Aspek FreeRTOS	Konfigurasi pada ESP32	Kegunaan dalam Proyek OTA Aman
Arsitektur CPU	Dual-core Xtensa LX6 (Core 0 & Core 1).	Pemisahan task intensif CPU (hash SHA256, HMAC, AES) dari task jaringan/wifi.

Aspek FreeRTOS	Konfigurasi pada ESP32	Kegunaan dalam Proyek OTA Aman
Jenis Scheduler	*Preemptive priority-based scheduler*.	CPU dialokasikan ke proses kriptografi dengan prioritas lebih tinggi saat update OTA.
Task Priority	Rentang 0 (idle) s.d. 25 (max).	Task hashing & decrypt diberi prioritas > koneksi agar tidak delay.
Context Switching	Otomatis berdasarkan tick interrupt.	Menjamin logging dan pengunduhan tetap berjalan *real-time*.
Tick Rate (ConfigTICK_RATE_HZ)	Umumnya 100–1000 Hz di ESP32-Arduino.	Semakin besar → respons lebih cepat, tetapi CPU usage meningkat.
Heap / Memory Management	Alokasi RAM menggunakan heap_4/heap_5.	Buffer OTA dan dekripsi dialokasikan dinamis tanpa menabrak task lain.
Synchronization	Mutex, Semaphore, Event Group.	Mencegah konflik akses memori saat flash write dan decrypt.
Interrupt Handling (ISR)	High-priority handler tanpa blocking.	Wi-Fi dan Flash I/O tetap berjalan saat task kriptografi berjalan.

2.6.2 Sensor Suhu

Gambar 2. 5 Sensor DS18B20

Sensor DS18B20 merupakan perangkat sensor temperature digital yang menggunakan protocol kabel tunggal (l-wire). Sensor DS18B20 menyediakan pembacaan suhu 9 bit hingga 12 bit. Komunikasi sensor ini dapat dilakukan melalui protocol bus satu kabel yang menggunakan satu jalur data berkomunikasi dengan mikroprosesor. Sensor suhu DS18B20 memiliki kelebihan yakni mudah di pasang dan deprogram serta memiliki tingkat kompatibilitas yang tinggi dengan berbagai system platform, termasuk system operasi computer, mikrokontroler dan system iot. Sensor DS18B20 memiliki kekurangan yaitu membutuhkan perangkat tambahan yang mengubah sinyal digital menjadi besar suhu[39].

Table 50 Spesifikasi Sensor DS18B20

Parameter	Nilai / Deskripsi
Model	Dallas Semiconductor DS18B20 (Waterproof Probe)
Tipe Sensor	Digital Temperature Sensor
Metode Komunikasi	One-Wire Protocol
Tegangan Operasi	3.0V – 5.5V
Resolusi	9–12 bit (pengaturan)

Rentang Suhu	-55°C hingga +125°C
Akurasi	±0.5°C pada rentang -10°C hingga +85°C
Tipe Output	Digital (Format 16-bit)
Protokol Bypass	Tidak membutuhkan ADC
Material Probe	Stainless Steel
Panjang Kabel	1 – 3 meter (tergantung varian)

2.7 Software

Pada sub-bab ini, dijelaskan mengenai perangkat lunak yang digunakan dalam pengembangan Tugas Akhir. Perangkat lunak yang digunakan bertujuan untuk menghubungkan perangkat keras, memproses data, serta menampilkan hasil secara real time.

2.7.1 Arduino IDE

Gambar 2. 6 Arduino IDE

(Sumber: https://blog.indobot.co.id/wp-content/uploads/2024/07/Fitur-Utama-Arduino-IDE.png)

Arduino IDE adalah perangkat lunak resmi yang digunakan untuk menulis, mengompilasi, dan mengunggah program ke papan mikrokontroler Arduino maupun mikrokontroler lain yang kompatibel, seperti ESP32 dan ESP8266. IDE ini menjadi jembatan antara pengguna dengan perangkat keras sehingga proses pengembangan menjadi lebih sederhana. Arduino IDE menggunakan bahasa pemrograman berbasis C/C++ dengan tambahan pustaka (library) khusus untuk memudahkan pengendalian perangkat keras seperti sensor, aktuator, dan modul komunikasi[40].

2.7.2 C++

C++ adalah bahasa pemrograman tingkat menengah yang dikembangkan oleh Bjarne Stroustrup pada tahun 1980-an sebagai pengembangan dari bahasa C. C++ menambahkan konsep pemrograman berorientasi objek ke dalam bahasa C, sehingga lebih fleksibel untuk membangun perangkat lunak kompleks sekaligus tetap efisien pada tingkat rendah untuk berinteraksi langsung dengan perangkat keras.

2.7.3 PHP

Gambar 2. 7 PHP

(Sumber : https://pngimg.com/uploads/php/php_PNG18.png)

PHP adalah bahasa pemrograman sisi *server* (server-side scripting) yang banyak [41]digunakan untuk membuat aplikasi web dinamis dan mengelola database. Dalam konteks OTA firmware update untuk perangkat IoT, PHP bisa berperan sebagai backend web service yang mengatur distribusi, otentikasi, dan manajemen firmware.

2.7.4 VS Code

Gambar 2. 8 Visual Studio Code

(Sumber : https://repository-images.githubusercontent.com/657248114/d3c7b91a-b285-4d1e-8429-5de1acc5f61e)

VS Code adalah editor kode sumber ringan namun powerful yang mendukung banyak bahasa pemrograman, termasuk PHP, C/C++, Python, dan JavaScript. VS Code banyak digunakan untuk mengembangkan aplikasi web, firmware, dan skrip backend karena fleksibilitas dan ekstensinya. Dalam proyek OTA IoT, VS Code berperan sebagai IDE (Integrated Development Environment) untuk menulis, menguji, dan debug kode[42].

2.8 Studi Terdahulu (Related Work)

2.8.1 Pengaplikasian Teknik Over The Air (OTA) untuk Smart Breaker ESP32

Penelitian yang dilakukan oleh Briliyanto dan Mashoedah (2024) di Universitas Negeri Yogyakarta mengaplikasikan teknik *Over-The-Air* (OTA) pada perangkat smart breaker berbasis ESP32. OTA digunakan untuk memperbarui kode program melalui protokol HTTP, di mana pengujian dilakukan pada dua perangkat secara bersamaan. Hasil penelitian menunjukkan bahwa kecepatan pembaruan firmware dipengaruhi oleh ukuran file firmware dan kualitas koneksi internet, sehingga semakin kecil ukuran file dan semakin baik koneksi maka waktu update semakin cepat. Studi ini menekankan pada efisiensi waktu update, namun belum membahas aspek penggunaan sumber daya perangkat seperti memori, prosesor, dan konsumsi energi. Hal ini menjadi celah penelitian yang relevan karena justru faktor *resource* menjadi isu utama pada perangkat IoT berskala kecil seperti ESP32[43].

2.8.2 Optimization Methods Regarding Battery Life and Write Speed to an SD-Card

Fokus penelitian mereka bukan pada OTA secara langsung, melainkan pada strategi memperpanjang umur baterai dan mempercepat operasi tulis ke media penyimpanan eksternal seperti SD-Card, termasuk perbandingan mode komunikasi (SPI vs SD bus 1-bit/4-bit), pengaruh frekuensi CPU, serta pemanfaatan sleep modes (modem-sleep, light-sleep, deep-sleep) terhadap arus dan durasi operasi tulis/baca. Temuan kunci mereka menunjukkan bahwa meskipun arus sesaat pada SD bus 4-bit lebih tinggi, total discharge baterai justru lebih rendah karena waktu tulis yang jauh lebih singkat; mereka juga merekomendasikan penulisan sesedikit mungkin dan pengaturan *clock* yang tepat untuk menurunkan konsumsi energi keseluruhan. Meskipun konteksnya berbeda, hasil penelitian ini relevan untuk studi OTA karena proses pembaruan firmware juga melibatkan aktivitas tulis data yang intensif ke memori internal; dengan demikian, prinsip optimasi *resource*

pada ESP32 dapat menjadi landasan penting dalam menganalisis *overhead* yang ditimbulkan oleh mekanisme keamanan pada OTA[44].

2.8.3 Pengembangan Perangkat Pembelajaran Mikrokontroler ESP32 Berbasis IoT & Bluetooth

Penelitian lain yang dilakukan di SMKN 2 Surabaya (2024) mengembangkan perangkat pembelajaran mikrokontroler berbasis ESP32 dengan dukungan IoT dan Bluetooth. Fokus utamanya adalah merancang trainer yang dapat digunakan untuk pembelajaran dasar pemrograman ESP32, termasuk uji coba terhadap latensi, kecepatan respon, dan keandalan konektivitas[45]. Walaupun penelitian ini tidak membahas mekanisme OTA maupun isu keamanan pembaruan firmware, studi ini menunjukkan bahwa penelitian ESP32 di Indonesia masih banyak berorientasi pada fungsi dasar dan penerapan praktis. Dengan demikian, belum banyak eksplorasi yang mendalam terkait analisis performa OTA, khususnya dalam hal konsumsi sumber daya perangkat ketika berbagai mekanisme keamanan diterapkan.

2.8.4 Update Time Delay pada Peralihan Fase Lampu Lalu Lintas Menggunakan Teknik OTA

Olin Ananda, Bomo Wibowo Sanjaya, dan Jannus Marpaung (2021) dari Universitas Tanjungpura melakukan penelitian tentang implementasi OTA untuk memperbarui parameter waktu *delay* pada sistem lampu lalu lintas berbasis *mikrokontroler*. OTA digunakan untuk menggantikan metode pembaruan manual, sehingga waktu pengaturan ulang dapat dilakukan secara jarak jauh melalui jaringan WiFi. Hasil pengujian menunjukkan bahwa sistem tetap dapat berjalan stabil meskipun pembaruan dilakukan secara OTA[46]. Namun, penelitian ini tidak mengeksplorasi lebih jauh dampak pembaruan tersebut terhadap performa perangkat, seperti konsumsi RAM, CPU. Hal ini berbeda dengan fokus penelitian yang tengah dilakukan, yang justru bertujuan untuk menganalisis aspek *resource* usage dari berbagai mekanisme OTA.

2.9 Gap Penelitian

Dari berbagai penelitian sebelumnya, ditemukan sejumlah kekurangan (gap) terkait implementasi OTA pada perangkat IoT sumber-daya terbatas seperti ESP32. Gap tersebut dapat dirangkum sebagai berikut:

- Penelitian masih minim melakukan pengukuran resource secara menyeluruh, misalnya beban CPU, RAM, flash, dan durasi update dalam satu eksperimen. Sebagian penelitian hanya menilai kecepatan update atau tingkat keberhasilan tanpa menganalisis overhead kriptografi.
- Sebagian besar studi hanya menguji satu algoritma keamanan secara tunggal, tanpa membandingkan beberapa mekanisme keamanan (Plain vs SHA-256 vs HMAC vs AES) dalam kondisi sistem dan perangkat yang sama[47][29].
- Banyak penelitian hanya melakukan simulasi algoritma, tanpa menguji alur OTA end-to-end yang lengkap (upload, manifest, download, validasi, flash write, reboot).
- Belum ada penelitian yang mengevaluasi trade-off antara tingkat keamanan dan efisiensi sumber daya, yang sangat penting pada perangkat kecil dan berbasis baterai.

Penelitian ini menawarkan kontribusi berupa:

- Studi komparatif OTA pada ESP32 secara penuh (end-to-end) menggunakan empat metode keamanan: Plain OTA, SHA-256, HMAC-SHA256, AES-128-CTR.
- Pengukuran empiris sumber daya meliputi: RAM, flash, beban CPU (Avg & Max).
- Menentukan mekanisme OTA paling efisien berdasarkan kompromi keamanan-vs-resource pada perangkat nyata ESP32.

Table 51 Gap Penelitian

Aspek yang Diteliti	Temuan Penelitian Terdahulu	Kekurangan (Gap)	Kontribusi Penelitian Ini
Penggunaan resource OTA (CPU, RAM, Flash)	Hanya sebagian aspek diukur (biasanya waktu update atau memori saja)	Tidak ada pengukuran menyeluruh pada satu eksperimen	Mengukur seluruh resource secara simultan pada ESP32
Variasi metode keamanan OTA	Umumnya hanya 1 algoritma (AES saja, SHA saja, token saja)	Tidak ada perbandingan multi-	Membandingkan Plain, SHA-256, HMAC, AES

Aspek yang Diteliti	Temuan Penelitian Terdahulu	Kekurangan (Gap)	Kontribusi Penelitian Ini
		algoritma dalam satu studi	
Implementasi OTA end-to-end	Banyak penelitian hanya simulasi hashing/enkripsi	Tidak mencakup alur lengkap (upload → manifest → download → verifikasi → flash → reboot)	Implementasi penuh OTA dari server → perangkat
Trade-off keamanan vs efisiensi	Keamanan dibahas tanpa mempertimbangkan penggunaan sumber daya	Tidak ada evaluasi keseimbangan keamanan-vs-efisiensi	Menilai tingkat efisiensi terbaik untuk IoT terbatas
Fokus peralatan nyata	Banyak berbasis simulasi atau perangkat selain ESP32	Minim eksperimen empiris langsung pada ESP32	Eksperimen langsung pada ESP32 modul DevKit

2.10 Jadwal Penelitian

Table 52 Jadwal Penelitian

KEGIATAN	JADWAL																
	Agustus				September				Oktober				November			Desember	
	Minggu Ke -																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pembagian Kelompok dan Dosen																	
Pembimbing																	
Penentuan Topik																	
Survei dan Review Jurnal																	
Survei dan Review Jurnal																	
Menulis / Menyusun Bab 2																	
Menulis / Menyusun Bab 3																	
Seminar Proposal																	
Revisi Proposal																	
Menulis / Menyusun Bab 4																	
Seminar TA 1																	
Revisi Seminar TA 1																	
Menulis / Menyusun Bab 5																	
Pra Sidang																	
Revisi Pra Sidang																	
Menulis / Menyusun Bab 6																	
Sidang Akhir																	

2.11 Estimasi Biaya Penelitian

Table 53 Estimasi Biaya Keseluruhan

No	Nama Barang	Jumlah	Harga @unit	Total
1	ESP32	1	80.000	80.000
				80.000

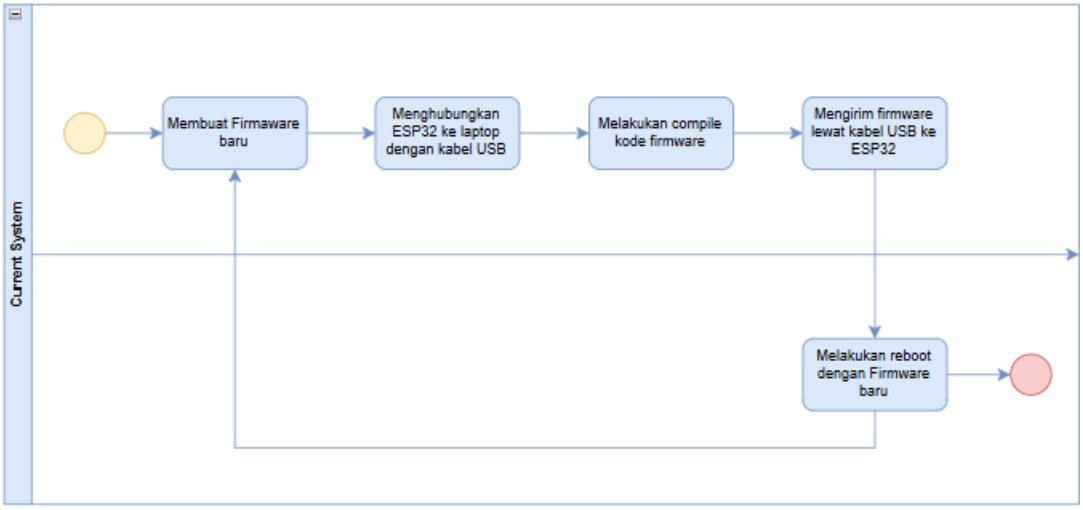
BAB III
ANALISIS DAN DESAIN

3.1 Analisis Sistem

Bab ini membahas analisis sistem yang menjadi dasar perancangan mekanisme pembaruan firmware OTA (Over-The-Air) pada ESP32. Analisis mencakup identifikasi masalah pada sistem yang berjalan saat ini, usulan solusi pada sistem target, serta kebutuhan sistem yang akan digunakan pada tahap implementasi dan pengujian. Fokus utama penelitian adalah bagaimana merancang mekanisme OTA yang efektif, andal, dan aman dengan mempertimbangkan keterbatasan sumber daya perangkat IoT.

3.1.1 Analisis Masalah

Perkembangan *Internet of Things* (IoT) menuntut perangkat untuk selalu diperbarui agar terhindar dari bug, celah keamanan, maupun peningkatan fitur. Namun, proses pembaruan *firmware* pada banyak perangkat IoT seperti ESP32 masih sering dilakukan secara manual dengan kabel USB, sehingga tidak efisien jika perangkat sudah tersebar luas di lapangan. Metode *Over-The-Air* (OTA) memang menawarkan solusi praktis, tetapi juga menghadirkan risiko keamanan karena *firmware* yang dikirim melalui jaringan rentan terhadap modifikasi, injeksi kode berbahaya, dan penyadapan. Metode sederhana seperti Plain OTA tidak memiliki perlindungan, sehingga integritas dan autentikasi *firmware* tidak terjamin.



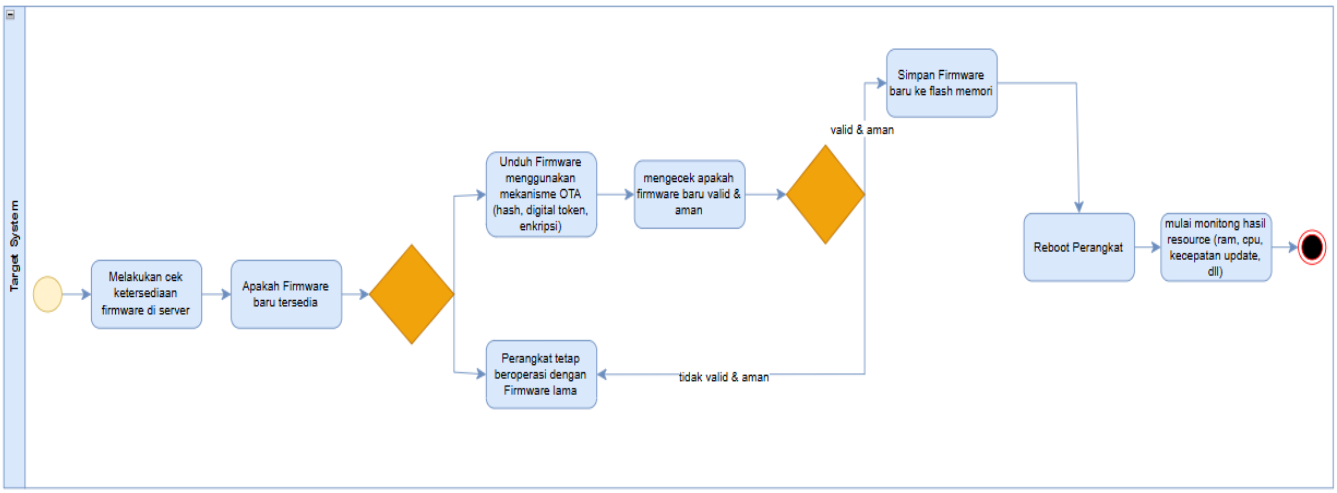
Gambar 3. 1 Current System

Selain itu, keterbatasan sumber daya perangkat IoT, seperti kapasitas memori, daya prosesor, energi yang membuat penerapan algoritma keamanan yang kompleks menjadi

tantangan tersendiri. Banyak penelitian sebelumnya hanya fokus pada satu mekanisme keamanan, sehingga belum ada gambaran komprehensif mengenai perbandingan efektivitas metode seperti pemeriksaan integritas (SHA-256), autentikasi (HMAC-SHA), maupun enkripsi (AES) jika diterapkan pada perangkat dengan keterbatasan sumber daya. Kondisi ini menjadi masalah utama yang perlu diteliti agar bisa menentukan metode OTA paling aman sekaligus efisien pada perangkat IoT.

3.1.2 Analisis Pemecahan Masalah

Salah satu faktor utama yang meningkatkan kerentanan perangkat IoT adalah keterlambatan atau bahkan ketiadaan pembaruan *firmware*. *Firmware* yang tidak diperbarui secara berkala akan tetap menyimpan bug lama, celah keamanan, serta ketidakcocokan dengan sistem dan protokol terbaru. Kondisi ini dapat berimplikasi pada berbagai masalah, mulai dari penurunan performa perangkat, kegagalan fungsi pada aplikasi IoT kritis, hingga terbukanya peluang serangan siber seperti injeksi kode berbahaya, exploit terhadap kerentanan lama, maupun serangan *Man-in-the-Middle* selama komunikasi. Dengan demikian, mekanisme pembaruan *firmware* yang andal, aman, dan efisien menjadi kebutuhan fundamental dalam pengelolaan siklus hidup perangkat IoT.



Gambar 3. 2 Target Sytem

Untuk menjawab permasalahan tersebut, penelitian ini mengusulkan pemanfaatan *Over-The-Air* (OTA) *firmware* update dengan pendekatan bertahap terhadap aspek keamanan. Mekanisme Plain OTA digunakan sebagai baseline sederhana untuk melihat kebutuhan minimum perangkat. Selanjutnya, pemeriksaan integritas menggunakan SHA-256 ditambahkan untuk memastikan *firmware* yang diterima identik dengan versi asli tanpa

adanya manipulasi. Untuk meningkatkan autentikasi sumber, digunakan digital token sehingga perangkat hanya akan menerima *firmware* resmi dari *server* tepercaya. Terakhir, enkripsi diterapkan agar proses transmisi *firmware* terlindungi dari penyadapan maupun reverse engineering oleh pihak tidak berwenang.

Kombinasi dari mekanisme tersebut diharapkan mampu menjawab dua aspek utama permasalahan: (1) menjamin *firmware* yang dipasang benar, utuh, dan resmi, serta (2) memastikan proses pembaruan berjalan dengan tingkat keamanan tinggi namun tetap efisien pada perangkat dengan sumber daya terbatas. Dengan demikian, bug lama dapat segera diperbaiki, patch keamanan dapat diterapkan lebih cepat, dan risiko serangan akibat *firmware* usang dapat diminimalisasi.

3.2 Analisis Kebutuhan Sistem

Pada tahap ini dilakukan identifikasi kebutuhan sistem yang digunakan untuk membangun mekanisme pembaruan firmware OTA pada ESP32. Kebutuhan sistem mencakup perangkat keras (hardware) dan perangkat lunak (software) yang mendukung proses pengembangan, penyimpanan firmware, komunikasi jaringan, hingga penerapan mekanisme keamanan seperti hashing, digital token, dan enkripsi.Perancangan kebutuhan sistem ini disusun berdasarkan karakteristik perangkat IoT yang memiliki keterbatasan memori, prosesor. Oleh karena itu, pemilihan perangkat keras dan perangkat lunak harus memperhatikan kemampuan untuk menjalankan proses OTA yang aman namun tetap efisien.

3.2.1 Analisis Kebutuhan Perangkat Keras (Hardware)

Tabel 3. 1 Analisis Kebutuhan Hardware

No	Perangkat Keras (Hardware)	Spesifikasi	Kegunaan
1	ESP32	WiFi Onboard, Flash 4–16 MB, RAM 520 KB, mendukung mbedTLS	Perangkat utama sebagai client OTA yang menerima, memverifikasi, mendekripsi, dan menjalankan firmware

No	Perangkat Keras (Hardware)	Spesifikasi	Kegunaan
2	Laptop	Windows/Linux, XAMPP/Apache, koneksi WiFi	Bertindak sebagai server OTA untuk menyimpan firmware, mengirim manifest JSON, melakukan hashing/HMAC/enkripsi
3.	Router / Hotspot	Jaringan 2.4 GHz (standar IoT)	Media komunikasi TCP/IP antar ESP32 dan server OTA
4.	Kabel USB		Upload firmware awal (boot OTA), debugging

Perangkat keras yang digunakan pada sistem ini terdiri dari beberapa komponen utama yang saling melengkapi. ESP32 berperan sebagai perangkat inti yang menjalankan firmware aplikasi sekaligus menjadi klien OTA. Modul ini dipilih karena memiliki WiFi onboard serta kapasitas flash dan RAM yang memadai untuk menjalankan mekanisme keamanan seperti SHA-256, HMAC, dan AES tanpa mengganggu fungsi utama sistem. Laptop digunakan sebagai server OTA, yang menjalankan XAMPP/PHP atau Python dan terhubung ke jaringan WiFi. Pada perangkat ini disimpan file firmware terbaru, manifest JSON, serta log proses pembaruan sehingga laptop menjadi pusat distribusi dan pengelolaan update.

Di antara kedua perangkat tersebut terdapat router atau hotspot yang menyediakan koneksi WiFi standar dan berfungsi sebagai media komunikasi TCP/IP antara ESP32 dan server OTA, sehingga proses request manifest dan download firmware dapat berlangsung melalui jaringan nirkabel. Selain itu, kabel USB digunakan pada tahap pemrograman awal dan debugging firmware ke ESP32, sekaligus dapat dimanfaatkan sebagai sumber catu daya yang stabil selama proses pengujian OTA berlangsung. Kombinasi keempat perangkat ini membentuk lingkungan uji yang representatif untuk mensimulasikan skenario pembaruan firmware pada perangkat IoT berbasis jaringan.

3.2.2 Analisis Kebutuhan Perangkat Lunak (Software)

Tabel 3. 2 Analisis Kebutuhan Software

No	Perangkat Lunak (Software)	Spesifikasi	Kegunaan
1	Arduino IDE	Versi terbaru, mendukung board ESP32 dan library WiFi/HTTP/Update/mbedTLS	Digunakan untuk menulis, mengompilasi, dan mengunggah program ke ESP32. Mendukung library penting seperti WiFi.h, HTTPClient.h, Update.h, serta mbedTLS untuk fitur keamanan (SHA-256, HMAC, AES).
2	XAMPP	PHP dengan ekstensi hash, openssl, dan json	Berfungsi sebagai server OTA untuk menyimpan firmware, membuat manifest JSON, menghitung hash SHA-256, HMAC token, dan melakukan enkripsi AES-CTR sebelum didistribusikan ke ESP32.
3	*Library* Kriptografi	Mendukung SHA-256, HMAC-SHA256, AES-128-CTR	Menyediakan fungsi keamanan di sisi ESP32 untuk memverifikasi integritas, autentikasi sumber firmware, dan mendekripsi firmware terenkripsi secara streaming.
4	Web Browser	Dukungan HTTP, form upload, tampilan respons	Digunakan oleh admin untuk mengunggah firmware, memicu pengecekan update, reboot perangkat jarak jauh, serta melihat status pembaruan.

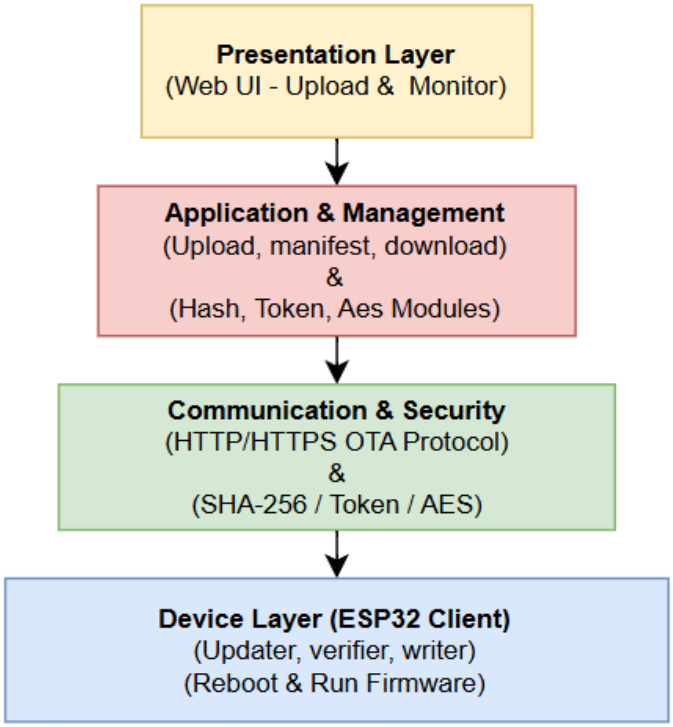
Pada penelitian ini digunakan tiga perangkat lunak utama yang saling melengkapi untuk mendukung implementasi dan pengujian mekanisme OTA pada ESP32. Arduino IDE

berperan sebagai lingkungan pengembangan yang digunakan untuk menulis, mengompilasi, dan mengunggah kode program ke perangkat ESP32. IDE ini mendukung berbagai *library* penting seperti WiFi.h untuk koneksi jaringan, HTTPClient.h untuk komunikasi HTTP, Update.h untuk proses OTA, serta mbedTLS yang memungkinkan implementasi mekanisme *hashing*, token digital, dan enkripsi. Sementara itu, XAMPP digunakan sebagai *server* lokal yang menjalankan layanan berbasis Apache, PHP, dan MySQL. Melalui *server* ini, *firmware* terbaru dapat diunggah dan disimpan, kemudian didistribusikan ke ESP32 menggunakan skrip PHP. Selain itu, XAMPP juga digunakan untuk mencatat log pembaruan *firmware*, yang menjadi sumber data penting dalam analisis performa sistem.

Selanjutnya, *library* kriptografi digunakan untuk mendukung variasi mekanisme OTA yang diuji, seperti SHA-256 untuk integritas data, digital token untuk autentikasi sederhana, serta AES untuk enkripsi *firmware*. Fokus utama penggunaan *library* ini bukan pada peningkatan keamanan, melainkan untuk melihat dampaknya terhadap penggunaan sumber daya ESP32 seperti RAM, CPU, serta waktu update.

3.3 Desain Sistem

3.3.1 Model Perancangan Sistem (Layer / Logical Architecture)



Gambar 3. 3 Model Perancangan Sistem

1. **Presentation Layer (User Interface)**

   Presentation Layer berfungsi sebagai antarmuka utama yang digunakan oleh pengguna atau administrator untuk mengelola proses pembaruan firmware. Pada sistem ini, antarmuka disediakan melalui website berbasis PHP dan HTML yang memungkinkan pengguna mengunggah firmware baru, memantau status update, serta melihat log aktivitas pembaruan. Melalui layer ini, interaksi pengguna dengan sistem berlangsung secara sederhana namun informatif, misalnya admin dapat mengecek apakah perangkat sudah menggunakan firmware terbaru atau apakah ada kegagalan update. Layer ini menjadi pintu masuk sistem sekaligus sarana monitoring hasil update firmware pada perangkat IoT.

2. **Application & Management Layer (Server-Side Module)**

   Layer ini merupakan inti dari proses manajemen pembaruan firmware yang berjalan di *server*. Modul-modul utama yang ada di layer ini meliputi upload module untuk menyimpan firmware ke *server*, *manifest* management untuk membuat dan

memperbarui file JSON yang berisi informasi versi firmware, serta download module untuk mendistribusikan firmware ke perangkat. Selain itu, di layer ini juga terdapat security management yang menangani fungsi-fungsi kriptografi, seperti generator *hash* SHA-256 untuk memastikan integritas file, digital token generator untuk autentikasi sumber *firmware*, serta AES *encryptor* untuk menjaga kerahasiaan firmware selama proses transmisi. Dengan demikian, Application & Management Layer memastikan firmware yang dikirim benar-benar valid dan terkelola dengan baik sebelum diterima perangkat IoT.

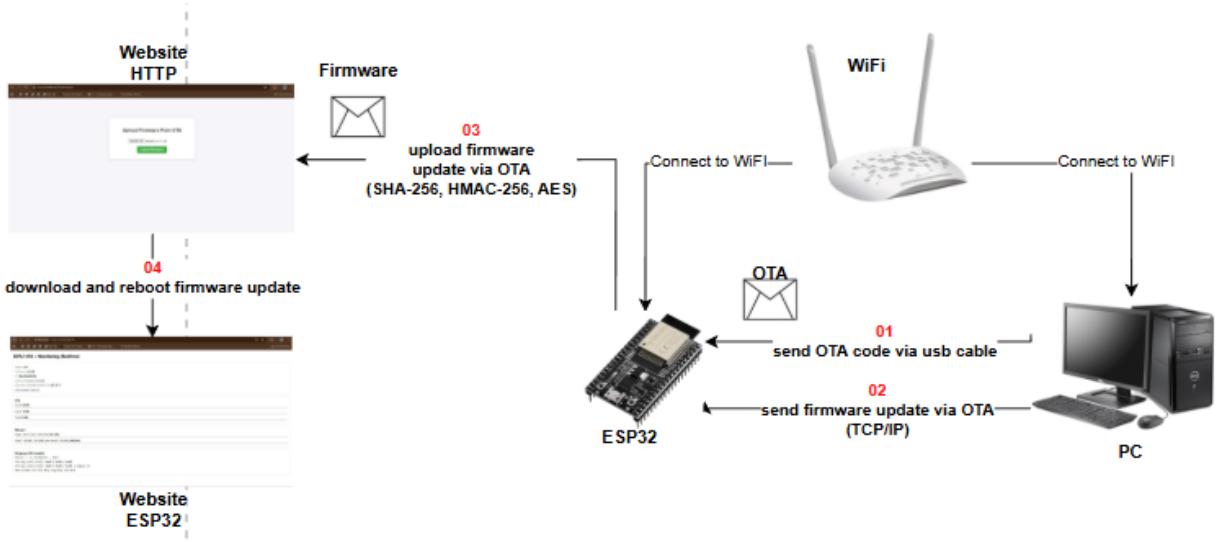
3. **Communication & Security Layer (OTA Protocol)**

   Layer ini mengatur komunikasi antara *server* dan perangkat IoT (ESP32) menggunakan protokol HTTP atau HTTPS. Pada tahap ini, firmware dapat dikirimkan dengan berbagai mekanisme keamanan tergantung metode yang dipilih dalam penelitian, yakni Plain OTA sebagai baseline sederhana, verifikasi integritas dengan SHA-256, autentikasi sumber firmware menggunakan digital token, atau enkripsi penuh dengan AES. Layer ini berfungsi sebagai jalur komunikasi sekaligus lapisan perlindungan agar data firmware yang ditransmisikan tetap aman, tidak dimodifikasi, serta tidak dapat diakses pihak ketiga yang tidak berwenang.

4. **Device Layer (ESP32 IoT Client)**

   Device Layer merupakan lapisan yang berada langsung pada perangkat IoT, dalam hal ini ESP32, yang berfungsi menerima, memverifikasi, dan menjalankan firmware baru. Pada layer ini, firmware updater bertugas mengecek versi terbaru di *server*. Jika tersedia, firmware diunduh lalu diverifikasi menggunakan *hash*, token, atau hasil dekripsi AES, tergantung mekanisme keamanan yang digunakan. Firmware yang lolos validasi akan ditulis ke flash memory oleh flash writer, kemudian perangkat akan melakukan reboot agar firmware baru dapat dijalankan. Dengan adanya proses verifikasi ini, perangkat hanya akan menjalankan firmware yang sah dan valid, sehingga resiko eksekusi kode berbahaya dapat diminimalisasi.

3.3.2 Desain Arsitektur Sistem

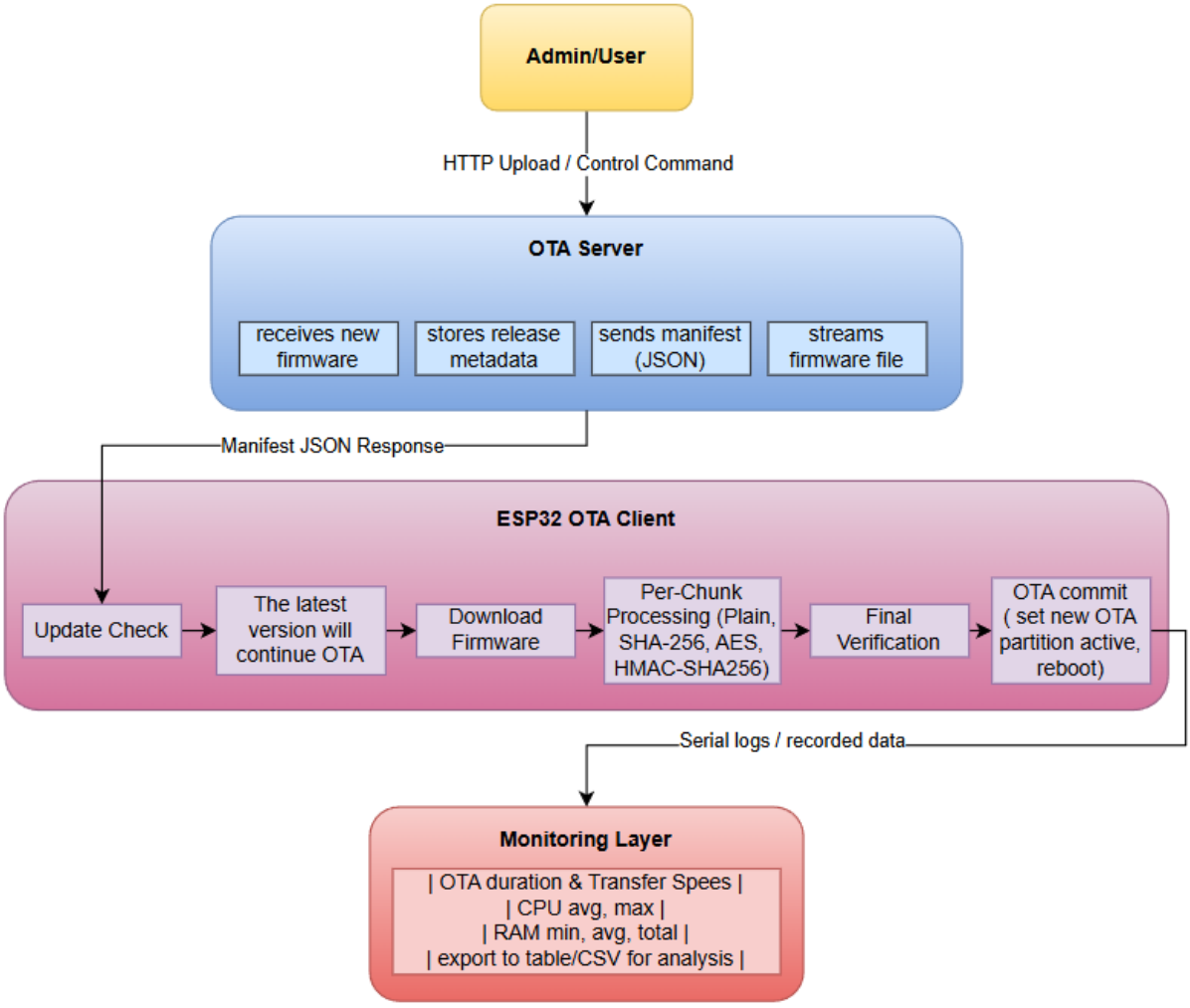


Gambar 3. 4 Desain Arsitektur Sistem OTA

Arsitektur sistem OTA pada ESP32 diawali dengan pengiriman firmware awal melalui kabel USB dari PC ke ESP32 agar fitur OTA aktif. Setelah perangkat terkoneksi ke jaringan Wi-Fi, proses pembaruan dilakukan secara nirkabel. PC/Server kemudian mengunggah file firmware terbaru melalui website OTA. File tersebut disimpan di server untuk kemudian diakses oleh ESP32.

ESP32 mengecek ketersediaan versi firmware melalui web server. Jika terdapat pembaruan, ESP32 mengunduh firmware baru melalui HTTP sesuai metode keamanan yang diterapkan (Plain, SHA-256, HMAC-SHA256, atau AES). Setelah proses download selesai, ESP32 melakukan verifikasi atau dekripsi berdasarkan algoritma yang digunakan. Jika valid, firmware ditulis ke partisi OTA baru dan perangkat melakukan reboot untuk menjalankan firmware terbaru. Status update dapat dilihat melalui website monitoring ESP32.

3.3.3 Diagram Blok Sistem



Gambar 3. 5 Diagram Blok Sistem

1. Admin/User

Admin atau user berperan sebagai pengendali proses pembaruan firmware melalui antarmuka web yang terhubung ke Server OTA. Tugas utamanya adalah menyiapkan dan mengunggah (upload) file firmware hasil kompilasi ke server menggunakan halaman upload yang telah disediakan. Melalui mekanisme ini, firmware dapat didistribusikan secara terpusat tanpa perlu melakukan konfigurasi pada setiap perangkat ESP32 secara manual. Tahapan ini sesuai dengan langkah (1) pada diagram blok, di mana admin berinteraksi langsung dengan Server OTA. Sebelum firmware diunggah, admin juga bertanggung jawab memastikan bahwa firmware telah diuji secara internal sehingga tidak menyebabkan kerusakan atau gangguan operasional ketika diinstal pada perangkat IoT. Pendekatan berbasis sentralisasi ini sangat relevan pada implementasi

IoT berskala besar, di mana jumlah perangkat dapat mencapai ratusan hingga ribuan unit.

2. *Server* OTA

Server OTA, sebagaimana digambarkan pada blok **(2)** dalam diagram, berfungsi sebagai pusat penyimpanan dan distribusi firmware. Server dibangun menggunakan skrip PHP pada platform web (misalnya XAMPP/Apache) dan memiliki beberapa fungsi penting, yaitu:

Table 54 Server OTA

Komponen Server	Fungsi
upload.php	Menerima file firmware yang diunggah admin
latest.json	Menyimpan metadata firmware seperti versi, hash, token, atau status enkripsi
check_update.php	Mengirim manifest JSON yang digunakan ESP32 untuk mengecek pembaruan
download.php	Menyediakan file firmware untuk diunduh ESP32 dalam bentuk streaming

Manifest JSON menjadi inti komunikasi antara ESP32 dan server, karena berisi informasi versi firmware, hash SHA-256 untuk integritas, token autentikasi (HMAC-SHA256), serta URL unduhan. Dengan demikian, server bertindak sebagai repository firmware yang aman dan terstruktur.

3. ESP32 *Transmitter*

ESP32 sebagai klien OTA menjalankan seluruh proses teknis pembaruan firmware. Langkah-langkah ini sesuai dengan blok (3)–(8) pada diagram sistem, yaitu:

Proses pada ESP32:

1. Cek Update via check_update.php untuk menerima manifest JSON.
2. Mengambil keputusan, apakah perlu melakukan update berdasarkan versi firmware.
3. Mengunduh firmware dari download.php menggunakan HTTP-streaming dengan ukuran chunk (±2 KB).
4. Validasi keamanan berdasarkan mode mekanisme:

Table 55 ESP32 Transmitter dengan Algoritma

Mode OTA	Fungsi Keamanan
Plain OTA	Langsung ditulis ke flash tanpa keamanan, cepat namun rentan manipulasi
SHA-256	Menghitung hash firmware lokal dan membandingkannya dengan hash manifest untuk memastikan integritas
HMAC Token	Memverifikasi token autentikasi untuk memastikan firmware berasal dari server resmi
AES-CTR	Melakukan dekripsi blok sebelum menulis ke flash, menjaga kerahasiaan firmware selama transfer

Jika seluruh validasi telah lulus, maka ESP32 akan:

- Menyimpan firmware ke flash
- Menutup proses penulisan (Update.end())
- Mengaktifkan partisi OTA baru
- Reboot dan menjalankan firmware terbaru

4. Monitoring Layer

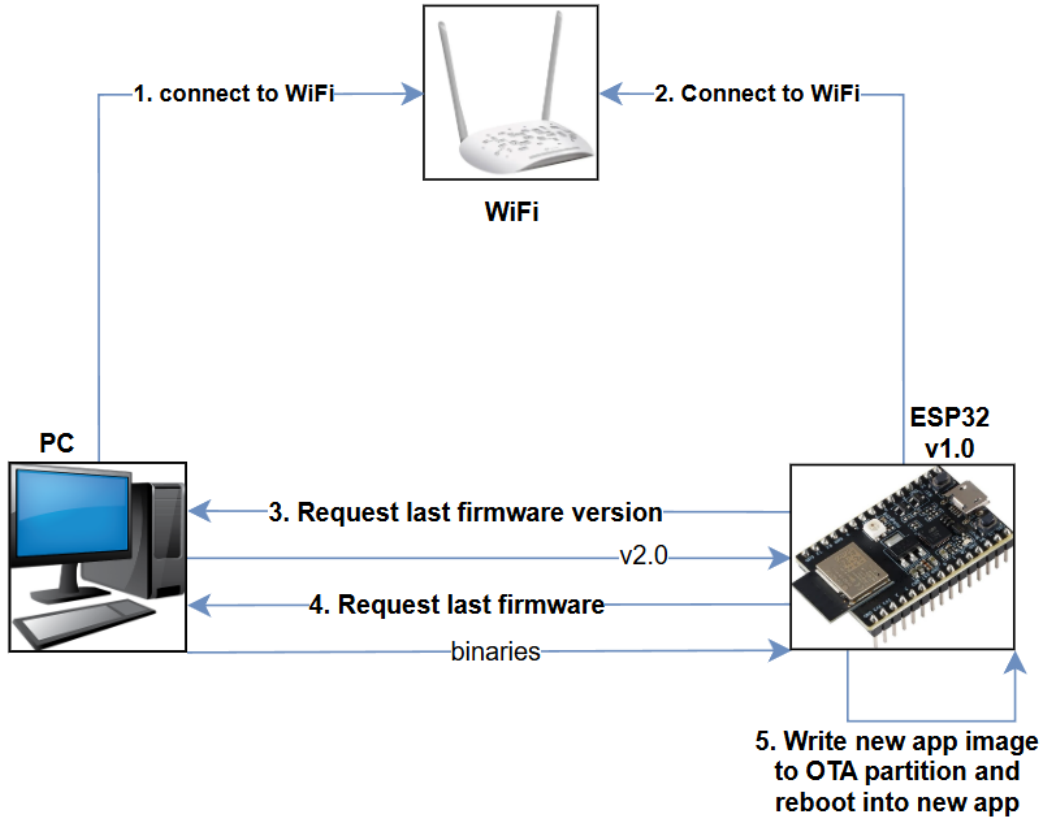
Monitoring layer merupakan komponen yang dirancang untuk mengevaluasi dampak pembaruan firmware terhadap performa perangkat ESP32. Monitoring dilakukan dengan tiga indikator utama, yaitu CPU usage, RAM usage, durasi.

Table 56 Monitoring Layer

Parameter	Deskripsi Pengukuran
CPU Usage	Mengukur beban prosesor selama proses download, validasi (SHA-256, token), dan instalasi firmware. Digunakan untuk melihat apakah mekanisme keamanan menambah load signifikan.
RAM Usage	Memantau penggunaan memori ESP32 selama OTA berlangsung. Penting untuk memastikan proses update tidak gagal akibat kekurangan memori, mengingat RAM perangkat terbatas.
Durasi	Memantau seberapa lama saat melakukan identifikasi, enkripsi, token dan mendownload file firmware

Dengan adanya monitoring layer, sistem tidak hanya menjalankan update, tetapi juga memberikan data evaluasi untuk menilai *trade-off* antara tingkat keamanan, performa.

3.3.4 Model Alur Komunikasi OTA



Gambar 3. 6 Model Alur Komunikasi OTA

Proses komunikasi data pada pembaruan firmware Over-the-Air (OTA) berlangsung antara perangkat ESP32, PC yang berperan sebagai server OTA, serta router WiFi sebagai media jaringan. ESP32 yang masih menggunakan firmware versi lama pertama-tama melakukan inisialisasi koneksi ke jaringan WiFi untuk mendapatkan akses ke server. Setelah perangkat berhasil terhubung ke jaringan, ESP32 mengirimkan permintaan HTTP GET kepada server melalui protokol TCP/IP untuk mengetahui apakah tersedia firmware terbaru. Permintaan ini diarahkan ke endpoint tertentu pada server, yang kemudian membalas dengan informasi versi (misalnya manifest JSON). Pada balasan tersebut, server menyertakan metadata seperti nomor versi firmware terbaru. Dengan menerima informasi ini, ESP32 kemudian membandingkan versinya saat ini (misalnya v1.0) dengan versi yang terdapat pada server (misalnya v2.0).

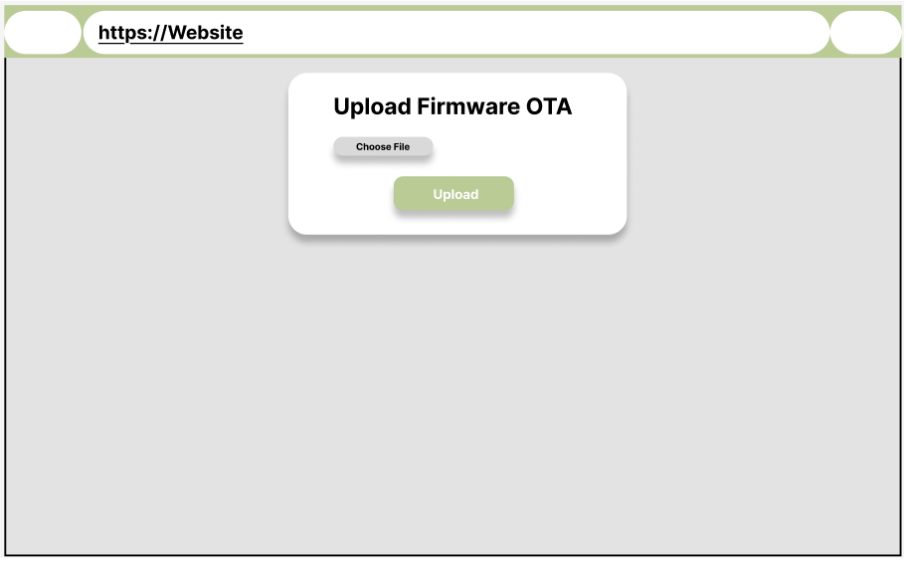
Apabila versi firmware yang tersedia pada server lebih baru, ESP32 melanjutkan proses dengan mengirimkan permintaan tambahan kepada server untuk mengunduh file firmware dalam bentuk biner. File firmware dikirim oleh server melalui metode streaming, sehingga

ESP32 tidak perlu menyimpan keseluruhan file ke dalam RAM. Data diterima dalam bentuk blok-blok kecil, kemudian langsung ditulis ke memori flash pada partisi khusus OTA sehingga tidak membebani keterbatasan memori perangkat. Selama proses penulisan berlangsung, ESP32 memastikan setiap bagian firmware diterima secara lengkap sesuai ukuran yang ditentukan, sebelum menandai firmware tersebut sebagai valid.

Setelah seluruh proses pengunduhan dan penulisan selesai, ESP32 melakukan finalisasi pembaruan dengan menutup akses flash dan mengatur partisi OTA baru sebagai firmware aktif pada proses boot berikutnya. Tahap terakhir adalah reboot perangkat, di mana ESP32 memulai ulang sistem dan menjalankan firmware terbaru yang sebelumnya disimpan di partisi OTA. Dengan alur komunikasi ini, proses pembaruan firmware dapat dilakukan secara otomatis tanpa perlu koneksi kabel, serta memanfaatkan jaringan WiFi sebagai perantara komunikasi antara server dan perangkat IoT. Proses ini memungkinkan penerapan pembaruan jarak jauh pada perangkat-perangkat IoT skala besar secara aman, terstruktur, dan efisien.

3.3.5 Desain Antarmuka Web (Upload/Cek Update/Reboot)

1. Web Upload

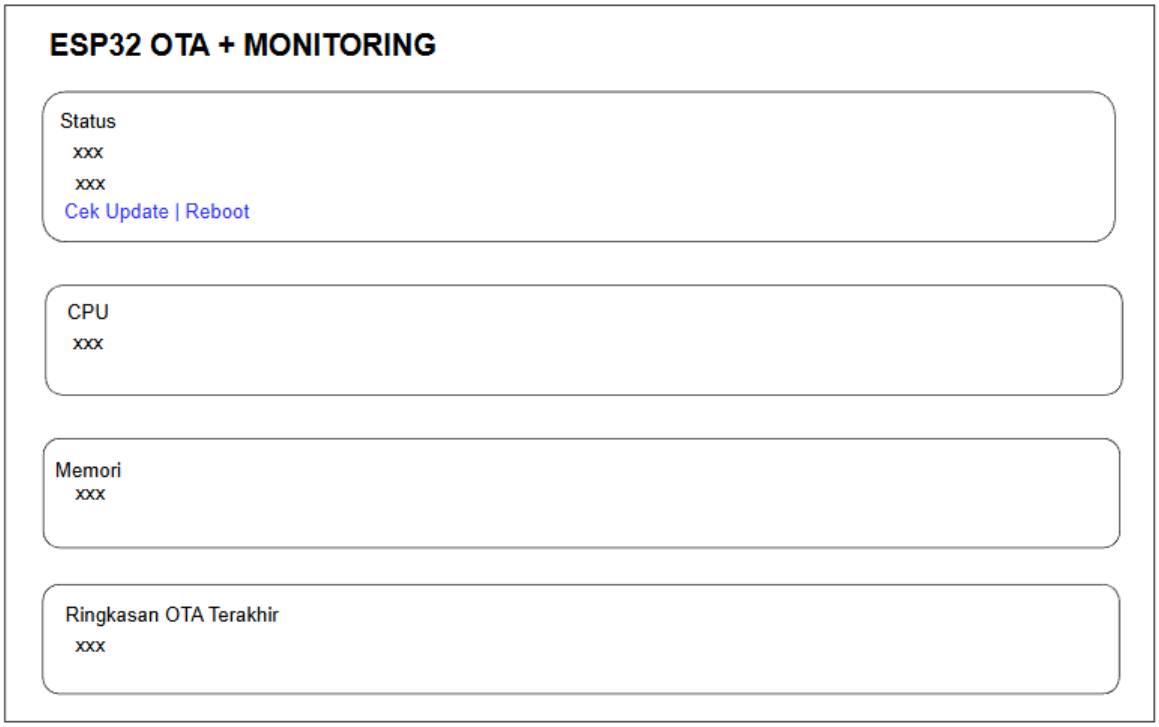


Gambar 3. 7 Desain Upload *Firmware*

Gambar tersebut menampilkan antarmuka web sederhana yang digunakan untuk melakukan proses *upload firmware* OTA (Over-the-Air), menunjukkan bahwa halaman ini dapat diakses melalui jaringan dengan protokol HTTPS. Antarmuka utama menampilkan sebuah *form* dengan judul *Upload Firmware* OTA yang dilengkapi tombol *Choose File* untuk

memilih file firmware dari komputer lokal, serta tombol *Upload* untuk mengunggah file tersebut ke server. Melalui mekanisme ini, admin atau pengelola sistem dapat dengan mudah memperbarui *firmware* perangkat IoT secara jarak jauh. *Firmware* yang berhasil diunggah kemudian akan disimpan di *server* dan dapat diakses oleh perangkat target yaitu ESP32, ketika melakukan proses pembaruan OTA.

2. Web Monitoring ESP32



Gambar 3. 8 Web Monitoring ESP32

Tampilan antarmuka web ESP32 OTA + Monitoring pada gambar di atas berfungsi untuk menampilkan informasi status perangkat secara real–time selama proses pembaruan firmware berlangsung maupun setelah firmware baru dijalankan. Pada bagian atas halaman, sistem menampilkan informasi dasar perangkat seperti status operasi (idle atau sedang update), versi firmware yang sedang digunakan, alamat IP perangkat, waktu aktif (uptime), serta hasil pembacaan sensor suhu DS18B20. Pengguna juga dapat melakukan dua aksi langsung dari halaman ini, yaitu cek pembaruan firmware (Check Update) dan melakukan reboot perangkat (Reboot) tanpa perlu koneksi fisik. Bagian berikutnya menampilkan penggunaan CPU ESP32 secara realtime, termasuk persentase load pada Core0 dan Core1, serta beban total yang menunjukkan seberapa besar proses firmware atau mekanisme keamanan mempengaruhi performa pemrosesan perangkat. Di bawahnya terdapat

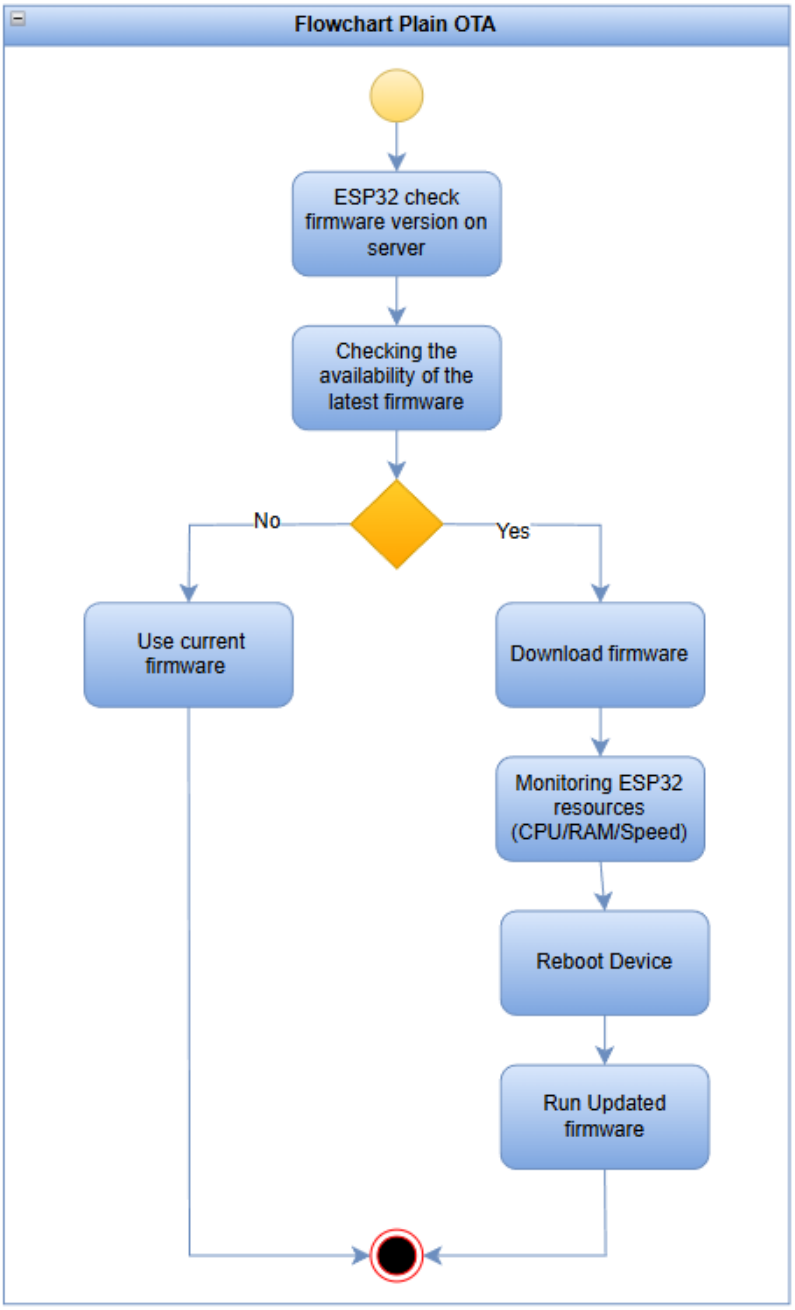
informasi memori, mencakup pemakaian heap (RAM) dan flash program, yang membantu mengetahui apakah firmware baru lebih berat dibandingkan versi sebelumnya.

Pada bagian terakhir, halaman menampilkan Ringkasan OTA terakhir, berisi metrik lengkap hasil proses pembaruan firmware sebelumnya, seperti durasi pengunduhan, kecepatan transfer, rata–rata penggunaan CPU, serta nilai minimum–maksimum penggunaan RAM selama proses OTA. Informasi ini memungkinkan pengguna membandingkan performa berbagai metode keamanan OTA (Plain, SHA-256, HMAC-SHA256, dan AES-CTR) secara objektif berdasarkan pengukuran langsung pada perangkat. Dengan demikian, website ini tidak hanya berfungsi sebagai interface update, tetapi juga sebagai alat evaluasi performa sistem OTA yang diterapkan.

3.4 Desain Proses (Flowchart OTA)

3.4.1 Flowchart Plain OTA

Proses pembaruan firmware melalui mekanisme Plain OTA pada ESP32 dimulai ketika perangkat dalam keadaan aktif dan melakukan pengecekan firmware terbaru pada server. Proses ini dilakukan dengan mengakses endpoint khusus yang menyediakan informasi versi firmware. Setelah memperoleh respons dari server, ESP32 membandingkan versi firmware yang sedang digunakan dengan versi terbaru yang tersedia. Jika versi pada perangkat masih sama atau tidak ada pembaruan pada server, maka perangkat tetap menjalankan firmware lama tanpa perubahan. Namun, apabila terdapat versi firmware yang lebih baru, ESP32 kemudian melanjutkan proses download.



Gambar 3. 9 Flowchart Plain OTA

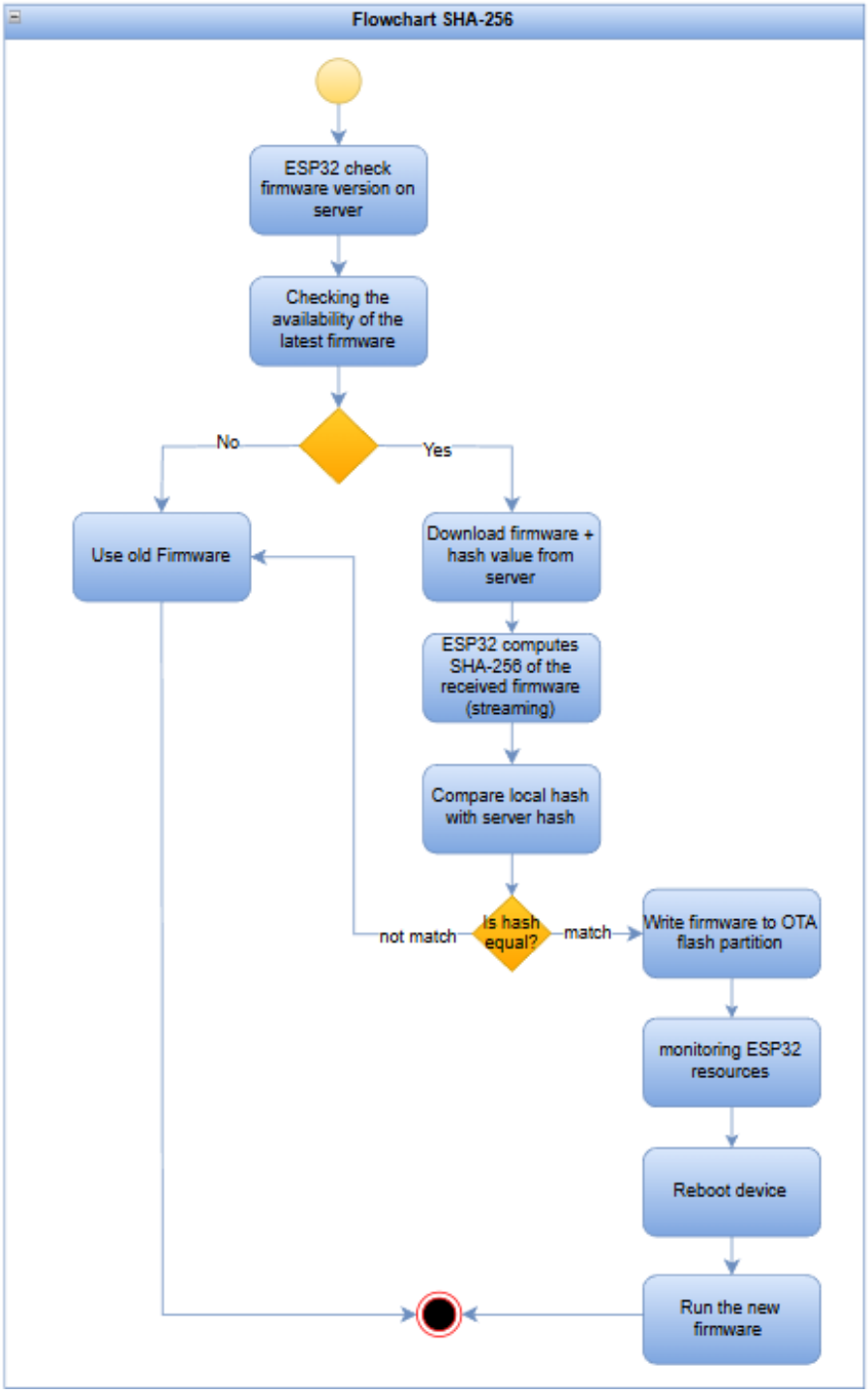
Setelah proses pengunduhan dimulai, ESP32 menerima data firmware dalam bentuk biner melalui HTTP. Pada tahap ini dilakukan monitoring resource perangkat, khususnya penggunaan CPU dan RAM selama transfer data berlangsung. Monitoring dilakukan pada tahap ini karena proses pengunduhan merupakan bagian yang paling intensif dalam penggunaan jaringan dan buffer memori, sehingga berkaitan langsung dengan performa perangkat. Data monitoring kemudian dicatat sebagai bagian dari analisis penelitian.

Setelah file firmware selesai diunduh, langkah selanjutnya adalah menuliskan firmware tersebut ke dalam partisi flash OTA. ESP32 menggunakan partisi OTA khusus yang memungkinkan firmware baru ditulis tanpa menghapus firmware lama terlebih dahulu. Setelah penulisan selesai dan diverifikasi, ESP32 melakukan proses reboot. Pada tahap reboot, bootloader ESP32 mengatur partisi yang berisi firmware baru sebagai partisi aktif, sehingga setelah perangkat menyala kembali, ESP32 menjalankan firmware terbaru secara otomatis. Dengan demikian, pembaruan firmware berhasil dilakukan sepenuhnya tanpa intervensi fisik, dan perangkat siap beroperasi kembali dengan versi firmware yang telah diperbarui.

3.4.2 Flowchart OTA dengan SHA-256

Pada mekanisme OTA dengan SHA-256 ini, alur dimulai dari sisi server ketika admin mengunggah file firmware baru (.bin). Server menghitung nilai hash SHA-256 dari file tersebut dan menyimpannya bersama metadata firmware di berkas manifest, sehingga server sudah memiliki hash referensi untuk setiap rilis firmware. Di sisi ESP32, ketika pengguna menekan tombol Check Update pada halaman web, perangkat memanggil check_update.php dan menerima respons JSON yang berisi versi terbaru, URL firmware, serta nilai SHA-256 dari server. Jika versi pada server lebih baru dibanding firmware yang sedang berjalan, ESP32 memulai proses download dengan mengakses download.php.

Selama proses download, server mengirimkan firmware dalam bentuk stream biner, sementara ESP32 melakukan tiga hal secara paralel untuk setiap chunk data yang diterima: (1) menulis chunk tersebut ke partisi OTA (misalnya OTA_1), (2) meng-update konteks SHA-256 menggunakan library mbedTLS untuk menghitung hash lokal, dan (3) mencatat durasi, penggunaan CPU, serta penggunaan RAM sebagai bagian dari monitoring resource. Setelah seluruh data diterima, ESP32 melakukan finalize SHA-256 sehingga diperoleh hash lokal, kemudian membandingkannya dengan hash referensi dari manifest server. Jika kedua hash tidak cocok, proses OTA dibatalkan, partisi OTA dianggap tidak valid, dan halaman web ESP32 menampilkan status “hash mismatch” sekaligus hasil pengukuran resource selama proses gagal tersebut.



Gambar 3. 10 Flowchart SHA-256

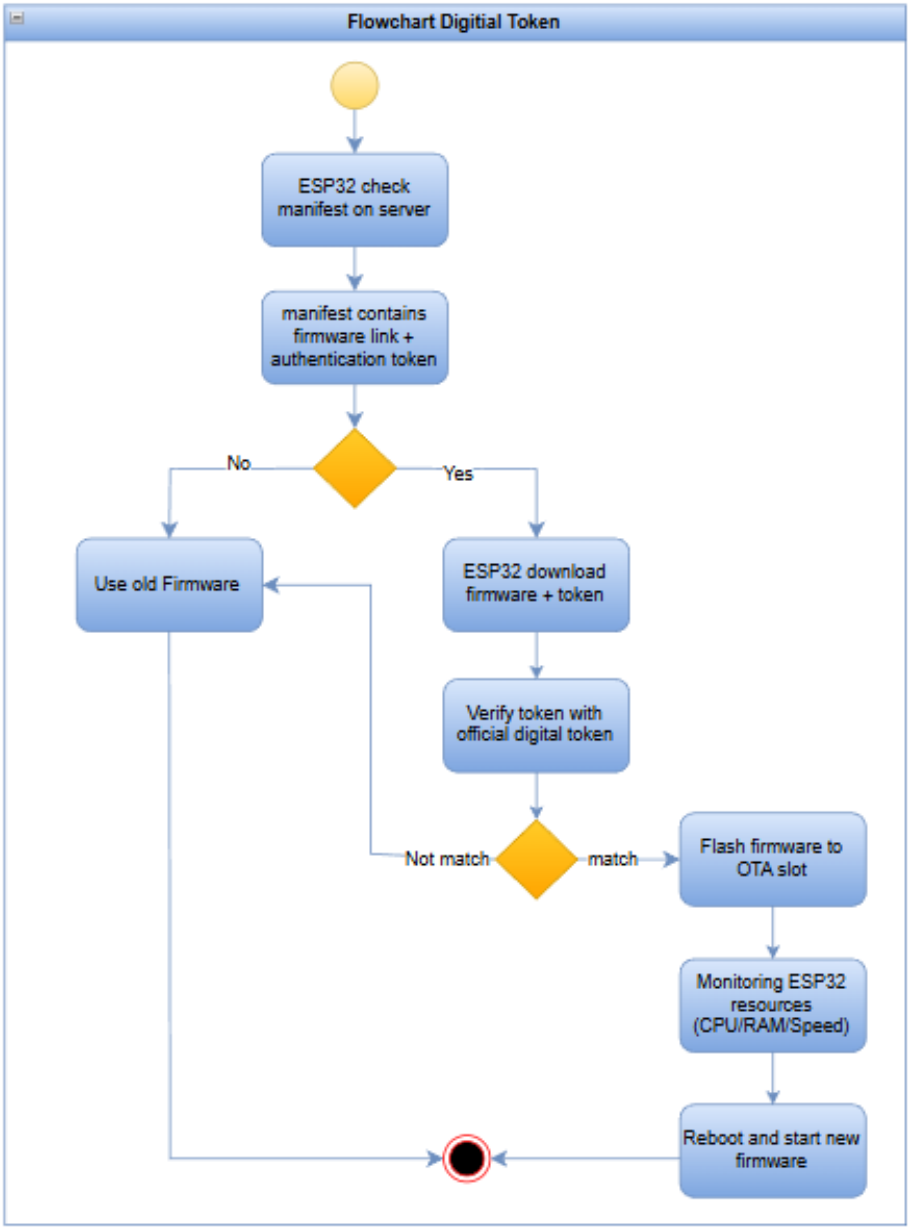
Sebaliknya, jika hash lokal dan hash server cocok, berarti firmware yang diterima utuh dan tidak dimodifikasi. ESP32 menandai image OTA sebagai valid, menghentikan proses monitoring, menghitung nilai rata-rata dan median dari data durasi/CPU/RAM, lalu menampilkan ringkasan hasil tersebut pada web (misalnya “Update success – please click reboot” beserta grafik resource). Pada tahap ini firmware baru belum dijalankan; perangkat

masih menggunakan firmware lama sampai pengguna menekan tombol Reboot. Setelah reboot, bootloader memilih partisi OTA yang baru diisi (misalnya beralih dari OTA_0 ke OTA_1), menjalankan firmware baru, dan sistem siap digunakan dengan versi terbaru. Dengan cara ini, penerapan SHA-256 tidak hanya menjamin integritas file firmware, tetapi juga memberikan visibilitas yang jelas di web ESP32 mengenai performa resource ESP32 selama proses download dan penulisan firmware berlangsung.

3.4.3 Flowchart OTA dengan Digital Token (HMAC-SHA256)

Pada mekanisme OTA dengan Digital Token, ESP32 melakukan proses pembaruan firmware dengan metode autentikasi berbasis HMAC-SHA256 untuk memastikan bahwa firmware yang diterima benar-benar berasal dari server resmi. Proses diawali ketika ESP32 melakukan pengecekan ke server untuk mengambil manifest yang berisi informasi firmware terbaru, seperti URL file pembaruan dan token autentikasi. Token ini digunakan sebagai nilai referensi untuk memverifikasi keaslian firmware sebelum dipasang ke memori flash. Jika hasil pengecekan menunjukkan bahwa tidak terdapat token valid pada manifest, maka pembaruan dibatalkan dan ESP32 tetap menggunakan firmware lama. Namun jika token tersedia, ESP32 mulai mengunduh firmware beserta token yang disertakan. Selama proses download berlangsung, ESP32 tidak menyimpan file sepenuhnya di RAM, tetapi menerimanya secara bertahap (streaming) per blok data dan secara paralel menghitung nilai HMAC-SHA256 berdasarkan isi data yang diterima.

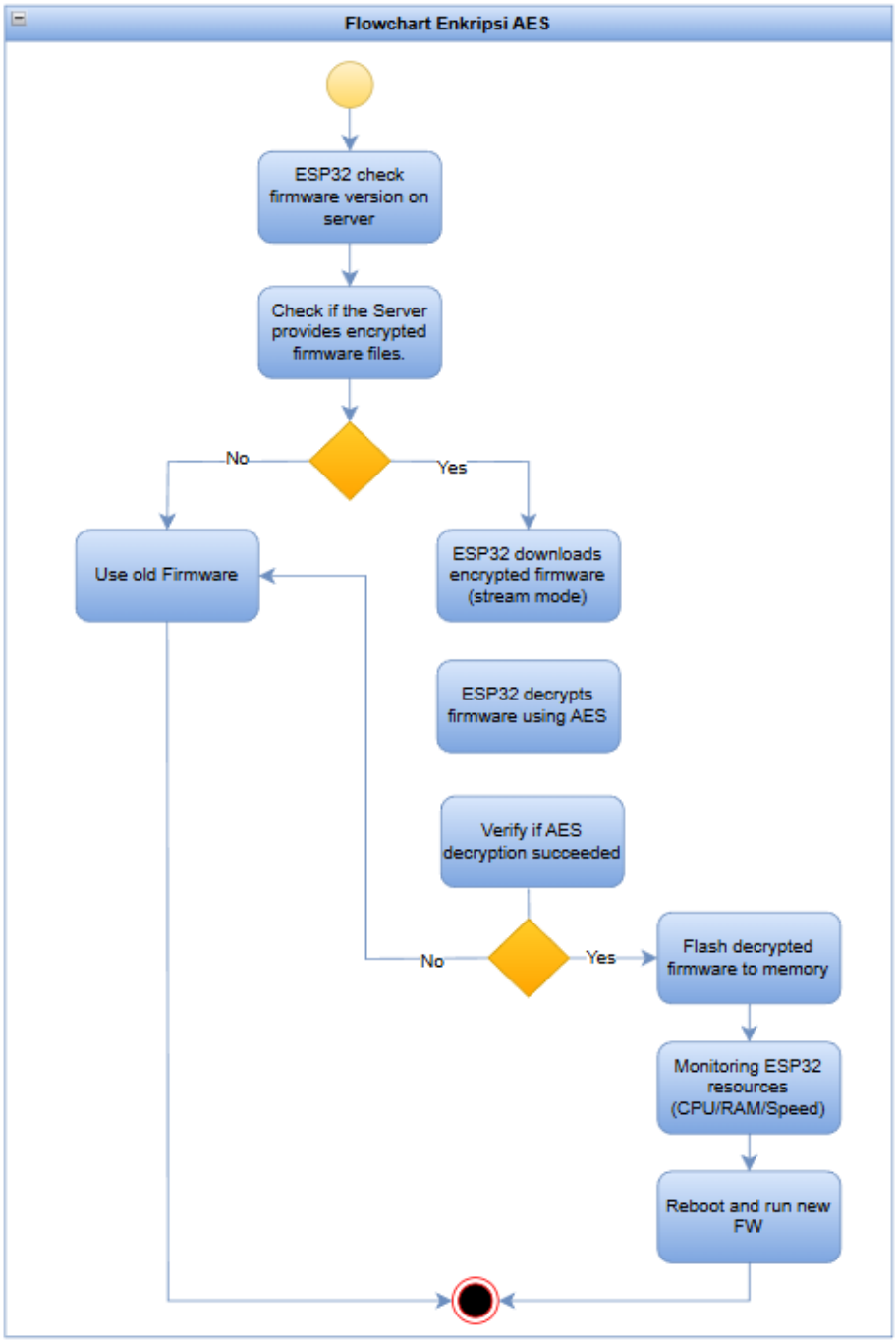
Setelah seluruh firmware selesai diunduh, ESP32 mencocokkan hasil perhitungan HMAC-SHA256 dengan token resmi yang dikirim oleh server. Jika nilai token tidak cocok, file dianggap tidak sah, sehingga ESP32 membatalkan proses pembaruan dan tetap menjalankan firmware lama untuk menghindari terjadinya modifikasi berbahaya. Sebaliknya, jika token valid dan cocok, firmware disimpan ke partisi OTA dan ESP32 menampilkan hasil monitoring resource, seperti penggunaan CPU, RAM, kecepatan unduh, dan durasi waktu pembaruan. Setelah proses ini selesai, perangkat melakukan reboot untuk mengaktifkan firmware versi terbaru.



Gambar 3. 11 Tampilan *Flowchart* Digital Token

3.4.4 Flowchart OTA dengan AES (Enkripsi)

Pada metode OTA dengan enkripsi AES, ESP32 pertama-tama melakukan pengecekan ke server untuk mengetahui apakah terdapat firmware baru yang tersedia dalam format terenkripsi. Jika server tidak menyediakan file firmware terenkripsi, perangkat membatalkan proses pembaruan dan tetap menggunakan firmware lama. Namun, jika ditemukan firmware terbaru dalam bentuk terenkripsi, ESP32 mengunduh data tersebut secara streaming sehingga tidak membebani kapasitas RAM.



Gambar 3. 12 Tampilan *Flowchart* AES

Setelah seluruh data firmware diterima, ESP32 mulai melakukan proses dekripsi menggunakan algoritma AES dalam mode CTR, yang mendukung pengolahan data secara bertahap sehingga sesuai dengan keterbatasan memori perangkat IoT. Setelah dekripsi selesai, ESP32 memverifikasi apakah hasil dekripsi valid dan utuh. Jika proses tersebut gagal, perangkat tidak menyimpan data firmware baru dan tetap melanjutkan memakai firmware yang lama.

Jika dekripsi dinyatakan berhasil, perangkat menuliskan firmware yang sudah didekripsi ke memori flash pada partisi OTA. Setelah proses keamanan AES selesai, ESP32 melakukan monitoring resource internal seperti penggunaan CPU, RAM, energi, serta waktu proses yang digunakan selama proses pembaruan. Hasil tersebut kemudian dikirimkan dan ditampilkan pada halaman web ESP32 untuk memberikan informasi performa dari algoritma AES. Setelah monitoring selesai, perangkat melakukan reboot dan menjalankan firmware baru.

3.5 Skenario Pengujian

3.5.1 Skenario Pengujian Fungsionalitas

Table 57 Skenario Pengujian Fungsionalitas

Komponen	Deskripsi Pengujian
Tujuan Pengujian	Memastikan seluruh proses OTA berjalan lengkap dan valid: cek versi → download → validasi keamanan → tulis firmware → reboot & jalankan versi baru.
Parameter yang Diuji	- Versi firmware baru terpasang - Proses download berjalan - Validasi hash/token/dekripsi berhasil - Firmware dapat dijalankan setelah reboot
Kondisi Awal	- ESP32 masih menggunakan firmware lama - Server OTA sudah memiliki firmware terbaru + manifest JSON
Langkah Pengujian	1. ESP32 mengirim request manifest ke server 2. Server mengirim metadata firmware (versi, hash/token/enkripsi) 3. ESP32 membandingkan versi lama & baru 4. ESP32 mengunduh file firmware 5. ESP32 melakukan verifikasi (SHA-256 / Token / AES) 6. Jika valid → firmware ditulis ke flash 7. ESP32 reboot otomatis & menjalankan firmware terbaru
Kriteria Kelulusan	Firmware berhasil diperbarui, device boot sesuai program baru, hasil resource ESP32 akurat, dan tidak ada error pada proses validasi ataupun penulisan flash

3.5.2 Skenario Pengujian Komunikasi Data

Table 58 Skenario Pengujian Komunikasi Data

Komponen	Deskripsi Pengujian
Tujuan Pengujian	Menilai stabilitas komunikasi jaringan antara ESP32 dan server OTA saat request manifest & download firmware
Parameter yang Diuji	- Respons server terhadap request HTTP GET - Stabilitas Wi-Fi selama download - Integritas data yang diterima (ukuran data sesuai) - Server tetap responsif ketika multi-client
Kondisi Awal	ESP32 terhubung ke jaringan Wi-Fi stabil, server menyediakan manifest JSON valid & file firmware siap di-download
Langkah Pengujian	1. ESP32 melakukan HTTP GET ke server 2. Server mengirim JSON (manifest firmware) 3. ESP32 parsing JSON dan cek versi 4. ESP32 download firmware secara streaming 5. ESP32 memverifikasi ukuran dan isi firmware 6. Ulangi pada beberapa ESP32 bersamaan (multi-client test)
Kriteria Kelulusan	Tidak ada paket error/korup, manifest diterima lengkap, server tetap responsif meski beberapa perangkat melakukan update bersamaan, firmware diterima utuh dan dapat diverifikasi

3.5.3 Skenario Pengujian Resource Usage (CPU, RAM, Energi, Waktu OTA)

Table 59 Skenario Pengujian Resource Usage

Parameter Uji	Deskripsi Pengukuran	Metode Akuisisi Data	Kriteria Evaluasi
CPU Usage (Avg & Max)	Mengukur beban prosesor selama download, verifikasi hash/token, & dekripsi	FreeRTOS runtime stats	Semakin rendah beban CPU semakin efisien
RAM Usage (Min, Avg, Max)	Mengukur memori minimum tersisa selama proses OTA,	ESP.getFreeHeap() secara periodik	Tidak terjadi low memory (<20 KB)

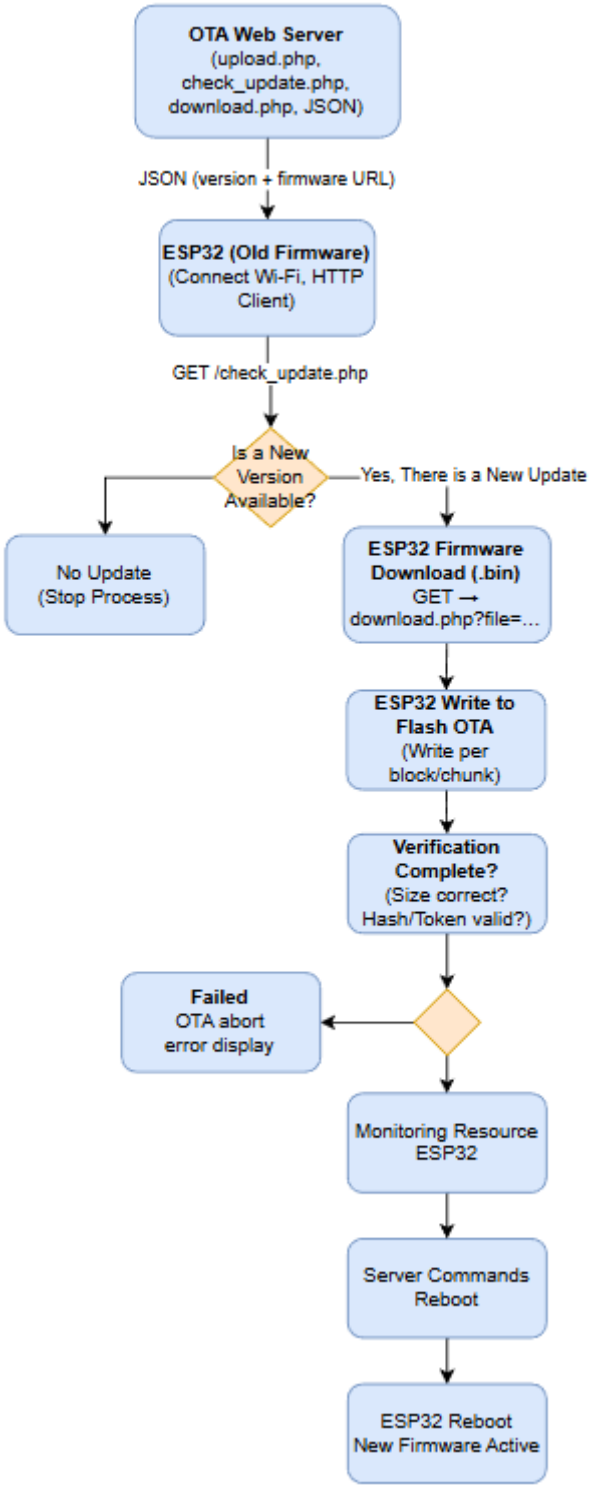
Parameter Uji	Deskripsi Pengukuran	Metode Akuisisi Data	Kriteria Evaluasi
	untuk menghindari crash		
Durasi OTA (Total Time)	Mengukur waktu dari cek update sampai reboot & menjalankan firmware baru	esp_timer_get_time()	Semakin cepat semakin efisien
Kecepatan Transfer (kb/s)	Mengukur throughput pengunduhan firmware	Ukuran file / waktu download	Throughput tinggi = lebih efisien
Keberhasilan Update / Rollback	Berapa kali update berhasil tanpa gagal	Counter keberhasilan pengujian	Tidak terjadi boot error/gagal OTA

BAB IV

HASIL DAN PEMABAHASAN

4.1 Implementasi Sistem OTA

4.1.1 Alur Proses Update OTA



Gambar 4. 1 Alur Proses Update OTA

Diagram diatas menunjukkan proses pembaruan firmware ESP32 melalui jaringan menggunakan mekanisme Over-The-Air (OTA). Proses dimulai dari server web yang menjadi sumber firmware dan metadata pembaruan. ESP32 terlebih dahulu mengirim permintaan ke check_update.php untuk memeriksa apakah terdapat versi firmware yang lebih baru. Jika versi sama, proses dihentikan. Jika terdapat versi baru, ESP32 akan mengunduh file firmware dari download.php dalam bentuk stream binary. Data yang diterima tidak disimpan penuh di RAM, tetapi ditulis secara bertahap ke partisi flash OTA menggunakan fungsi Update.write().

Setelah seluruh data selesai diterima, ESP32 memverifikasi hasil pembaruan sesuai ukuran file dan mekanisme keamanan (misalnya hash, token atau enkripsi). Jika verifikasi gagal, proses OTA dibatalkan (abort) dan pesan kesalahan ditampilkan. Jika berhasil, server akan menginstruksikan perangkat untuk melakukan reboot. Saat reboot, bootloader ESP32 mengaktifkan firmware baru sehingga perangkat sudah menjalankan versi terbaru tanpa memerlukan kabel atau intervensi fisik.

4.1.2 Implementasi Algoritma Keamanan OTA

4.1.2.1 Implementasi SHA-256

4.1.2.1.1 Implementasi Sisi Server (upload, hash, dan manifest)

Pada sisi server, proses utama SHA-256 dimulai ketika admin mengunggah file firmware .bin. File ini disimpan di direktori khusus, kemudian server menghitung nilai hash SHA-256 dan menyimpan metadata rilis ke dalam berkas latest.json. Metadata inilah yang nantinya digunakan ESP32 untuk memverifikasi integritas firmware.

```
$destPath  = $target_dir .
basename($_FILES['firmware']['name']);
move_uploaded_file($_FILES['firmware']['tmp_name'],
$destPath);
```

- Potongan di atas menunjukkan tahap awal proses upload. Variabel $destPath menyimpan path lengkap file tujuan di folder firmware/, sedangkan fungsi move_uploaded_file() memindahkan file sementara yang dikirim melalui form ke lokasi permanen di server. Setelah baris ini, file firmware .bin resmi tersimpan dan siap diproses lebih lanjut.

```
$sha256 = hash_file('sha256', $destPath);
```

- Baris ini menghitung nilai hash SHA-256 dari file firmware yang baru saja diunggah. Fungsi hash_file('sha256', ...) membaca isi file dan menghasilkan digest 64 karakter heksadesimal. Nilai inilah yang nantinya menjadi referensi integritas di sisi ESP32: jika hash lokal tidak sama dengan nilai ini, firmware dianggap rusak atau telah dimodifikasi.

```
$meta = [
    "version" => $version,
    "file"    => basename($destPath),
    "size"    => filesize($destPath),
    "sha256"  => $sha256,
    "url"       => $base_url . "/download.php?file=" .
rawurlencode(basename($destPath))
];

file_put_contents(
    $target_dir . "latest.json",
    json_encode($meta, JSON_PRETTY_PRINT)
);
```

- Bagian ini menyusun array $meta yang berisi informasi penting tentang rilis firmware terbaru: nomor versi, nama file di server, ukuran file dalam byte, nilai hash SHA-256, dan URL yang akan digunakan perangkat untuk mengunduh firmware. Seluruh metadata disimpan ke berkas latest.json dalam format JSON yang rapi. Berkas manifest ini menjadi sumber kebenaran (single source of truth) bagi semua perangkat ESP32 saat melakukan pengecekan update dan verifikasi integritas.

4.1.2.1.2 Implementasi Sisi Server (check_update.php dan download.php)

Setelah metadata tersedia, server menyediakan dua endpoint utama: satu untuk mengirim manifest dan satu lagi untuk men-*stream* file firmware ke ESP32.

```
$files = glob($firmware_dir . "*.bin");
usort($files, function($a, $b){ return filemtime($b) -
filemtime($a); });
$latest  = $files[0];
$version = preg_match('/v[\d.]+/', $latest, $m) ? $m[0]
: "unknown";
$sha256  = hash_file('sha256', $latest);

echo json_encode([
    "version" => $version,
```

```
    "url"       => $HOST_BASE . "/download.php?file=" .
basename($latest),
    "size"    => filesize($latest),
    "sha256"  => $sha256
]);
```

- Potongan ini merepresentasikan logika di check_update.php. Fungsi glob() mencari semua file .bin di direktori firmware, lalu usort() mengurutkannya berdasarkan waktu modifikasi terakhir sehingga elemen pertama $files[0] adalah firmware terbaru. Versi firmware diekstrak dari nama file (misalnya firmware_v1.2.bin → v1.2), kemudian server kembali menghitung hash SHA-256 sebagai nilai referensi. Semua informasi tersebut dikirimkan ke ESP32 dalam bentuk JSON yang memuat version, url, size, dan sha256.

```
$filepath = $firmware_dir . basename($_GET['file']);
$sha256   = hash_file('sha256', $filepath);

header('Content-Type: application/octet-stream');
header('Content-Length: ' . filesize($filepath));
header('X-Firmware-SHA256: ' . $sha256); // opsional

readfile($filepath);
```

- Kode ini berada pada download.php dan bertugas mengirim file firmware ke ESP32 sebagai stream biner. Variabel $filepath menentukan file yang akan dikirim berdasarkan parameter file di URL. Server menambahkan header Content-Type dan Content-Length untuk memberi tahu ukuran file, dan secara opsional menyertakan header X-Firmware-SHA256 sebagai backup nilai hash. Fungsi readfile() kemudian mengirim isi file apa adanya ke klien, sementara proses verifikasi integritas sepenuhnya dilakukan oleh ESP32.

4.1.2.1.3 Implementasi Sisi ESP32 (Konfigurasi SHA-256)

Di sisi ESP32, perhitungan hash SHA-256 dilakukan menggunakan library mbedTLS bawaan ESP-IDF. Beberapa variabel global digunakan untuk menyimpan hash referensi dari server dan hash hasil perhitungan lokal.

```
#include "mbedtls/sha256.h"

String expectedSHA = "";  // hash dari server (manifest)
String lastCalcSHA = "";  // hash hasil perhitungan lokal
```

- Baris #include "mbedtls/sha256.h" mengaktifkan modul SHA-256 di mbedTLS sehingga ESP32 dapat memanggil fungsi-fungsi hash. Variabel expectedSHA berisi nilai SHA-256 dalam bentuk string hex 64 karakter yang dikirim dari server melalui manifest, sedangkan lastCalcSHA akan diisi setelah ESP32 selesai menghitung hash firmware yang diunduh.

```
String toHex(const uint8_t* dig, size_t len) {
  const char* hex = "0123456789abcdef";
  String out; out.reserve(len * 2);
  for (int i = 0; i < len; i++) {
    out += hex[dig[i] >> 4];
    out += hex[dig[i] & 0x0F];
  }
  return out;
}
```

- Fungsi utilitas toHex() mengubah array byte hasil hash (32 byte) menjadi string heksadesimal 64 karakter, agar formatnya sama dengan hash yang dikirim server. Konversi ini penting karena perbandingan hash dilakukan dalam bentuk teks, bukan biner.

4.1.2.1.4 Implementasi Sisi ESP32 (Streaming Hash saat OTA)

Proses inti integritas SHA-256 terjadi di fungsi OTA, di mana ESP32 menghitung hash sambil mengunduh firmware dan menulisnya ke flash.

```
mbedtls_sha256_context shaCtx;
mbedtls_sha256_init(&shaCtx);
mbedtls_sha256_starts(&shaCtx, 0);     // 0 = SHA-256
(bukan SHA-224)
```

- Pertama, ESP32 menyiapkan konteks hash SHA-256. Fungsi mbedtls_sha256_init() menginisialisasi struktur internal, sementara mbedtls_sha256_starts() mereset state sehingga konteks siap menerima data firmware yang akan di-hash.

```
uint8_t buf[2048];

int n = stream->readBytes((char*)buf, sizeof(buf));
if (n > 0) {
  Update.write(buf, n);                    // tulis ke
partisi OTA
  mbedtls_sha256_update(&shaCtx, buf, n);   // update
hash dengan chunk yang sama
}
```

- Setiap kali ESP32 menerima chunk data firmware (maksimum 2048 byte), data tersebut langsung ditangani dalam dua langkah: ditulis ke partisi OTA melalui Update.write() dan secara paralel diumpankan ke algoritma hash menggunakan mbedtls_sha256_update(). Pola ini memungkinkan perhitungan hash secara streaming tanpa menyimpan keseluruhan file firmware di RAM, yang sangat penting untuk perangkat dengan memori terbatas seperti ESP32.

```
uint8_t digest[32];
mbedtls_sha256_finish(&shaCtx, digest);
mbedtls_sha256_free(&shaCtx);

lastCalcSHA = toHex(digest, sizeof(digest));  // 32 byte
-> 64 hex
```

- Setelah seluruh data firmware diterima, mbedtls_sha256_finish() menghasilkan digest 32 byte yang disimpan di array digest. Fungsi toHex() kemudian mengubahnya menjadi string hex 64 karakter yang disimpan di lastCalcSHA. Konteks SHA-256 kemudian dibersihkan dengan mbedtls_sha256_free().

```
if (expectedSHA.length() > 0 && lastCalcSHA !=
expectedSHA) {
  Update.abort();       // batalkan OTA, jangan pakai
firmware ini
  return false;
}
```

- Pada tahap verifikasi, ESP32 membandingkan lastCalcSHA dengan expectedSHA dari manifest. Jika kedua nilai berbeda, berarti firmware yang diterima tidak identik dengan versi di server (korup atau dimodifikasi), sehingga proses OTA dibatalkan dengan Update.abort(). Dengan demikian, ESP32 tidak akan pernah melakukan boot dengan firmware yang integritasnya tidak terjamin.

```
if (expectedSize > 0 && totalWritten != expectedSize) {
  Update.abort();
  return false;
}
```

- Selain verifikasi hash, sistem juga memeriksa kesesuaian ukuran file yang diterima dengan expectedSize dari manifest. Jika jumlah byte yang tertulis ke flash tidak sama dengan nilai yang diharapkan, update dibatalkan untuk mencegah firmware tidak lengkap dijalankan.

4.1.2.1.5 Implementasi Sisi ESP32 (Pembacaan Manifest SHA-256)

Sebelum memulai proses OTA, ESP32 terlebih dahulu mengambil manifest dari server untuk mendapatkan hash dan informasi versi terbaru.

```
HTTPClient http;
if (!http.begin(CHECK_URL)) return;

if (http.GET() != HTTP_CODE_OK) {
    http.end();
    return;
}

String body = http.getString();
http.end();
```

- Kode ini membuka koneksi HTTP ke endpoint CHECK_URL (misalnya check_update.php) dan melakukan permintaan GET. Jika status HTTP tidak OK, proses dihentikan karena tanpa manifest yang valid perangkat tidak bisa meneruskan update.

```
StaticJsonDocument<512> doc;
if (deserializeJson(doc, body)) return;

expectedSHA  = doc["sha256"] | "";
expectedSize = doc["size"]   | 0;
String fwurl = doc["url"]    | "";
String latest = doc["version"] | "";
```

- Manifest yang diterima dalam bentuk JSON diparsing ke objek doc. ESP32 kemudian mengambil tiga nilai penting: sha256 sebagai hash referensi, size sebagai ukuran firmware, dan url sebagai lokasi file firmware. Nilai version juga dibaca untuk menentukan apakah firmware di perangkat sudah terbaru atau belum.

```
expectedSHA.trim();
expectedSHA.toLowerCase();

if (expectedSHA.length() != 64 || fwurl == "") return;
if (latest == CURRENT_VERSION) return;
```

- Pada tahap validasi, string hash dibersihkan dari spasi dan dipaksa menjadi huruf kecil, lalu dicek panjangnya apakah tepat 64 karakter. Jika hash atau URL tidak valid, atau jika versi firmware di server sama dengan CURRENT_VERSION, maka proses update tidak dilanjutkan untuk menghindari download yang tidak perlu.

```
doOTAFromURL(fwurl);
```

- Jika semua parameter manifest valid dan versi yang tersedia lebih baru, ESP32 memanggil doOTAFromURL() untuk memulai proses OTA. Selama proses download, mekanisme streaming SHA-256 yang telah dijelaskan sebelumnya akan berjalan untuk menjamin integritas firmware.

4.1.2.2 Implementasi HMAC-SHA256 (Token Digital)

4.1.2.2.1 Implementasi Sisi ESP32 (Konfigurasi dan Fungsi HMAC)

```
#include "mbedtls/md.h"

// Kunci rahasia HMAC (harus sama dengan server)
static const char* K_MAC = "kelompokTA09";
```

- Pada mekanisme digital token, ESP32 menggunakan modul mbedtls/md.h untuk menghitung HMAC-SHA256. Konstanta K_MAC adalah kunci rahasia yang dibagikan antara server dan perangkat. Kunci ini tidak pernah dikirim melalui jaringan sehingga hanya pihak yang memiliki K_MAC yang dapat menghasilkan token yang valid.

```
uint8_t out[32];
mbedtls_md_context_t ctx;
const mbedtls_md_info_t* info =
    mbedtls_md_info_from_type(MBEDTLS_MD_SHA256);
Bagian ini menyiapkan buffer out untuk menampung hasil
HMAC 32 byte, membuat konteks ctx, dan memilih algoritma
hash yang digunakan, yaitu SHA-256, melalui
mbedtls_md_info_from_type().
mbedtls_md_init(&ctx);
mbedtls_md_setup(&ctx, info, 1 /* HMAC */);
mbedtls_md_hmac_starts(&ctx, (const unsigned char*)key,
strlen(key));
mbedtls_md_hmac_update(&ctx, (const unsigned
char*)msg.c_str(), msg.length());
mbedtls_md_hmac_finish(&ctx, out);
mbedtls_md_free(&ctx);
```

- Pada blok ini, konteks HMAC diinisialisasi dan disiapkan untuk algoritma SHA-256. Fungsi mbedtls_md_hmac_starts() memulai proses HMAC dengan kunci rahasia, mbedtls_md_hmac_update() memasukkan isi pesan msg, dan mbedtls_md_hmac_finish() menghasilkan nilai HMAC 32 byte yang disimpan ke dalam out. Setelah selesai, konteks dibersihkan menggunakan mbedtls_md_free().

```
char hex[65];
for (int i=0; i<32; i++) sprintf(hex+2*i, "%02x",
out[i]);
hex[64]='\0';
return String(hex);
```

- Di bagian akhir, nilai HMAC biner dikonversi menjadi string heksadesimal 64 karakter menggunakan loop sprintf. String ini kemudian dikembalikan sebagai String dan digunakan untuk dibandingkan dengan token yang dikirim server pada manifest. Jika keduanya cocok, manifest dianggap valid.

4.1.2.2.2 Implementasi Sisi ESP32 (Verifikasi Digital Token)

Pada proses update berbasis token, ESP32 hanya akan meneruskan OTA jika manifest memiliki HMAC yang valid. Hal ini diatur dalam fungsi checkAndUpdate().

```
String ver   = doc["version"] | "";
String fwurl = doc["url"]     | "";
uint32_t size= doc["size"]    | 0;
String nonce = doc["nonce"]   | "";
uint32_t ts  = doc["ts"]      | 0;
String token = doc["token"]   | "";
```

- Setelah JSON manifest berhasil diparsing, ESP32 mengekstrak beberapa field penting: version, url, size, nonce, ts (timestamp), dan token (HMAC dari server). Semua field ini diperlukan untuk membangun kembali pesan yang sama persis dengan yang ditandatangani di server.

```
String msg = "ver=" + ver
           + "|size=" + String(size)
           + "|url=" + fwurl
           + "|nonce=" + nonce
           + "|ts=" + String(ts);
```

- String msg disusun dengan format tetap ver=...|size=...|url=...|nonce=...|ts=.... Format ini harus identik dengan sisi server; jika susunan atau separator diubah, nilai HMAC yang dihasilkan tidak akan cocok lagi.

```
String tloc = hmac_sha256_hex(msg, K_MAC);
if (tloc != token) {
  lastStatus = "HMAC mismatch";
  return;
}
```

- Fungsi hmac_sha256_hex() digunakan untuk menghitung HMAC lokal tloc berdasarkan pesan msg dan kunci K_MAC. Jika nilai ini tidak sama dengan token

dari manifest, artinya manifest sudah dimodifikasi atau dibuat oleh pihak yang tidak memiliki kunci rahasia, sehingga proses OTA dibatalkan dengan status “HMAC mismatch”.

```
if (ver == CURRENT_VERSION) {
  lastStatus = "Sudah versi terbaru";
  return;
}

lastStatus = "Versi baru " + ver + " (token OK), update...";
doOTAFromURL(fwurl);
```

- Setelah token dinyatakan valid, ESP32 melakukan pengecekan anti-rollback sederhana dengan membandingkan ver terhadap CURRENT_VERSION. Jika versi yang tersedia lebih baru, perangkat melanjutkan proses update menggunakan doOTAFromURL(). Dengan demikian, hanya manifest yang ditandatangani server resmi yang dapat memicu pembaruan firmware.

4.1.2.2.3 Implementasi Sisi Server (Pembuatan Token HMAC)

Di sisi server, pembuatan token digital dilakukan di check_update.php setelah firmware terbaru dipilih dan metadata dasarnya ditentukan.

```
$msg = "ver={$version}"
     . "|size={$size}"
     . "|url={$download_url}"
     . "|nonce={$nonce}"
     . "|ts={$ts}";
```

- Server terlebih dahulu membentuk string msg yang berisi kombinasi informasi versi, ukuran firmware, URL download, nonce acak, dan timestamp saat manifest dibuat. Nilai nonce dan ts berfungsi sebagai proteksi terhadap serangan replay, karena setiap manifest akan memiliki nilai berbeda-beda.

```
$token = hash_hmac('sha256', $msg, $K_MAC);
```

- Baris ini menghitung HMAC-SHA256 dari msg menggunakan kunci rahasia $K_MAC yang sama dengan milik ESP32. Hasilnya disimpan dalam variabel $token, yang kemudian dikirim bersama manifest.

```
echo json_encode([
    "version" => $version,
    "url"     => $download_url,
```

```
    "size"    => $size,
    "nonce"   => $nonce,
    "ts"      => $ts,
    "token"   => $token
], JSON_UNESCAPED_SLASHES);
```

- Manifest yang dikirim ke ESP32 berisi field version, url, size, nonce, ts, dan token. Karena hanya server dan ESP32 yang mengetahui kunci HMAC yang benar, tidak ada pihak lain yang dapat membuat kombinasi field ini menghasilkan token yang sama. Setiap perubahan kecil pada salah satu nilai akan membuat perbandingan HMAC gagal di sisi klien.

4.1.2.2.4 Implementasi Sisi Server (Upload dan Distribusi Firmware)

```
$target_dir = __DIR__ . "/firmware/";

if    ($_SERVER['REQUEST_METHOD']    ===    'POST'    &&
isset($_FILES['firmware'])) {
    $origName = basename($_FILES['firmware']['name']);

    if (!preg_match('/\.bin$/i', $origName)) {
        $err = " Hanya file .bin yang diperbolehkan.";
    } else {
        $destPath = $target_dir . $origName;

move_uploaded_file($_FILES['firmware']['tmp_name'],
$destPath);
    }
}
```

- Potongan ini berada pada upload.php dan digunakan admin untuk mengunggah firmware baru. Server memastikan bahwa hanya file dengan ekstensi .bin yang diterima. Jika valid, file dipindahkan ke direktori firmware/ sebagai rilis resmi yang akan dijadikan sumber OTA.

```
$version = "latest";
if (preg_match('/v[0-9]+\.[0-9]+\.[0-9]+/i', $origName,
$m)) {
    $version = $m[0];
}
```

- Di sini server mengekstrak informasi versi dari nama file, misalnya firmware_v1.2.3.bin menjadi v1.2.3. Informasi versi ini penting untuk menentukan apakah perangkat di lapangan perlu update atau sudah menggunakan firmware terbaru.

```
$base_url     = "http://10.248.38.135/digitaltoken";
$download_url     =     rtrim($base_url,     '/')     .
"/download.php?file=" . rawurlencode($origName);
```

- Server lalu menyusun URL lengkap yang akan digunakan ESP32 untuk mengunduh file firmware. Nilai download_url ini nantinya dimasukkan ke manifest, baik untuk skema SHA-256 maupun skema HMAC.

```
$meta = [
    "version"     => $version,
    "file"        => $origName,
    "size"        => filesize($destPath),
    "url"         => $download_url,
    "uploaded_at" => date('c')
];

file_put_contents(
    $target_dir . "latest.json",
    json_encode($meta,        JSON_PRETTY_PRINT        |
JSON_UNESCAPED_SLASHES)
);
```

- Terakhir, metadata rilis yang berisi versi, nama file, ukuran, URL download, dan waktu upload disimpan ke latest.json. Berkas ini menjadi referensi bagi check_update.php untuk selalu merujuk ke firmware terbaru ketika perangkat memeriksa pembaruan.

```
$firmware_dir  = __DIR__ . "/firmware/";
$manifest_path = $firmware_dir . "latest.json";

$file = $_GET['file'] ?? '';
if ($file === '' || strtolower($file) === 'latest') {
    $meta                                                =
json_decode(file_get_contents($manifest_path), true);
    $file = $meta['file'];
}
```

- Pada download.php, jika parameter file tidak diberikan atau bernilai latest, server secara otomatis mengganti nama file yang akan dikirim dengan nama file terbaru dari latest.json. Dengan begitu, ESP32 cukup meminta latest tanpa harus mengetahui nama berkas firmware yang spesifik.

```
$basename = basename($file);
if (!preg_match('/\.bin$/i', $basename)) {
    http_response_code(400);
    echo "Hanya file .bin yang diizinkan.";
```

```
    exit;
}
```

- Server kemudian memastikan hanya file dengan ekstensi .bin yang boleh diakses melalui endpoint ini. Validasi ini mencegah penyalahgunaan endpoint download.php untuk mengambil file lain yang tidak terkait firmware.

```
$filepath = $firmware_dir . $basename;
$filesize = filesize($filepath);

header('Content-Type: application/octet-stream');
header('Content-Disposition:                attachment;
filename="'.$basename.'"');
header('Content-Length: ' . $filesize);

$fp = fopen($filepath, 'rb');
fpassthru($fp);
fclose($fp);
```

- Terakhir, file firmware dibuka dari disk dan dikirim sebagai stream biner menggunakan fpassthru(). Header Content-Type dan Content-Length memberi informasi tipe dan ukuran file kepada klien. Di sisi ESP32, stream ini dibaca oleh fungsi OTA (doOTAFromURL()) untuk kemudian ditulis ke flash setelah hash SHA-256 dan token HMAC diverifikasi sesuai skenario.

4.1.2.3 Implementasi AES-CTR

4.1.2.3.1 Implementasi Sisi ESP32 (Inisialisasi Kunci dan IV)

```
#include "mbedtls/aes.h"
```

- Baris ini mengaktifkan modul AES di mbedTLS sehingga ESP32 dapat memanggil fungsi-fungsi kriptografi seperti mbedtls_aes_setkey_enc() dan mbedtls_aes_crypt_ctr() untuk melakukan dekripsi firmware yang dikirim server dalam mode AES-128 CTR.

```
static const uint8_t K_ENC[16] = {
  0x00,0x11,0x22,0x33,
  0x44,0x55,0x66,0x77,
  0x88,0x99,0xAA,0xBB,
  0xCC,0xDD,0xEE,0xFF
};
```

- Array K_ENC menyimpan kunci simetris AES-128 sepanjang 16 byte yang digunakan oleh ESP32 untuk mendekripsi firmware terenkripsi. Nilai kunci ini

harus identik dengan kunci yang digunakan di sisi server saat enkripsi; perbedaan satu byte saja akan menyebabkan dekripsi gagal dan firmware tidak dapat dijalankan.

```
if (hex.length() != 32) return false;
```

- Dalam fungsi konversi IV (parseHex16()), panjang string hex diperiksa terlebih dahulu. Nilai IV yang valid harus terdiri dari 32 karakter hex (setara 16 byte). Jika panjangnya tidak sesuai, fungsi langsung mengembalikan false dan proses dekripsi tidak diteruskan.

```
out16[i] = (uint8_t)((hi << 4) | lo);
```

- Baris ini adalah inti konversi IV: dua digit heksadesimal (nibble tinggi hi dan nibble rendah lo) digabung menjadi satu byte biner dan disimpan ke dalam array out16. Hasil array inilah yang digunakan sebagai nonce atau initial counter untuk mode AES-CTR.

4.1.2.3.2 Implementasi Sisi ESP32 (Dekripsi Streaming Firmware)

```
uint8_t iv[16];
if (!parseHex16(ivHex, iv)) return false;
```

- Langkah pertama doOTAFromURLEncrypted() adalah mengubah string IV dari manifest (ivHex) menjadi array biner 16 byte (iv). Jika format IV tidak valid, fungsi langsung mengembalikan false untuk mencegah dekripsi dengan parameter yang salah.

```
HTTPClient http;
if (!http.begin(binURL)) return false;
if (http.GET() != HTTP_CODE_OK) { http.end(); return
false; }

int contentLength = http.getSize();
if (!Update.begin(contentLength > 0 ? contentLength :
UPDATE_SIZE_UNKNOWN)) {
  http.end();
  return false;
}

WiFiClient* stream = http.getStreamPtr();
```

- ESP32 kemudian membuka koneksi HTTP ke URL firmware terenkripsi dan memastikan status respons adalah OK. Fungsi Update.begin() dipanggil untuk

menyiapkan partisi OTA, dengan ukuran sesuai contentLength jika diketahui. Objek stream menjadi sumber data ciphertext yang akan dibaca per chunk selama proses dekripsi.

```
mbedtls_aes_context aes;
mbedtls_aes_init(&aes);
mbedtls_aes_setkey_enc(&aes, K_ENC, 128);

unsigned char nonce_counter[16];
memcpy(nonce_counter, iv, 16);
unsigned char stream_block[16] = {0};
size_t nc_off = 0;
```

- Bagian ini menginisialisasi konteks AES (aes), mengatur kunci enkripsi dengan mbedtls_aes_setkey_enc() untuk AES-128, dan menyalin IV ke nonce_counter sebagai nilai awal counter mode CTR. Variabel stream_block dan nc_off digunakan internal oleh mbedTLS untuk menjaga state dekripsi streaming antar chunk.

```
uint8_t inbuf[2048];
uint8_t outbuf[2048];

while (true) {
  int      n      =      stream->readBytes((char*)inbuf,
sizeof(inbuf));
  if (n < 0) { Update.abort(); break; }
  if (n == 0) {
    if (!stream->available()) break;
    delay(1);
    continue;
  }

  mbedtls_aes_crypt_ctr(&aes, n, &nc_off,
                        nonce_counter, stream_block,
                        inbuf, outbuf);

  if (Update.write(outbuf, n) != (size_t)n) {
    Update.abort(); break;
  }
}
```

- Pada inti loop, ESP32 membaca ciphertext dari jaringan ke buffer inbuf dalam ukuran 2 KB. Fungsi mbedtls_aes_crypt_ctr() digunakan untuk mendekripsi inbuf menjadi plaintext outbuf dengan mode AES-CTR secara streaming. Hasil plaintext kemudian langsung ditulis ke partisi OTA dengan Update.write(). Karena data

diproses per chunk dan tidak pernah disimpan penuh di RAM, metode ini tetap efisien untuk perangkat dengan memori terbatas.

```
http.end();
mbedtls_aes_free(&aes);

if (!Update.end() || !Update.isFinished()) return false;
return true;
```

- Setelah semua data firmware diproses, koneksi HTTP ditutup dan konteks AES dibersihkan. Fungsi Update.end() dan Update.isFinished() memastikan bahwa proses penulisan ke flash selesai dengan benar dan image OTA valid. Jika kedua pengecekan berhasil, fungsi mengembalikan true dan firmware baru siap dijalankan pada reboot berikutnya.

4.1.2.3.3 Implementasi Sisi ESP32 (Memilih Mode OTA: Plain vs AES)

```
HTTPClient http;
if (!http.begin(CHECK_URL)) return;
if (http.GET() != HTTP_CODE_OK) { http.end(); return; }

String body = http.getString();
http.end();
```

- Fungsi checkAndUpdate() kembali digunakan untuk membaca manifest dari server melalui URL CHECK_URL. Jika koneksi gagal atau status HTTP bukan OK, proses pembaruan tidak dilanjutkan.

```
StaticJsonDocument<512> doc;
if (deserializeJson(doc, body)) return;

String latest = doc["version"] | "";
String fwurl  = doc["url"]     | "";
bool   enc    = doc["enc"]     | false;
String ivHex  = doc["ctr_iv"]  | "";
uint32_t size = doc["size"]    | 0;
```

- Manifest kemudian diparsing untuk mengambil informasi versi, URL firmware, flag enc (apakah firmware terenkripsi), nilai IV dalam bentuk hex (ctr_iv), dan ukuran file. Field enc menjadi penanda utama apakah perangkat harus menjalankan OTA biasa atau OTA dengan dekripsi AES.

```
if (latest == "" || fwurl == "") return;
if (latest == CURRENT_VERSION)  return;
```

- ESP32 memastikan bahwa versi dan URL tidak kosong serta hanya melanjutkan update jika versi di server berbeda dari firmware yang sedang digunakan. Ini mencegah perangkat berulang kali mengunduh firmware yang sama.

```
if (enc) {
  doOTAFromURLEncrypted(fwurl, ivHex, size);
} else {
  doOTAFromURL(fwurl);     // OTA plain (tanpa enkripsi)
}
```

- Berdasarkan nilai enc, perangkat memutuskan mode OTA yang akan dipakai. Jika enc = true, fungsi doOTAFromURLEncrypted() dipanggil sehingga data firmware akan didekripsi menggunakan AES-CTR sebelum ditulis ke flash. Jika enc = false, perangkat menggunakan OTA biasa tanpa enkripsi melalui doOTAFromURL(). Dengan pendekatan ini, pengubahan skenario pengujian cukup dilakukan di sisi server dengan memodifikasi manifest, tanpa perlu mengganti firmware ESP32.

4.1.2.3.4 Implementasi Sisi Server (Enkripsi Firmware dan Manifest AES)

```
// Kunci AES-128 (HARUS sama dengan di ESP32)
$K_ENC = pack('H*', '00112233445566778899AABBCCDDEEFF');

// Baca firmware plaintext dari upload
$plain                                                    =
file_get_contents($_FILES['firmware']['tmp_name']);
```

- Pada upload.php untuk mode terenkripsi, server mengonversi representasi heksadesimal kunci menjadi 16 byte dengan pack('H*', ...) sehingga formatnya sesuai dengan K_ENC di ESP32. File firmware .bin yang diunggah admin dibaca sebagai plaintext ke dalam variabel $plain.

```
$iv = rand16();   // IV random 16 byte

$cipher = openssl_encrypt(
    $plain,
    'aes-128-ctr',
    $K_ENC,
    OPENSSL_RAW_DATA,
    $iv
);
```

- Fungsi rand16() menghasilkan IV acak 16 byte untuk setiap rilis firmware, yang penting untuk menjaga keamanan mode CTR. Fungsi openssl_encrypt() lalu digunakan untuk mengenkripsi firmware dengan algoritma aes-128-ctr, kunci

$K_ENC, dan IV acak tersebut, menghasilkan ciphertext mentah dalam variabel $cipher.

```
$encName  = $origName . ".enc";
$destPath = $target_dir . $encName;
file_put_contents($destPath, $cipher);
```

- Hasil enkripsi kemudian disimpan ke file baru dengan nama <nama_asli>.bin.enc di direktori firmware/. File .enc inilah yang nantinya akan diunduh oleh ESP32 dan didekripsi secara streaming menggunakan AES-CTR.

```
$meta = [
  "ok"      => true,
  "version" => $version,
  "file"    => $encName,
  "enc"     => true,
  "ctr_iv"  => bin2hex($iv),  // IV dikirim dalam bentuk
hex
  "size"    => strlen($cipher),
  "url"        => $base_url . "/download.php?file=" .
rawurlencode($encName)
];

file_put_contents(
  $target_dir . "latest.json",
  json_encode($meta,         JSON_PRETTY_PRINT         |
JSON_UNESCAPED_SLASHES)
);
```

- Terakhir, server menyusun manifest latest.json khusus untuk mode terenkripsi. Di sini, flag enc diset true, nama file ciphertext, ukuran ciphertext, dan URL download disimpan bersama. IV acak dikirim dalam bentuk string hex (ctr_iv) agar bisa diparsing kembali oleh ESP32 menjadi 16 byte. Manifest inilah yang dibaca ESP32 untuk memutuskan bahwa mode OTA yang digunakan adalah AES-CTR dan untuk menginisialisasi dekripsi dengan IV yang benar.

```
$firmware_dir  = __DIR__ . "/firmware/";
$file          = $_GET['file'] ?? '';

$basename = basename($file);
if (!preg_match('/\.bin\.enc$/i', $basename)) {
    http_response_code(400);
    echo  "Hanya  file  terenkripsi  (.bin.enc)  yang
diizinkan.";
    exit;
}
```

```
$filepath = $firmware_dir . $basename;
if (!is_file($filepath)) {
    http_response_code(404);
    echo "Firmware tidak ditemukan.";
    exit;
}

$filesize = filesize($filepath);
header('Content-Type: application/octet-stream');
header('Content-Disposition:                attachment;
filename="'.$basename.'"');
header('Content-Length: '.$filesize);

$fp = fopen($filepath, 'rb');
fpassthru($fp);
fclose($fp);
```

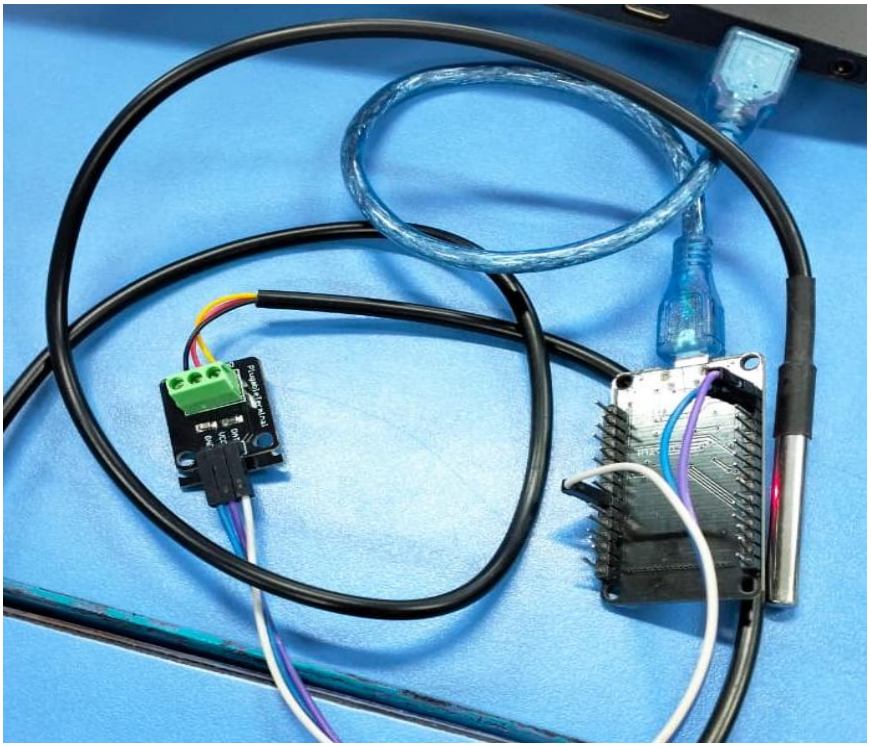
- Endpoint download.php untuk mode AES memastikan hanya file dengan ekstensi .bin.enc yang dapat diunduh, memeriksa keberadaan file, kemudian mengirim ciphertext sebagai stream biner ke klien. Di sisi ESP32, stream ini dibaca chunk demi chunk oleh doOTAFromURLEncrypted(), didekripsi dengan AES-CTR, dan langsung ditulis ke partisi OTA hingga proses selesai.

4.2 Hasil Implementasi Pengiriman OTA

Bagian ini membahas hasil implementasi proses pembaruan firmware ESP32 melalui mekanisme Over-The-Air (OTA). Pengujian dilakukan secara langsung dengan cara mengunggah firmware ke server web, melakukan pengecekan update, dan mengunduh firmware ke perangkat ESP32 melalui jaringan WiFi. Hasil pengujian dipaparkan berdasarkan tahapan proses OTA yang terjadi pada sistem.

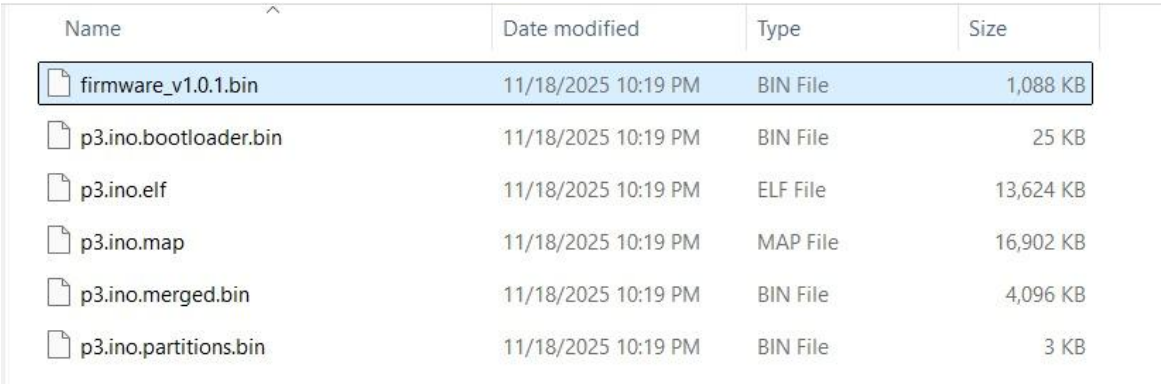
4.2.1 Tahap Pengunggahan Firmware ke Server OTA

Tahap awal proses pembaruan dimulai setelah firmware versi baru selesai dikompilasi menggunakan Arduino IDE. File hasil kompilasi kemudian diekspor menjadi berkas biner (.bin), karena hanya format biner yang dapat dibaca oleh mekanisme OTA ESP32. Berkas tersebut kemudian ditempatkan pada direktori server lokal yang digunakan oleh website OTA (misalnya melalui XAMPP/Apache).



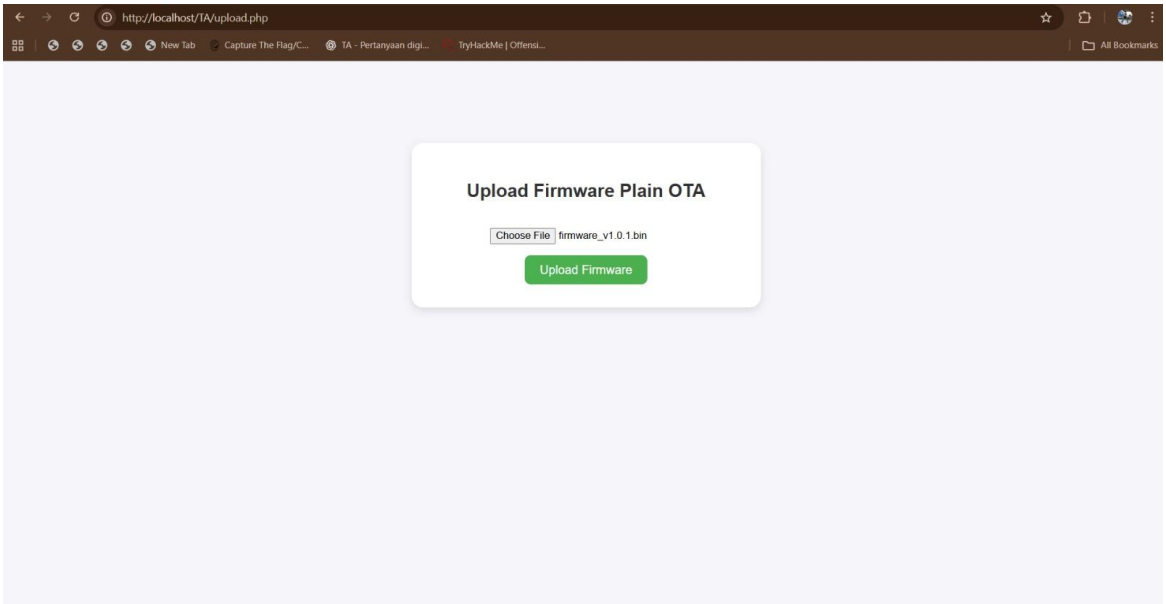
Gambar 4. 2 Rangkaian ESP32 dengan Sensor Suhu

Selanjutnya, pengguna membuka halaman web OTA uploader melalui browser. Pada halaman ini tersedia form unggah (upload form) untuk mengirim file firmware terbaru ke server. Pengguna memilih file binary firmware tersebut, kemudian menekan tombol Upload.



Gambar 4. 3 File Firmware yang tersimpan di Server Lokal

Setelah berhasil diunggah, file firmware disimpan di direktori server, namun belum dipasang pada ESP32. File ini hanya menjadi kandidat update yang akan dikirimkan kepada ESP32 saat proses OTA dimulai.



Gambar 4. 4 Website Upload Firmware

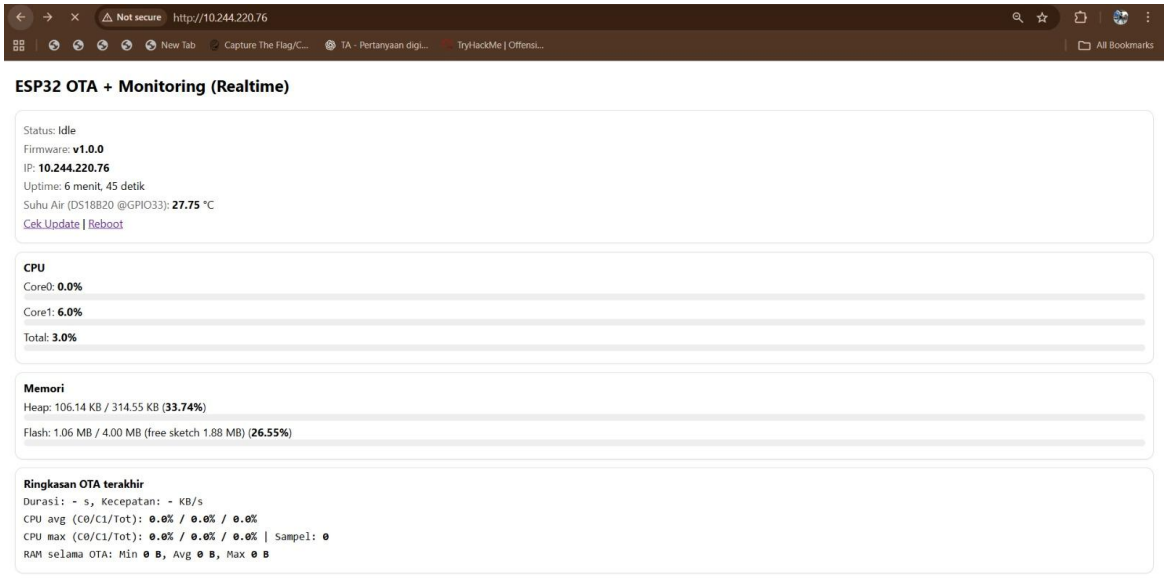
4.2.2 Tahap Pemeriksaan Ketersediaan Update (Check Update)

Setelah file firmware baru tersedia di server, pengguna menekan tombol Cek Update pada halaman web ESP32. Pada tahap ini, proses berikut terjadi:

- Website server memeriksa apakah terdapat firmware terbaru di direktori server.
- Jika tersedia, server mengirimkan metadata firmware (versi, ukuran, URL file) kepada ESP32.
- ESP32 kemudian membandingkan versi firmware saat ini dengan versi terbaru yang tersedia.

- Jika versi lebih baru terdeteksi, proses OTA akan dimulai.

Pada tahap ini, mekanisme keamanan yang ditanamkan dalam firmware, seperti SHA-256, HMAC-SHA256, atau AES-CTR dan mulai diterapkan. Setiap algoritma akan berfungsi sesuai tugasnya, seperti memverifikasi integritas hash, memvalidasi token, atau mendekripsi firmware.



Gambar 4. 5 Tahap Pemeriksaan Ketersediaan Update

4.2.3 Tahap Pengunduhan Firmware ke ESP32 (Download OTA)

Jika versi update valid, ESP32 mulai mengunduh firmware dari server lokal melalui jaringan WiFi. Proses pengunduhan dilakukan secara streaming sehingga file tidak perlu disimpan penuh di RAM.

Selama proses download ini berlangsung:

- Firmware dikirim dari server dalam bentuk chunk data.
- ESP32 menulis chunk tersebut ke partisi OTA di flash memory.
- Pada saat yang sama, setiap algoritma keamanan aktif akan bekerja:
  - SHA-256 menghitung hash dari setiap chunk yang diterima.
  - HMAC-SHA256 memverifikasi token manifest.
  - AES-CTR mendekripsi setiap chunk firmware sebelum ditulis ke flash.

Pada tahap inilah sistem juga melakukan pencatatan resource seperti:

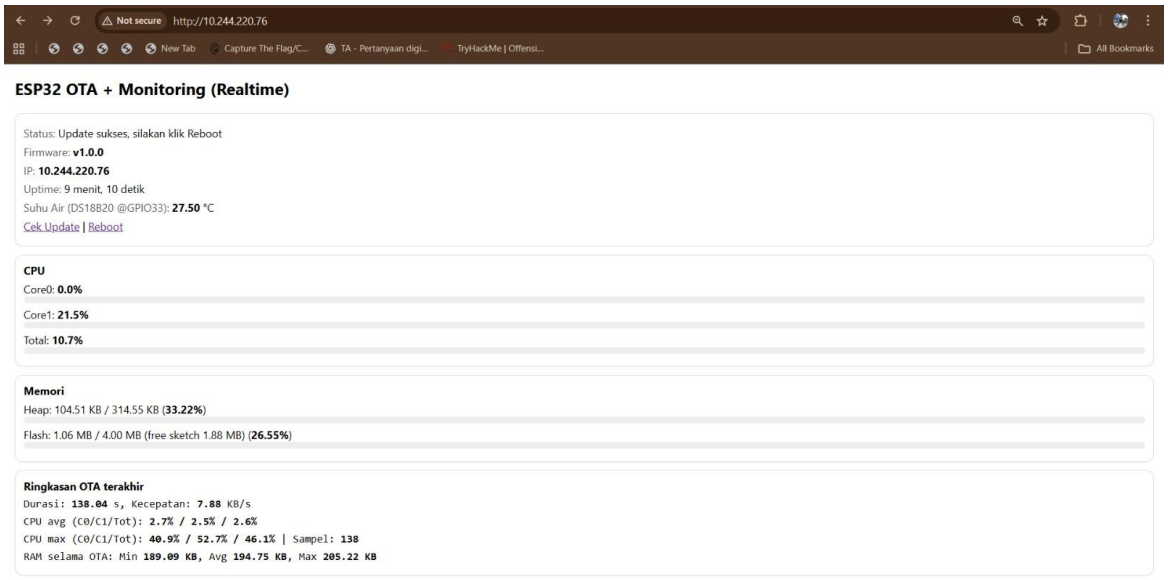
- waktu total download,
- penggunaan RAM,

- penggunaan CPU,

Setelah seluruh firmware berhasil ditulis, halaman web ESP32 akan menampilkan status:

“Update sukses, silakan klik reboot.”

Firmware telah tersimpan lengkap, tetapi belum dijalankan karena masih berada pada partisi OTA yang belum aktif.



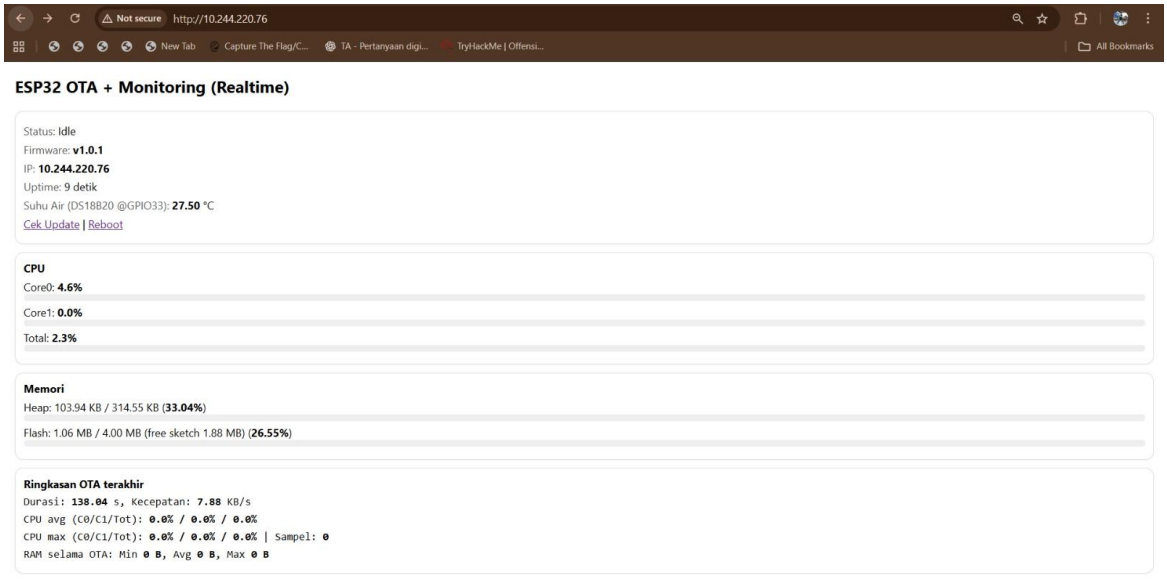
Gambar 4. 6 Tahap Pengunduhan Firmware ke ESP32

4.2.4 Aktivasi Firmware Baru (Reboot & Verifikasi Versi)

Tahap terakhir adalah menekan tombol Reboot pada halaman web. Ketika perintah ini dikirim:

- ESP32 melakukan restart.
- Bootloader mengalihkan partisi boot ke partisi OTA yang baru diisi.
- Firmware terbaru mulai dijalankan.

Setelah reboot selesai, halaman web ESP32 diperbarui dan akan menampilkan informasi versi firmware yang baru. Jika proses berhasil, maka versi firmware yang tampil di halaman web akan berubah sesuai versi update yang dikirimkan. Dengan demikian, pembaruan OTA telah selesai dan ESP32 kini menjalankan firmware versi terbaru tanpa memerlukan koneksi kabel ke komputer.



Gambar 4. 7 Aktivasi Firmware Baru

4.3 Hasil Pengukuran Resource Selama OTA

Bagian ini membahas hasil pengukuran penggunaan sumber daya (resource) pada ESP32 selama proses pembaruan firmware melalui mekanisme Over-The-Air (OTA). Pengujian dilakukan untuk mengetahui pengaruh metode keamanan terhadap performa perangkat saat proses update berlangsung.

Parameter yang diukur meliputi:

- Waktu pembaruan (durasi)
- Penggunaan RAM dan CPU
- Perbandingan efisiensi antar metode

Empat metode OTA digunakan dalam pengujian, yaitu:

- OTA tanpa keamanan (Plain)
- OTA dengan SHA-256
- OTA dengan HMAC-SHA256
- OTA dengan AES-CTR

Untuk memastikan hasil yang objektif, setiap metode diujikan beberapa kali dan dihitung nilai rata-rata (mean) dan median, sehingga hasil pengujian tidak dipengaruhi nilai ekstrem (misalnya gangguan Wi-Fi atau lonjakan CPU). Nilai rata-rata (mean) digunakan untuk melihat kecenderungan umum dari setiap parameter yang diukur. Rumus mean diberikan sebagai berikut:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Keterangan:

- $\bar{x}$ = nilai rata-rata (mean)
- x_i = data ke-i dari total pengukuran
- $\sum_{i=1}^{n} x_i$ = penjumlahan seluruh nilai data dari x_1 sampai x_n
- n = jumlah data

Sedangkan median digunakan untuk menilai kestabilan hasil pengukuran dan menghindari bias akibat lonjakan nilai tertentu. Rumus median dituliskan sebagai:

$$Median = \begin{cases} x_{\frac{n+1}{2}} & n\ ganjil \\ \frac{x_{\frac{n}{2}} + x_{\frac{n+1}{2}}}{2} & n\ genap \end{cases}$$

Keterangan:

- median : nilai tengah dari data yang telah diurutkan
- n = jumlah data
- $x_{\frac{n+1}{2}}$ = data dari posisi tengah jika jumlah data ganjil
- $\frac{x_{\frac{n}{2}} + x_{\frac{n+1}{2}}}{2}$ = rata rata dua nilai tengah jika dia genap

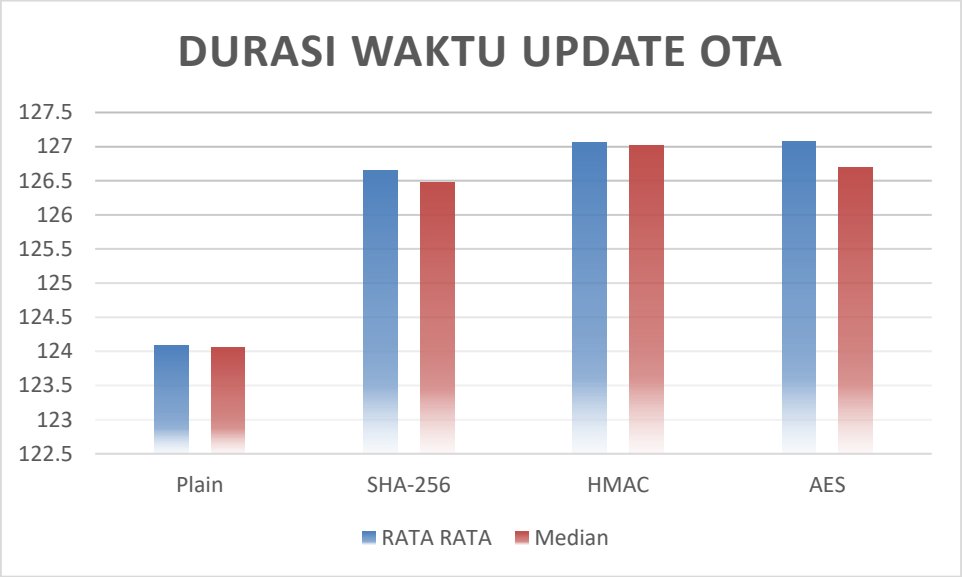
4.3.1 Durasi Waktu Update OTA

Durasi dihitung mulai dari proses pengunduhan firmware dimulai hingga proses penulisan selesai pada partisi OTA ESP32.

Table 60 Perbandingan Durasi OTA (detik)

No	Plain	SHA-256	HMAC	AES
1	124.05	125.98	126.59	125.34
2	124.33	125.78	126.70	126.21
3	123.35	125.90	126.71	127.05
4	123.63	126.06	126.77	127.57
5	124.62	125.95	126.87	126.70
6	124.19	126.36	126.91	126.17
7	124.91	127.10	127.00	125.95
8	124.05	126.98	127.01	127.70

9	124.33	126.47	127.18	126.01
10	123.20	127.82	127.23	133.08
11	124.33	126.99	127.25	126.17
12	123.91	126.82	127.26	127.35
13	125.05	127.03	127.30	126.30
14	123.49	128.28	127.31	127.49
15	123.91	126.19	127.85	127.07



Gambar 4. 8 Grafik Durasi Waktu Update

Grafik durasi waktu update OTA menunjukkan perbandingan waktu yang dibutuhkan ESP32 untuk menyelesaikan proses pembaruan firmware pada empat metode yang berbeda, yaitu Plain OTA, SHA-256, HMAC-SHA256, dan AES-CTR. Secara umum, Plain OTA memiliki durasi pembaruan paling cepat, dengan rata-rata sekitar 124,32 detik, karena tidak terdapat proses kriptografi tambahan selama pengunduhan maupun penulisan firmware. Metode SHA-256 berada sedikit di atas Plain OTA, dengan durasi rata-rata 126,62 detik. Kenaikan waktu ini disebabkan oleh proses perhitungan hash integritas yang dilakukan secara streaming selama firmware diunduh.

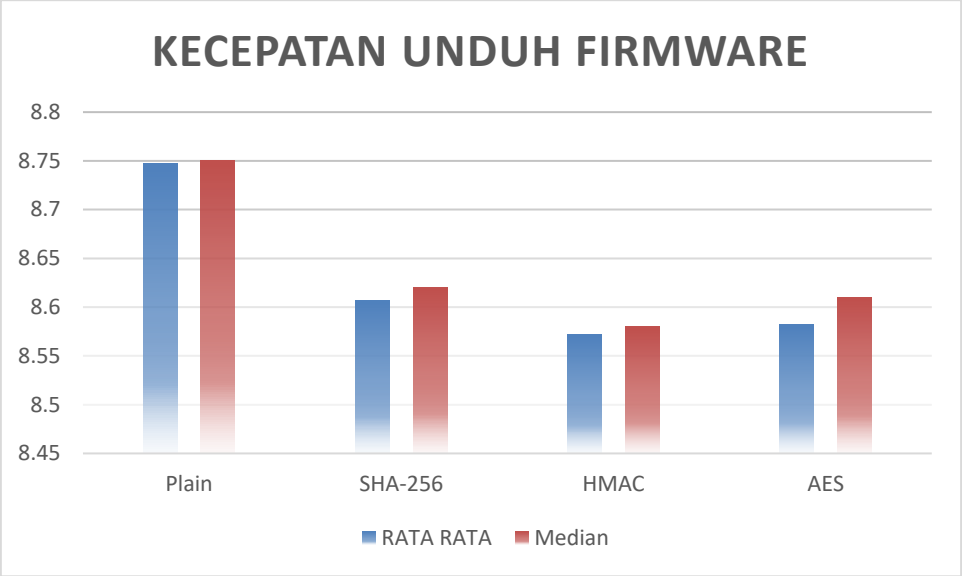
Selanjutnya, metode HMAC-SHA256 menjadi yang paling lambat, dengan durasi rata-rata 127,0 detik. Hal ini terjadi karena selain menghitung hash, HMAC juga membutuhkan pemrosesan tambahan berupa verifikasi token menggunakan kunci rahasia, sehingga menambah beban komputasi. Sementara itu, algoritma AES-CTR berada sedikit di bawah

HMAC, dengan waktu rata-rata 126,96 detik. Meskipun AES melibatkan proses dekripsi yang lebih berat dibanding hashing, penggunaan dukungan akselerasi hardware pada ESP32 mampu menekan waktu eksekusi sehingga performanya tidak seburuk HMAC. Perbandingan ini menunjukkan bahwa semakin kompleks mekanisme keamanan yang diterapkan, semakin besar waktu pembaruan yang diperlukan, namun pengaruhnya juga ditentukan oleh dukungan hardware kriptografi pada perangkat IoT seperti ESP32.

4.3.2 Kecepatan Unduh Firmware (kb/s)

Table 61 Kecepatan Unduh Firmware

No	Plain	SHA-256	HMAC	AES
1	8.75	8.65	8.60	8.70
2	8.73	8.67	8.60	8.64
3	8.80	8.66	8.60	8.58
4	8.78	8.65	8.59	8.55
5	8.71	8.65	8.58	8.61
6	8.74	8.63	8.58	8.64
7	8.69	8.58	8.58	8.66
8	8.75	8.58	8.58	8.54
9	8.73	8.62	8.56	8.65
10	8.81	8.53	8.56	8.20
11	8.73	8.58	8.56	8.64
12	8.76	8.59	8.56	8.56
13	8.68	8.58	8.56	8.63
14	8.79	8.50	8.55	8.55
15	8.76	8.64	8.52	8.58



Gambar 4. 9 Grafik Kecepatan Unduh Firmware

Grafik memperlihatkan perbandingan kecepatan unduh firmware (kb/s) pada empat mekanisme pembaruan OTA, yaitu Plain OTA, SHA-256, HMAC-SHA256, dan AES-CTR. Berdasarkan grafik, Plain OTA menghasilkan kecepatan unduh tertinggi, baik pada nilai rata-rata maupun median. Hal ini disebabkan karena Plain OTA tidak melakukan proses enkripsi, hashing, maupun validasi token, sehingga seluruh sumber daya CPU dapat difokuskan pada proses transfer data tanpa tambahan latensi komputasi.

Metode SHA-256 dan HMAC-SHA256 menunjukkan penurunan kecepatan dibanding Plain. Penurunan ini terjadi akibat proses hashing dan pembuatan/verifikasi token pada saat firmware diunduh, sehingga sebagian waktu CPU digunakan untuk perhitungan kriptografi. Dibanding SHA-256 biasa, HMAC-SHA256 sedikit lebih lambat, karena HMAC memerlukan dua lapisan hashing (inner dan outer hash) yang meningkatkan overhead komputasi.

Sementara itu, AES-CTR menunjukkan performa kecepatan yang lebih baik dibanding HMAC, namun tetap lebih rendah dari Plain OTA. AES mengalami overhead karena proses dekripsi streaming pada saat firmware diterima, namun dukungan hardware acceleration pada ESP32 membantu mengurangi penurunan performanya. Oleh karena itu, posisi kecepatan AES berada di antara HMAC dan SHA-256. Secara keseluruhan, grafik menunjukkan bahwa semakin kompleks algoritma keamanan yang diterapkan, semakin besar penurunan kecepatan unduh firmware OTA. Namun, perbedaan ini masih dalam

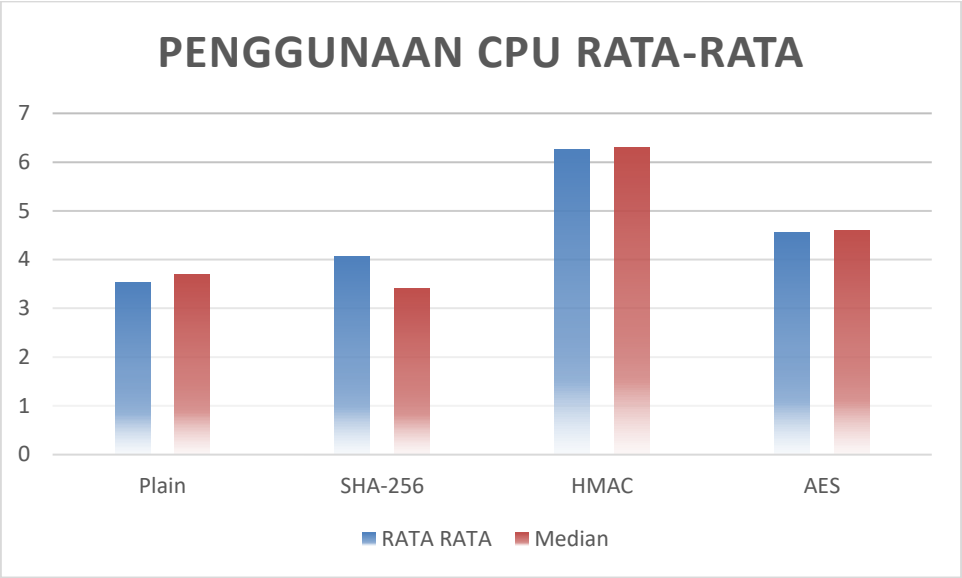
rentang kecil sehingga masih dapat diterima dalam implementasi IoT yang memerlukan keamanan tinggi.

4.3.3 Pengukuran Penggunaan CPU Rata-rata (CPU Avg %)

Penggunaan CPU rata-rata menunjukkan beban proses yang berjalan selama OTA. Semakin kompleks metode keamanan (hashing, autentikasi, enkripsi), semakin tinggi penggunaan CPU.

Table 62 Pengukuran Penggunaan CPU Rata rata

No	Plain	SHA-256	HMAC	AES
1	3.0	3.2	5.9	4.8
2	3.4	3.0	5.2	4.6
3	3.7	3.4	5.6	4.6
4	3.8	3.3	6.9	4.6
5	3.8	4.9	6.5	5.6
6	3.6	4.5	5.7	4.6
7	2.6	3.3	7.9	4.9
8	3.0	3.2	4.6	4.2
9	3.8	5.1	6.4	4.5
10	3.9	4.1	6.2	4.4
11	3.7	5.9	5.7	5.2
12	4.0	3.1	6.4	3.9
13	4.4	5.7	7.1	3.9
14	3.3	2.8	7.4	4.8
15	3.1	5.6	6.3	3.8



Gambar 4. 10 Grafik Penggunaan CPU Rata Rata

Grafik penggunaan CPU rata-rata menunjukkan seberapa besar beban pemrosesan yang terjadi pada ESP32 selama proses pembaruan firmware OTA. Berdasarkan hasil pengujian, metode Plain OTA memiliki penggunaan CPU rata-rata paling rendah, yaitu sekitar 3,5%, karena proses yang dilakukan hanya sebatas mengunduh dan menulis firmware ke memori flash tanpa adanya proses kriptografi tambahan. Selanjutnya, metode SHA-256 menunjukkan kenaikan penggunaan CPU menjadi sekitar 4%, disebabkan adanya proses hashing untuk verifikasi integritas data firmware yang menambah beban perhitungan matematis pada prosesor. Metode AES-CTR memiliki penggunaan CPU yang lebih tinggi lagi, sekitar 4,6%, akibat proses enkripsi/dekripsi yang memerlukan operasi blok cipher berulang secara streaming. Penggunaan CPU tertinggi terjadi pada metode HMAC-SHA256, yaitu mencapai rata-rata 6,2%, karena metode ini tidak hanya melakukan hashing, tetapi juga menambahkan proses autentikasi berbasis kunci rahasia (HMAC), sehingga membutuhkan komputasi lebih besar dibandingkan algoritma lainnya.

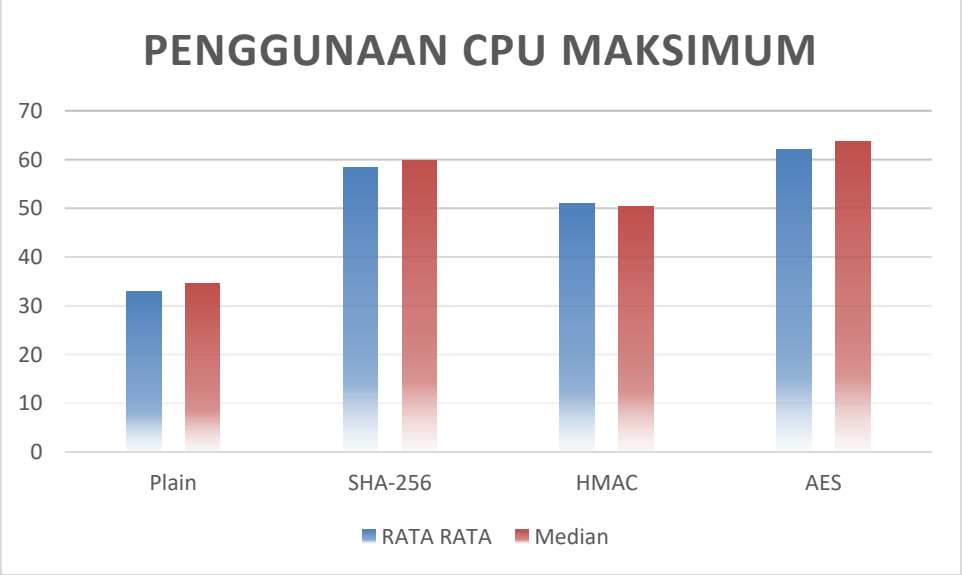
Dengan demikian, grafik ini menunjukkan bahwa semakin kompleks mekanisme keamanan yang digunakan, semakin tinggi pula beban CPU pada ESP32 saat proses OTA. Hal ini terlihat jelas pada metode HMAC-SHA256 dan AES, di mana proses kriptografi memiliki dampak langsung terhadap peningkatan konsumsi sumber daya pemrosesan dibandingkan metode Plain OTA yang tidak mengandung teknik keamanan sama sekali.

4.3.4 Penggunaan CPU Maksimum (CPU Max %)

CPU max mencerminkan “lonjakan beban puncak” selama proses OTA, terutama ketika aktivitas hashing atau dekripsi terjadi.

Table 63 Penggunaan CPU Maksimum

No	Plain	SHA-256	HMAC	AES
1	40.2	68.5	58.7	71.9
2	30.6	67.3	51.6	69.2
3	27.9	65.1	50.3	66.4
4	37.7	64.3	51.1	64.2
5	34.6	61.0	55.2	65.4
6	32.2	60.5	46.2	65.9
7	20.4	60.1	50.1	64.7
8	35.9	59.9	53.3	58.4
9	36.3	50.3	55.5	61.6
10	29.5	54.7	53.5	61.1
11	18.6	51.9	48.0	63.8
12	36.9	56.8	48.0	54.3
13	27.8	53.3	48.5	54.1
14	47.8	55.2	48.2	51.3
15	36.75	47.5	46.7	57.3



Gambar 4. 11 Grafik Penggunaan CPU Maksimum

Grafik penggunaan CPU maksimum menunjukkan besarnya lonjakan beban pemrosesan tertinggi yang terjadi pada ESP32 selama proses pembaruan firmware OTA dengan berbagai algoritma keamanan. Dari grafik tersebut terlihat bahwa metode Plain OTA memiliki nilai maksimum paling rendah, yaitu di kisaran 32–35%, karena proses ini hanya melakukan pengunduhan dan penulisan firmware tanpa adanya tahapan hashing, autentikasi token, maupun dekripsi. Hal ini menunjukkan bahwa Plain OTA hampir tidak menambahkan beban komputasi di luar aktivitas I/O dasar pada flash.

Sebaliknya, metode dengan keamanan kriptografis memperlihatkan peningkatan tajam pada beban CPU maksimum. SHA-256 menunjukkan peningkatan puncak CPU hingga sekitar 58–60%, karena perangkat harus melakukan proses hashing blok demi blok untuk memverifikasi integritas firmware yang di-download. Hal ini membutuhkan operasi matematika bitwise dan transformasi berulang, sehingga memicu lonjakan komputasi dalam beberapa interval waktu. Pada mekanisme HMAC-SHA256, penggunaan CPU maksimum berada pada kisaran 49–55%, sedikit lebih rendah dibanding SHA-256. Hal ini karena HMAC hanya melakukan hashing pada data tertentu menggunakan kunci, bukan pada seluruh file firmware secara penuh saat proses verifikasi berjalan, sehingga intensitas operasi hashing menjadi lebih singkat.

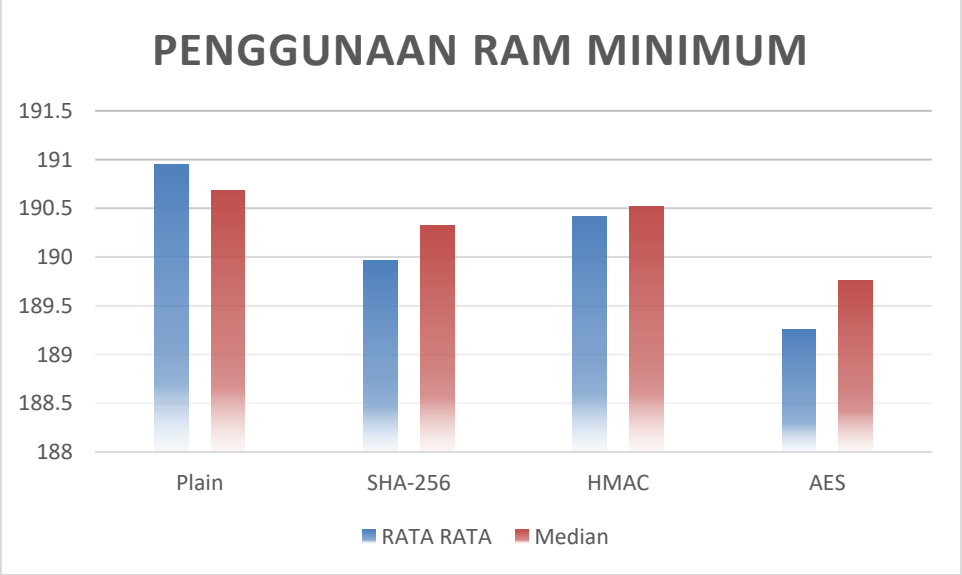
Metode AES-CTR memiliki lonjakan CPU tertinggi, yaitu sekitar 62–64%, karena proses dekripsi AES dilakukan secara streaming pada setiap blok firmware yang diterima. Proses enkripsi simetris memerlukan transformasi matriks, substitusi byte, dan operasi XOR berulang pada setiap blok data. Meskipun ESP32 memiliki akselerator hardware AES, beban komputasi tetap meningkat terutama saat buffer data lebih besar dan proses berlangsung secara kontinu.

4.3.5 Penggunaan RAM (MAX/MIN/AVG)

RAM minimum menunjukkan penggunaan memori terendah selama proses OTA.

Table 64 Penggunaan RAM (MAX/MIN/AVG)

No	Plain	SHA-256	HMAC	AES
1	190.73	190.13	190.46	185.28
2	190.61	191.30	190.98	190.05
3	190.63	190.15	189.80	186.17
4	190.43	190.07	189.70	189.88
5	190.61	188.91	191.08	189.76
6	191.89	190.32	191.14	189.37
7	192.30	185.05	190.52	189.64
8	192.20	191.76	189.83	190.73
9	190.68	190.43	190.84	189.81
10	190.66	190.55	189.52	189.82
11	189.90	190.73	190.66	189.85
12	190.47	186.64	191.14	189.58
13	190.70	191.53	189.71	189.84
14	191.45	191.54	189.91	189.58
15	191.01	190.31	190.88	189.50



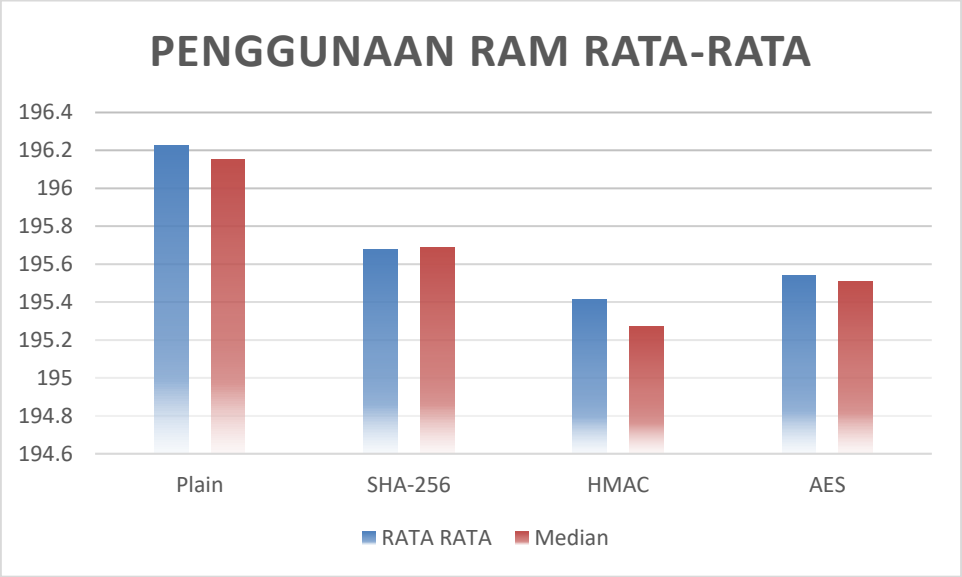
Gambar 4. 12 Grafik Penggunaan RAM Minimum

Grafik Penggunaan RAM Minimum memperlihatkan bahwa Plain memiliki nilai RAM minimum terbesar dibanding algoritma lain. Nilai minimum menunjukkan titik terendah sisa RAM selama OTA, dan Plain tetap menunjukkan sisa RAM tertinggi karena hanya memproses download dan write tanpa melakukan operasi kriptografi. Pada metode keamanan, nilai RAM minimum lebih rendah karena pada fase tertentu, sistem harus mengalokasikan blok memori sementara untuk hashing (SHA-256), token verification (HMAC), atau dekripsi (AES), sehingga sisa RAM turun pada titik tertentu.

Berikut merupakan data untuk table penggunaan RAM rata rata.

Table 65 Penggunaan RAM Rata rata

No	Plain	SHA-256	HMAC	AES
1	196.18	195.57	195.20	195.60
2	196.09	196.56	195.35	195.45
3	196.15	195.66	195.27	195.95
4	196.14	195.69	196.45	195.42
5	196.11	195.67	195.22	195.44
6	196.08	195.66	196.34	195.55
7	196.18	193.94	195.31	195.50
8	197.19	196.53	195.25	195.81
9	196.06	195.76	195.20	195.48
10	196.17	195.74	195.29	195.56
11	196.17	195.75	195.19	195.54
12	196.13	195.63	195.24	195.29
13	196.15	195.70	195.31	195.50
14	196.43	195.69	195.27	195.51
15	196.13	195.64	195.33	195.53



Gambar 4. 13 Grafik Penggunaan RAM Rata Rata

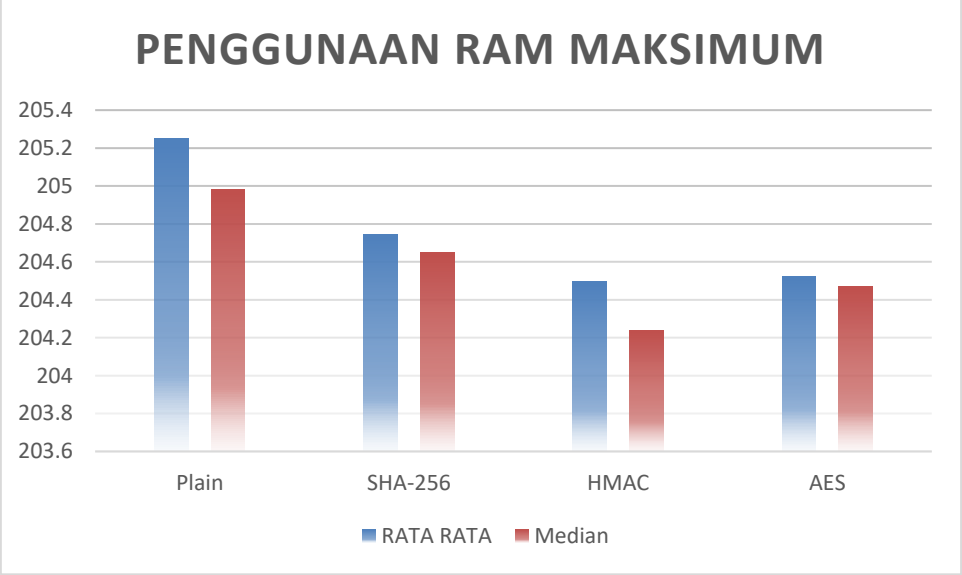
Grafik Penggunaan RAM Rata-Rata memperlihatkan bahwa metode Plain OTA memiliki nilai RAM rata-rata tertinggi dibanding ketiga metode OTA yang menggunakan keamanan (SHA-256, HMAC, dan AES). Hal ini menunjukkan bahwa selama proses update berlangsung, Plain tidak membutuhkan alokasi memori tambahan sehingga sisa RAM lebih besar. Berbeda dengan SHA-256 dan AES yang melakukan hashing dan dekripsi secara streaming sehingga membutuhkan buffer tambahan, dan lebih jelas lagi pada HMAC-SHA256 yang membutuhkan memori untuk verifikasi token. Akibatnya, ketiga algoritma tersebut membuat sisa RAM lebih rendah daripada Plain.

RAM maksimum menunjukkan puncak memori ketika aktivitas paling berat terjadi (hash finalize, outer HMAC, atau blok AES).

Table 66 Penggunaan RAM Maksimum

No	Plain	SHA-256	HMAC	AES
1	204.46	204.02	204.16	204.74
2	204.96	206.40	205.55	204.46
3	204.98	204.66	204.24	205.82
4	204.95	204.64	204.93	204.44
5	204.96	204.61	204.88	204.69
6	204.98	204.46	204.23	203.85
7	205.82	204.71	204.46	204.95

8	206.94	205.74	204.59	204.65
9	204.59	204.69	204.20	203.92
10	204.98	204.66	204.89	204.67
11	205.21	204.24	204.20	203.74
12	205.22	205.06	203.87	204.47
13	205.86	204.61	203.86	204.46
14	205.01	204.03	204.20	205.12
15	205.81	204.65	205.16	203.89



Gambar 4. 14 Penggunaan RAM Maksimum

Grafik Penggunaan RAM Maksimum menampilkan penggunaan puncak RAM ketika aktivitas berat terjadi dalam proses OTA. Plain menunjukkan nilai RAM maksimum tertinggi karena tidak ada proses kriptografi yang mengakibatkan lonjakan memori. Pada metode keamanan, puncak penggunaan RAM lebih rendah karena pada saat hashing, token verification, atau dekripsi, sebagian memori dialokasikan sebagai buffer kerja sehingga mengurangi sisa RAM yang terbaca. AES sedikit fluktuatif dibanding HMAC dan SHA karena proses dekripsi dilakukan bertahap dengan double buffer in/out, sehingga RAM maksimum tidak setinggi Plain.

Dari nilai RAM minimum, maksimum, dan rata-rata, tampak bahwa Plain selalu memiliki sisa RAM terbesar karena:

$$Sisa\ RAM = Total\ RAM - (RAM\ OTA\ dasar + RAM\ Kriptografi)$$

Pada Plain:

$$RAM\ Kriptografi = 0$$

Sehingga:

$$Sisa\ RAM_{Plain} = Total\ RAM - RAM\ OTA$$

Pada OTA dengan keamanan:

$$Sisa\ RAM_{Secure} = Total\ RAM - (RAM\ OTA\ dasar + Buffer\ Hash/Token/AES)$$

Dengan kata lain, semakin banyak lapisan keamanan, semakin besar buffer tambahan, dan semakin kecil sisa RAM yang tercatat.

4.4 Hasil Hipotesa Algoritma

Table 67 Pembuktian Hipotesis Penggunaan Resource

Aspek Pengujian	Plain OTA	SHA-256	HMAC-SHA256	AES-CTR	Kesimpulan Hipotesis
Durasi OTA	Paling cepat	Lambat sedikit	Paling lambat	Dekat HMAC	Semakin kompleks algoritma → semakin lama durasi pembaruan
Kecepatan Unduh	Tertinggi	Tinggi	Menurun paling besar	Menurun sedang	Perhitungan kriptografi menurunkan throughput unduh
CPU Max (puncak)	30–40%	55–60%	49–55%	60–65%	Algoritma kriptografi memicu lonjakan CPU; AES paling tinggi
RAM	Sisa RAM tertinggi	Lebih rendah	Lebih rendah	Fluktuatif, sedikit lebih tinggi dibanding HMAC	Buffer kriptografi menyerap RAM tambahan
Keamanan	Tidak ada	Integritas	Integritas + Autentikasi	Kerahasiaan (Enkripsi)	Kompleksitas keamanan berbanding

Aspek Pengujian	Plain OTA	SHA-256	HMAC-SHA256	AES-CTR	Kesimpulan Hipotesis
					lurus dengan beban sumber daya

Table 68 Pembuktian Hipotesis Efisiensi Algoritma

Kriteria Evaluasi	SHA-256	HMAC-SHA256	AES-CTR	Kesimpulan Efisiensi
CPU Max	Lebih rendah dari AES (55–60%)	Lebih rendah dari AES (49–55%)	**Tertinggi** (60–65%)	SHA-256 dan HMAC lebih efisien daripada AES untuk peak CPU
Durasi OTA	Lebih cepat	Paling lambat	Sedikit lebih lambat dari SHA-256	SHA-256 paling cepat di antara algoritma aman
RAM	Stabil	Stabil	Sedikit lebih tinggi	Perbedaan RAM tidak signifikan; ketiganya aman digunakan
Karakter Keamanan	Integritas	Autentikasi + Integritas	Enkripsi (Kerahasiaan)	Pilihan tergantung kebutuhan keamanan
Kesimpulan Akhir	**Paling efisien untuk IoT terbatas**	Dipilih jika autentikasi kuat dibutuhkan	Dipilih jika kerahasiaan firmware diperlukan	SHA-256 merupakan pilihan paling optimal secara keseluruhan

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian ini berhasil mengimplementasikan sistem pembaruan firmware Over-The-Air (OTA) pada ESP32 menggunakan web server berbasis PHP, di mana seluruh proses update mulai dari unggah firmware, pemeriksaan versi, pengunduhan file biner, verifikasi keamanan, penulisan ke flash, hingga aktivasi firmware dapat berjalan otomatis tanpa mengganggu tugas lain berkat dukungan FreeRTOS. Hasil pengujian menunjukkan bahwa penambahan fitur keamanan memiliki dampak langsung terhadap sumber daya perangkat, khususnya pada penggunaan CPU dan durasi proses pembaruan.

Plain OTA menjadi metode tercepat dengan penggunaan RAM terendah, namun tidak memberikan perlindungan apa pun sehingga berisiko terhadap manipulasi dan pencurian firmware. SHA-256 memberikan proteksi integritas dengan tambahan beban komputasi yang relatif kecil, sehingga cocok digunakan apabila hanya diperlukan validasi keaslian data tanpa kerahasiaan. HMAC-SHA256 menghasilkan workload CPU tertinggi karena proses hashing ganda, namun menawarkan autentikasi firmware yang sangat kuat dan efektif untuk mencegah instalasi firmware palsu. Sementara itu, AES mampu menjaga kerahasiaan firmware dengan durasi update yang sedikit lebih lama, tetapi tetap efisien berkat adanya hardware accelerator pada ESP32 yang dapat menekan beban CPU dan RAM.

Secara keseluruhan, hasil penelitian membuktikan adanya trade-off antara tingkat keamanan dan efisiensi sumber daya perangkat, sehingga pemilihan algoritma OTA harus disesuaikan dengan kebutuhan proteksi firmware serta keterbatasan komputasi pada perangkat IoT yang digunakan.

5.2 Saran

Berdasarkan hasil penelitian ini, beberapa saran dapat diberikan untuk pengembangan maupun implementasi lebih lanjut:

A. Saran implementatif sistem

1. Sistem OTA sebaiknya dioperasikan pada jaringan yang terlindungi (misalnya WPA2-Enterprise atau VPN) untuk mengurangi risiko penyadapan selama pengunduhan firmware.
2. Firmware server penyedia update perlu diamankan menggunakan HTTPS dan sertifikat TLS agar mencegah serangan *man-in-the-middle* saat dilakukan distribusi OTA.
3. Penggunaan HMAC-SHA256 atau AES perlu disesuaikan dengan kemampuan perangkat, terutama pada deployment masif dengan kapasitas baterai terbatas.

B. Saran penyempurnaan sistem

1. Penelitian selanjutnya dapat menambahkan metode keamanan berbasis tanda tangan digital (*digital signature*) seperti ECDSA untuk mengurangi beban CPU namun tetap memberikan autentikasi kuat.
2. Sistem dapat dikembangkan menggunakan cloud OTA management sehingga mendukung pembaruan pada skala lebih besar di luar jaringan lokal.
3. Penambahan *rollback protection* perlu diterapkan untuk mencegah firmware kembali ke versi lama yang rentan eksploitasi.

C. Saran penelitian lanjutan

1. Perlu dilakukan kajian tambahan mengenai pengaruh OTA terhadap konsumsi daya baterai pada perangkat IoT yang bekerja secara *low-power* atau berbasis energi mandiri (*energy harvesting*).
2. Perbandingan lebih luas dapat dilakukan dengan algoritma kriptografi lain (misalnya Blake2s, ChaCha20-Poly1305) untuk menilai efisiensi terhadap perangkat IoT kelas mikro.
3. Pengujian dapat diperluas pada model ESP32 berbeda serta mikrokontroler lain seperti STM32, RP2040, atau ESP8266 untuk menghasilkan rekomendasi pemilihan platform IoT yang lebih komprehensif.

Daftar Pustaka

[1] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017, doi: 10.1109/JIOT.2017.2683200.

[2] A. Choudhary, "Internet of Things: a comprehensive overview, architectures, applications, simulation tools, challenges and future directions," *Discov. Internet Things*, vol. 4, no. 1, p. 31, Dec. 2024, doi: 10.1007/s43926-024-00084-3.

[3] M. N. Mohammed, S. F. Desyansah, S. Al-Zubaidi, and E. Yusuf, "An internet of things-based smart homes and healthcare monitoring and management system: Review," *J. Phys. Conf. Ser.*, vol. 1450, no. 1, p. 012079, Feb. 2020, doi: 10.1088/1742-6596/1450/1/012079.

[4] J. Q, *Performance Evaluation of Cryptographic Algorithms on ESP32 with Cryptographic Hardware Acceleration Feature*. Sweden, 2022.

[5] N. S. Albalawi, A. G. Alanazi, S. Alshammari, F. Alhamazani, and A. A. Alshammari, "A review of parallel processing in resource-constrained Internet of Things (IoT) devices," *Int. J. Adv. Appl. Sci.*, vol. 12, no. 9, pp. 21–35, Aug. 2025, doi: 10.21833/ijaas.2025.09.003.

[6] I. G. N. D. Paramartha, I. N. H. Kurniawan, G. B. Subiksa, and A. S. Kartika, "Arsitektur Internet of Things (IoT) Berskala Industri dengan fitur Over The Air Update," *TIERS Inf. Technol. J.*, vol. 2, no. 2, pp. 31–36, Dec. 2021, doi: 10.38043/tiers.v2i2.3311.

[7] Y. Safitri, Dahlan, and Maulan Muhammad Jogo Samodro, "Strategi dan Efektivitas Deep Learning untuk Mitigasi Ancaman Keamanan Jaringan di Era IoT," *Sci. J. Comput. Sci. Informatics*, vol. 2, no. 1, pp. 15–22, Jan. 2025, doi: 10.34304/scientific.v1i2.338.

[8] T. Bakhshi, B. Ghita, and I. Kuzminykh, "A Review of IoT Firmware Vulnerabilities and Auditing Techniques," *Sensors*, vol. 24, no. 2, p. 708, Jan. 2024, doi: 10.3390/s24020708.

[9] M. Bettayeb, Q. Nasir, and M. A. Talib, "Firmware Update Attacks and Security for IoT Devices," in *Proceedings of the ArabWIC 6th Annual International Conference Research Track*, New York, NY, USA: ACM, Mar. 2019, pp. 1–6. doi:

10.1145/3333165.3333169.

[10] H. Huang and J. Wang, “Systematic Literature Review on Security Mechanisms for IoT Device Firmware Update,” 2025. doi: 10.2139/ssrn.5359954.

[11] I. Mavromatis *et al.*, “Reliable IoT Firmware Updates: A Large-scale Mesh Network Performance Investigation,” Mar. 2022, doi: 10.1109/WCNC51071.2022.9771708.

[12] N. M. Kumar and P. K. Mallick, “The Internet of Things: Insights into the building blocks, component interactions, and architecture layers,” *Procedia Comput. Sci.*, vol. 132, pp. 109–117, 2018, doi: 10.1016/j.procs.2018.05.170.

[13] B. Pearson *et al.*, “On Misconception of Hardware and Cost in IoT Security and Privacy,” in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, IEEE, May 2019, pp. 1–7. doi: 10.1109/ICC.2019.8761062.

[14] J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman, and E. De Poorter, “Over-the-Air Software Updates in the Internet of Things: An Overview of Key Principles,” *IEEE Commun. Mag.*, vol. 58, no. 2, pp. 35–41, Feb. 2020, doi: 10.1109/MCOM.001.1900125.

[15] J. Furtak, “The Cryptographic Key Distribution System for IoT Systems in the MQTT Environment,” *Sensors*, vol. 23, no. 11, p. 5102, May 2023, doi: 10.3390/s23115102.

[16] J. Arm, O. Baštán, O. Mihálik, and Z. Bradáč, “Measuring the Performance of FreeRTOS on ESP32 Multi-Core,” *IFAC-PapersOnLine*, vol. 55, no. 4, pp. 292–297, 2022, doi: 10.1016/j.ifacol.2022.06.048.

[17] C.-Y. Park, S.-J. Lee, and I.-G. Lee, “Secure and Lightweight Firmware Over-the-Air Update Mechanism for Internet of Things,” *Electronics*, vol. 14, no. 8, p. 1583, Apr. 2025, doi: 10.3390/electronics14081583.

[18] M. J. B. de Sousa, L. F. G. Gonzalez, E. M. Ferdinando, and J. F. Borin, “Over-the-air firmware update for IoT devices on the wild,” *Internet of Things*, vol. 19, p. 100578, Aug. 2022, doi: 10.1016/j.iot.2022.100578.

[19] S. El Jaouhari, “Toward a Secure Firmware OTA Updates for constrained IoT devices,” in *2022 IEEE International Smart Cities Conference (ISC2)*, IEEE, Sep. 2022, pp. 1–6. doi: 10.1109/ISC255366.2022.9922087.

[20] A. De Simone, G. Turvani, and F. Riente, “Incremental Firmware Update Over-

the-Air for Low-Power IoT Devices over LoRaWAN," Jul. 2025, doi: 10.1016/j.iot.2025.101772.

[21] S. P. Jadhav, "Towards Light Weight CryptographySchemes for Resource ConstraintDevices in IoT," *J. Mob. Multimed.*, vol. 15, no. 1, pp. 91–110, 2020, doi: 10.13052/jmm1550-4646.15125.

[22] S. A. Ansari and S. Ali, "A systematic review of lightweight cryptographic schemes for security and privacy in IoT," *Discov. Comput.*, vol. 28, no. 1, p. 266, Nov. 2025, doi: 10.1007/s10791-025-09755-3.

[23] L. Zhang and L. Wang, "A hybrid encryption approach for efficient and secure data transmission in IoT devices," *J. Eng. Appl. Sci.*, vol. 71, no. 1, p. 138, Dec. 2024, doi: 10.1186/s44147-024-00459-x.

[24] S. Łeska and J. Furtak, "Procedures for Building a Secure Environment in IoT Networks Using the LoRa Interface," *Sensors*, vol. 25, no. 13, p. 3881, Jun. 2025, doi: 10.3390/s25133881.

[25] Z. Panjaitan, "Modifikasi SHA-256 dengan Algoritma Hill Cipher untuk Pengamanan Fungsi Hash dari Upaya Decode Hash," *SAINTIKOM*, vol. 19, 2020.

[26] A. Chairil, "Implementasi Algoritma OTP dan HMAC untuk Two Factor Authentication Sistem Login Relawan Pemilu," *Teknika*, vol. 19, 2025.

[27] O. D. DWIPA, "KOMBINASI PERLINDUNGAN DATA MATERIAL SAP MENGGUNAKAN ALGORITMA AES 128 BIT DAN REVERSE CIPHER DI PT INDONESIA COMNET PLUS," 2025.

[28] L. Mustika, "Implementasi Algoritma AES Untuk Pengamanan Login Dan Data Customer Pada E-Commerce Berbasis Web," *JURIKOM (Jurnal Ris. Komputer)*, vol. 7, no. 1, p. 148, Feb. 2020, doi: 10.30865/jurikom.v7i1.1943.

[29] Faris Munir, Basuki Rahmat, and Fawwaz Ali Akbar, "Rancang Bangun Sistem Pembaruan Firmware Over-the-Air (OTA) untuk Perangkat ESP32 Berbasis Layanan Cloud," *J. Ilm. Tek. Inform. dan Komun.*, vol. 5, no. 2, pp. 554–565, Jun. 2025, doi: 10.55606/juitik.v5i2.1180.

[30] I. K. Yusri, K. Kasim, and A. M. Akbar, "Penerapan Hypertext Transfer Protocol Web Server untuk Over-The-Air Auto Update Firmware pada Perangkat IoT," *Elektron J. Ilm.*, pp. 67–71, Dec. 2022, doi: 10.30630/eji.14.2.298.

[31] L. Hakim, W. A. Kusuma, M. Faiqurahman, and Supriyanto, "Over The Air Update

Firmware pada Perangkat IoT Dengan Protokol MQTT," *J. Sist. dan Inform.*, vol. 14, no. 2, pp. 99–105, Aug. 2020, doi: 10.30864/jsi.v14i2.244.

[32] "Security Threats and Concerns, Firmware Vulnerability Analysis in Industrial Internet of Things," *Int. J. Emerg. Trends Eng. Res.*, vol. 8, no. 9, pp. 5255–5258, Sep. 2020, doi: 10.30534/ijeter/2020/59892020.

[33] W. M. A. B. Wijesundara, J.-S. Lee, D. Tith, E. Aloupogianni, H. Suzuki, and T. Obi, "Security-enhanced firmware management scheme for smart home IoT devices using distributed ledger technologies," *Int. J. Inf. Secur.*, vol. 23, no. 3, pp. 1927–1937, Jun. 2024, doi: 10.1007/s10207-024-00827-x.

[34] F. Mahfoudhi, A. K. Sultania, and J. Famaey, "Over-the-Air Firmware Updates for Constrained NB-IoT Devices," *Sensors*, vol. 22, no. 19, p. 7572, Oct. 2022, doi: 10.3390/s22197572.

[35] K. Arakadakis, P. Charalampidis, A. Makrogiannakis, and A. Fragkiadakis, "Firmware over-the-air programming techniques for IoT networks -- A survey," Sep. 2020, [Online]. Available: http://arxiv.org/abs/2009.02260

[36] E. Climent, M. Hecht, and K. Rurack, "Loading and Release of Charged and Neutral Fluorescent Dyes into and from Mesoporous Materials: A Key Role for Sensing Applications," *Micromachines*, vol. 12, no. 3, p. 249, Feb. 2021, doi: 10.3390/mi12030249.

[37] J. Beningo, "How to Perform Over-the-Air (OTA) Updates Using the ESP32 Microcontroller and its ESP-IDF," 2021.

[38] A. Noerifanza, "Analisa Kelayakan Modul Esp32 Sebagai Kamera untuk Pengenalan Objek Sehari-hari," *J. Comput. Electron. Telecommun.*, vol. 3, no. 2, Dec. 2022, doi: 10.52435/complete.v3i2.263.

[39] N. Rahman, "Analisis Perbandingan Kinerja Sensor Suhu DS18B20, Sensor Suhu LM35 , dan Sensor PT 100 untuk Sistem Pengukuran Kualitas Air dengan Metode Kalibrasi Eutramet CG-13," 2023.

[40] A. E. Prasetiyanto and C. P. Hadisusila, "Aplikasi Arduino dalam Teknik I/O untuk Mengintegrasikan dan Mengendalikan Perangkat Elektronik," *Nusant. Eng.*, vol. 6, no. 2, pp. 96–102, Oct. 2023, doi: 10.29407/noe.v6i2.21308.

[41] F. Sinlae, E. Irwanda, Z. Maulana, and V. Eka Syahputra, "Penggunaan Framework Laravel dalam Membangun Aplikasi Website Berbasis PHP," *J. Siber Multi*

Disiplin, vol. 2, no. 2, pp. 119–132, Jul. 2024, doi: 10.38035/jsmd.v2i2.186.

[42] et al Dyah Yuliana Abidah, “Pengaruh Platform Visual Studio Code Terhadap Hasil Belajar Siswa pada Mata pelajaran Pemrograman Dasar Kelas X Jurusan Teknik Komputer dan Jaringan SMKN 3 Malang,” vol. 9, 2025.

[43] A. Briliyanto, “PENGAPLIKASIAN TEKNIK OVER THE AIR (OTA) UNTUK MELAKUKAN PEMBARUAN KODE PROGRAM PADA SMART BREAKER ESP32 DENGAN PROTOKOL HTTP,” vol. 2, 2024.

[44] L. M. Broell, C. Hanshans, and D. Kimmerle, “IoT on an ESP32: Optimization Methods Regarding Battery Life and Write Speed to an SD-Card,” in *Edge Computing - Technology, Management and Integration*, IntechOpen, 2023. doi: 10.5772/intechopen.110752.

[45] M. Z. Abdillah, Farid Baskoro, Muhammad Syariffudien Zuhrie, and Nur Kholis, “Pengembangan Perangkat Pembelajaran Mikrokontroler Esp32 Berbasis Internet of Things & Bluetooth di SMKN 2 Surabaya,” *JE-Unisla*, vol. 9, no. 2, pp. 110–123, Sep. 2024, doi: 10.30736/je-unisla.v9i2.1234.

[46] et al Ananda Olin, “UPDATE TIME DELAY PADA PERALIHAN FASE PENYALAAN LAMPU LALU LINTAS MENGGUNAKAN TEKNIK OVER THE AIR (OTA),” 2021.

[47] B. et al Pearson, “On Misconception of Hardware and Cost in IoT Security and Privacy,” 2019.

LAMPIRAN

1. AES 128

File Firmware Suhu

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Update.h>
#include <ArduinoJson.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include "esp_system.h"

// ===== FreeRTOS / idle hook (tanpa runtime stats) =====
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_freertos_hooks.h"

// ====== Tambahan untuk dekripsi AES-CTR ======
#include "mbedtls/aes.h"

// ====== DS18B20 (OneWire) ======
#include <OneWire.h>
#include <DallasTemperature.h>
#define DS18B20_PIN 33   // DATA DS18B20 di GPIO33 (dengan pull-up 4.7k
ke 3.3V)
OneWire oneWire(DS18B20_PIN);
DallasTemperature ds18(&oneWire);

// State pembacaan DS18B20 (non-blocking)
float waterTempC = NAN;
const uint32_t DS_PERIOD_MS = 1000;  // kirim / perbarui setiap 10 detik
const uint32_t DS_CONV_MS   = 750;    // 12-bit conv. time
unsigned long dsLastTickMs = 0;       // penjadwalan periodik
unsigned long dsConvStartMs = 0;      // waktu mulai konversi
bool dsConvPending = false;           // true setelah
requestTemperatures()

// ====== KONFIGURASI ======
const char* WIFI_SSID = "POCO F4";
const char* WIFI_PASS = "11111111";
const char* CHECK_URL = "http://10.248.38.135/qwen/check_update.php";
// server OTA
const char* HOSTNAME  = "esp32-dashboard"; // mDNS: http://esp32-
dashboard.local

// Versi firmware saat ini
#define CURRENT_VERSION "v1.0.0"

// ===== Kunci AES-128 untuk dekripsi OTA terenkripsi (GANTI ke kunci
milikmu, 16 byte persis) =====
static const uint8_t K_ENC[16] = {

0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0xAA,0xBB,0xCC,0xDD,0x
EE,0xFF
};

// Web server
WebServer server(80);
```

```
// ===== Status update OTA =====
String lastStatus = "Idle";
unsigned long lastOtaTime = 0;   // durasi update (ms)
float lastOtaSpeed = 0.0f;       // KB/s
long minHeapDuringOTA = 0;       // heap min selama OTA (bytes)
long maxHeapDuringOTA = 0;       // heap max selama OTA (bytes)

// ----- OTA CPU stats (avg / max) -----
static volatile bool otaMeasuring = false;
static float  otaCpuSumCore[2] = {0,0};
static float  otaCpuSumTotal   = 0;
static uint32_t otaCpuSamples  = 0;

static float lastOtaCpuAvgCore0  = 0, lastOtaCpuAvgCore1  = 0,
lastOtaCpuAvgTotal  = 0;
static float lastOtaCpuMaxCore0  = 0, lastOtaCpuMaxCore1  = 0,
lastOtaCpuMaxTotal  = 0;

// ====== (BARU) OTA RAM avg ======
static uint64_t otaHeapSum = 0;        // penjumlahan sampel heap
(bytes)
static uint32_t otaHeapSamples = 0;    // jumlah sampel
static float    lastOtaHeapAvg = 0;    // hasil rata-rata (bytes)

// ---------- CPU (idle-hook estimator, per-core & total) ----------
static volatile uint64_t idleCnt[portNUM_PROCESSORS] = {0, 0};
static uint64_t idleBaseline[portNUM_PROCESSORS]     = {1, 1};
static float cpuCore[portNUM_PROCESSORS]             = {0.0f, 0.0f};
static float cpuTotal                                 = 0.0f;
TaskHandle_t cpuMonTaskHandle                         = nullptr;

bool idleHook0() { idleCnt[0]++; return true; }
bool idleHook1() { idleCnt[1]++; return true; }

// ---------- Util Format ----------
String formatBytes(long bytes) {
  if (bytes < 1024) return String(bytes) + " B";
  else if (bytes < (1024 * 1024)) return String(bytes / 1024.0, 2) + "
KB";
  else return String(bytes / 1024.0 / 1024.0, 2) + " MB";
}

// Helpers OTA CPU/RAM stats
static inline void otaStatsStart() {
  otaMeasuring = true;
  otaCpuSumCore[0] = otaCpuSumCore[1] = 0;
  otaCpuSumTotal   = 0;
  otaCpuSamples    = 0;
  lastOtaCpuMaxCore0 = lastOtaCpuMaxCore1 = lastOtaCpuMaxTotal = 0;

  // RAM min/max + avg init
  minHeapDuringOTA = ESP.getFreeHeap();
  maxHeapDuringOTA = minHeapDuringOTA;
  otaHeapSum = 0;
  otaHeapSamples = 0;
  lastOtaHeapAvg = 0;
}
static inline void otaStatsFinish() {
```

```
  otaMeasuring = false;
  if (otaCpuSamples > 0) {
    lastOtaCpuAvgCore0 = otaCpuSumCore[0] / otaCpuSamples;
    lastOtaCpuAvgCore1 = otaCpuSumCore[1] / otaCpuSamples;
    lastOtaCpuAvgTotal = otaCpuSumTotal   / otaCpuSamples;
  } else {
    lastOtaCpuAvgCore0 = lastOtaCpuAvgCore1 = lastOtaCpuAvgTotal = 0;
  }
  // hitung RAM avg
  if (otaHeapSamples > 0) lastOtaHeapAvg = (float)otaHeapSum /
(float)otaHeapSamples;
  else lastOtaHeapAvg = 0;
}

// ---------- OTA (PLAIN) ----------
bool doOTAFromURL(const String& binURL) {
  lastStatus = "Mengunduh firmware...";
  otaStatsStart();

  HTTPClient http;
  http.setTimeout(60000);

  if (!http.begin(binURL)) { lastStatus = "http.begin gagal";
otaStatsFinish(); return false; }
  int code = http.GET();
  if (code != HTTP_CODE_OK) { lastStatus = "HTTP error " + String(code);
http.end(); otaStatsFinish(); return false; }

  int contentLength = http.getSize();
  if (!Update.begin(contentLength > 0 ? contentLength :
UPDATE_SIZE_UNKNOWN)) {
    lastStatus = "Update.begin gagal"; http.end(); otaStatsFinish();
return false;
  }

  WiFiClient* stream = http.getStreamPtr();
  uint8_t buf[2048];
  size_t totalWritten = 0;
  unsigned long t0 = millis();

  size_t mark = 0;
  while (true) {
    int n = stream->readBytes((char*)buf, sizeof(buf));
    if (n < 0) { lastStatus = "read error"; Update.abort(); http.end();
otaStatsFinish(); return false; }
    if (n == 0) { if (!stream->available()) break; delay(1); continue; }
    if (Update.write(buf, n) != (size_t)n) {
      lastStatus = "Update.write error"; Update.abort(); http.end();
otaStatsFinish(); return false;
    }
    totalWritten += n;

    // RAM min/max
    long heapNow = ESP.getFreeHeap();
    if (heapNow < minHeapDuringOTA) minHeapDuringOTA = heapNow;
    if (heapNow > maxHeapDuringOTA) maxHeapDuringOTA = heapNow;

    if (totalWritten - mark >= 100 * 1024) { mark = totalWritten; }
  }
```

```
  http.end();

  if (!Update.end() || !Update.isFinished()) {
    lastStatus = "Update gagal, err=" + String(Update.getError());
    otaStatsFinish();
    return false;
  }

  unsigned long dt = millis() - t0;
  lastOtaTime  = dt;
  lastOtaSpeed = (totalWritten / 1024.0) / (dt / 1000.0);
  lastStatus   = "Update sukses, silakan klik Reboot";

  otaStatsFinish();

  Serial.printf("[OTA] Total: %u bytes, Durasi: %.2f s, Kecepatan: %.2f
KB/s\n",
                (unsigned)totalWritten, dt / 1000.0, lastOtaSpeed);
  Serial.printf("[RAM] OTA -> Min: %.2f KB, Avg: %.2f KB, Max: %.2f
KB\n",
                minHeapDuringOTA / 1024.0, lastOtaHeapAvg / 1024.0,
maxHeapDuringOTA / 1024.0);
  Serial.printf("[CPU] OTA avg C0=%.1f C1=%.1f Tot=%.1f | max C0=%.1f
C1=%.1f Tot=%.1f | sample=%u\n",
                lastOtaCpuAvgCore0, lastOtaCpuAvgCore1,
lastOtaCpuAvgTotal,
                lastOtaCpuMaxCore0, lastOtaCpuMaxCore1,
lastOtaCpuMaxTotal,
                otaCpuSamples);
  return true;
}

// ---------- Util: parse hex 32 char → 16 byte IV ----------
static bool parseHex16(const String& hex, uint8_t out16[16]) {
  if (hex.length() != 32) return false;
  auto nybble = [](char c)->int{
    if (c>='0' && c<='9') return c-'0';
    if (c>='a' && c<='f') return c-'a'+10;
    if (c>='A' && c<='F') return c-'A'+10;
    return -1;
  };
  for (int i=0;i<16;i++){
    int hi = nybble(hex[2*i]);
    int lo = nybble(hex[2*i+1]);
    if (hi<0 || lo<0) return false;
    out16[i] = (uint8_t)((hi<<4)|lo);
  }
  return true;
}

// ---------- OTA (ENCRYPTED, AES-128-CTR STREAM) ----------
bool doOTAFromURLEncrypted(const String& binURL, const String& ivHex,
uint32_t expectedSize) {
  lastStatus = "Mengunduh (encrypted)...";
  otaStatsStart();

  // Parse IV hex → 16 byte
  uint8_t iv[16];
  if (!parseHex16(ivHex, iv)) {
```

```
    lastStatus = "IV hex invalid";
    otaStatsFinish();
    return false;
  }

  HTTPClient http;
  http.setTimeout(60000);
  if (!http.begin(binURL)) { lastStatus = "http.begin gagal";
otaStatsFinish(); return false; }
  int code = http.GET();
  if (code != HTTP_CODE_OK) { lastStatus = "HTTP error " + String(code);
http.end(); otaStatsFinish(); return false; }

  int contentLength = http.getSize();
  if (expectedSize > 0 && contentLength > 0 && (uint32_t)contentLength
!= expectedSize) {
    lastStatus = "Size mismatch";
    http.end(); otaStatsFinish(); return false;
  }
  if (!Update.begin(contentLength > 0 ? contentLength :
UPDATE_SIZE_UNKNOWN)) {
    lastStatus = "Update.begin gagal"; http.end(); otaStatsFinish();
return false;
  }

  WiFiClient* stream = http.getStreamPtr();

  // Siapkan AES-CTR 128
  mbedtls_aes_context aes;
  mbedtls_aes_init(&aes);
  mbedtls_aes_setkey_enc(&aes, K_ENC, 128);

  unsigned char nonce_counter[16];
  memcpy(nonce_counter, iv, 16);
  unsigned char stream_block[16] = {0};
  size_t nc_off = 0;

  uint8_t inbuf[2048];
  uint8_t outbuf[2048];

  size_t totalWritten = 0;
  unsigned long t0 = millis();

  while (true) {
    int n = stream->readBytes((char*)inbuf, sizeof(inbuf));
    if (n < 0) { lastStatus = "read error"; Update.abort(); http.end();
mbedtls_aes_free(&aes); otaStatsFinish(); return false; }
    if (n == 0) { if (!stream->available()) break; delay(1); continue; }

    // Dekripsi CTR: ciphertext -> plaintext
    mbedtls_aes_crypt_ctr(&aes, n, &nc_off, nonce_counter, stream_block,
inbuf, outbuf);

    if (Update.write(outbuf, n) != (size_t)n) {
      lastStatus = "Update.write error";
      Update.abort(); http.end(); mbedtls_aes_free(&aes);
otaStatsFinish(); return false;
    }
    totalWritten += n;
```

```
    // RAM min/max (telemetri)
    long heapNow = ESP.getFreeHeap();
    if (heapNow < minHeapDuringOTA) minHeapDuringOTA = heapNow;
    if (heapNow > maxHeapDuringOTA) maxHeapDuringOTA = heapNow;
  }

  http.end();
  mbedtls_aes_free(&aes);

  if (!Update.end() || !Update.isFinished()) {
    lastStatus = "Update gagal, err=" + String(Update.getError());
    otaStatsFinish();
    return false;
  }

  unsigned long dt = millis() - t0;
  lastOtaTime  = dt;
  lastOtaSpeed = (totalWritten / 1024.0) / (dt / 1000.0);
  lastStatus   = "Update sukses (encrypted), klik Reboot";

  otaStatsFinish();

  Serial.printf("[OTA-ENC] Total: %u bytes, Durasi: %.2f s, Kecepatan:
%.2f KB/s\n",
                (unsigned)totalWritten, dt/1000.0, lastOtaSpeed);
  Serial.printf("[RAM] OTA -> Min: %.2f KB, Avg: %.2f KB, Max: %.2f
KB\n",
                minHeapDuringOTA/1024.0, lastOtaHeapAvg/1024.0,
maxHeapDuringOTA/1024.0);
  Serial.printf("[CPU] OTA avg C0=%.1f C1=%.1f Tot=%.1f | max C0=%.1f
C1=%.1f Tot=%.1f | sample=%u\n",
                lastOtaCpuAvgCore0, lastOtaCpuAvgCore1,
lastOtaCpuAvgTotal,
                lastOtaCpuMaxCore0, lastOtaCpuMaxCore1,
lastOtaCpuMaxTotal,
                otaCpuSamples);
  return true;
}

// ---------- CHECK MANIFEST & DECIDE (ENC/PLAIN) ----------
void checkAndUpdate() {
  lastStatus = "Memeriksa manifest...";
  HTTPClient http;
  if (!http.begin(CHECK_URL)) { lastStatus = "Manifest gagal dibuka";
return; }
  int code = http.GET();
  if (code != HTTP_CODE_OK) { lastStatus = "Manifest HTTP " +
String(code); http.end(); return; }

  String body = http.getString();
  http.end();

  // Buffer JSON diperbesar sedikit untuk field tambahan
  StaticJsonDocument<512> doc;
  if (deserializeJson(doc, body)) { lastStatus = "JSON parse error";
return; }

  String latest = doc["version"] | "";
```

```
  String fwurl  = doc["url"]     | "";
  bool   enc    = doc["enc"]     | false;
  String ivHex  = doc["ctr_iv"]  | "";
  uint32_t size = doc["size"]    | 0;

  if (latest == "" || fwurl == "") { lastStatus = "Manifest tidak
lengkap"; return; }
  if (latest == CURRENT_VERSION)   { lastStatus = "Sudah versi terbaru";
return; }

  if (enc) {
    lastStatus = "Versi baru " + latest + " (encrypted) tersedia,
update...";
    doOTAFromURLEncrypted(fwurl, ivHex, size);
  } else {
    lastStatus = "Versi baru " + latest + " (plain) tersedia,
update...";
    doOTAFromURL(fwurl);
  }
}

// ---------- WEB (Realtime via AJAX /metrics) ----------
void handleRoot() {
  String html;
  html.reserve(9000);
  html  = "<!doctype html><html><head><meta charset='utf-8'><title>ESP32
OTA + Resource</title>";
  html += "<meta name='viewport' content='width=device-width,initial-
scale=1'>";
  html += "<style>"
          "body{font-family:system-ui,Arial;margin:16px}"
          ".k{color:#555}.row{margin:6px 0}"
          ".card{border:1px solid #ddd;border-
radius:12px;padding:12px;margin:12px 0;box-shadow:0 1px 3px
rgba(0,0,0,.05)}"
          ".h{font-weight:700}"
          ".bar{height:10px;background:#eee;border-
radius:8px;overflow:hidden}"
          ".fill{height:100%;width:0%}"
          ".mono{font-family:ui-monospace,Consolas,monospace}"
          "</style>";
  html += "</head><body>";
  html += "<h2>ESP32 OTA + Monitoring (Realtime)</h2>";

  html += "<div class='card'><div class='row'><span class='k'>Status:
</span><span id='status'>-</span></div>";
  html += "<div class='row'><span class='k'>Firmware: </span><b
id='fw'>-</b></div>";
  html += "<div class='row'><span class='k'>IP: </span><b>" +
WiFi.localIP().toString() + "</b></div>";
  html += "<div class='row'><span class='k'>Uptime: </span><span
id='uptime'>-</span></div>";
  html += "<div class='row'><span class='k'>Suhu Air (DS18B20):
</span><b id='ds18'>-</b> °C</div>";
  html += "<div class='row'><a href='/check'>Cek Update</a> | <a
href='/reboot'>Reboot</a></div></div>";

  html += "<div class='card'><div class='h'>CPU</div>";
```

```
  html += "<div class='row'>Core0: <b id='c0'>0%</b><div
class='bar'><div id='c0b' class='fill'></div></div></div>";
  html += "<div class='row'>Core1: <b id='c1'>0%</b><div
class='bar'><div id='c1b' class='fill'></div></div></div>";
  html += "<div class='row'>Total: <b id='ct'>0%</b><div
class='bar'><div id='ctb' class='fill'></div></div></div></div>";

  html += "<div class='card'><div class='h'>Memori</div>";
  html += "<div class='row'>Heap: <span id='heap'>-</span> (<b
id='heapp'>-</b>)<div class='bar'><div id='heapb'
class='fill'></div></div></div>";
  html += "<div class='row'>Flash: <span id='flash'>-</span> (<b
id='flashp'>-</b>)<div class='bar'><div id='flashb'
class='fill'></div></div></div></div>";

  html += "<div class='card'><div class='h'>Ringkasan OTA
terakhir</div>";
  html += "<div class='row mono'>Durasi: <b id='otat'>-</b> s,
Kecepatan: <b id='otas'>-</b> KB/s</div>";
  html += "<div class='row mono'>CPU avg (C0/C1/Tot): <b id='otaavg'>-
</b></div>";
  html += "<div class='row mono'>CPU max (C0/C1/Tot): <b id='otamax'>-
</b> | Sampel: <b id='otasamp'>-</b></div>";
  html += "<div class='row mono'>RAM selama OTA: Min <b id='otarammin'>-
</b>, Avg <b id='otaramavg'>-</b>, Max <b id='otarammax'>-</b></div>";
  html += "</div>";

  html += "<script>";
  html += "function fmtBytes(x){if(x<1024)return x+'
B';if(x<1048576)return (x/1024).toFixed(2)+' KB';return
(x/1048576).toFixed(2)+' MB'};";
  html += "function fmtUp(ms){let
s=Math.floor(ms/1000),m=Math.floor(s/60),h=Math.floor(m/60),d=Math.floor
(h/24);"
          "return (d?d+' hari, ':'')+((h%24)?(h%24)+' jam,
':'')+((m%60)?(m%60)+' menit, ':'')+(s%60)+' detik'};";
  html += "async function tick(){try{const r=await
fetch('/metrics');const j=await r.json();";
  html += "document.getElementById('status').textContent=j.status;";
  html += "document.getElementById('fw').textContent=j.firmware;";
  html +=
"document.getElementById('uptime').textContent=fmtUp(j.uptime_ms);";
  // suhu
  html +=
"if(j.water_temp_c!==null){document.getElementById('ds18').textContent=j
.water_temp_c.toFixed(2);}";

  // CPU
  html += "const
c0=j.cpu_core0||0,c1=j.cpu_core1||0,ct=j.cpu_total||0;";
  html +=
"document.getElementById('c0').textContent=c0.toFixed(1)+'%';";
  html +=
"document.getElementById('c1').textContent=c1.toFixed(1)+'%';";
  html +=
"document.getElementById('ct').textContent=ct.toFixed(1)+'%';";
  html += "document.getElementById('c0b').style.width=c0+'%';";
  html += "document.getElementById('c1b').style.width=c1+'%';";
  html += "document.getElementById('ctb').style.width=ct+'%';";
```

```
  // RAM
  html += "const heapUsed=j.heap_total-j.heap_free;";
  html +=
"document.getElementById('heap').textContent=fmtBytes(heapUsed)+' /
'+fmtBytes(j.heap_total);";
  html +=
"document.getElementById('heapp').textContent=(j.heap_used_percent||0).t
oFixed(2)+'%';";
  html +=
"document.getElementById('heapb').style.width=(j.heap_used_percent||0)+'
%';";
  // Flash
  html +=
"document.getElementById('flash').textContent=fmtBytes(j.flash_sketch)+'
/ '+fmtBytes(j.flash_total)+' (free sketch
'+fmtBytes(j.flash_free_sketch)+')';";
  html +=
"document.getElementById('flashp').textContent=(j.flash_used_percent||0)
.toFixed(2)+'%';";
  html +=
"document.getElementById('flashb').style.width=(j.flash_used_percent||0)
+'%';";
  // OTA summary
  html +=
"if(j.ota_time_ms>0){document.getElementById('otat').textContent=(j.ota_
time_ms/1000).toFixed(2);document.getElementById('otas').textContent=(j.
ota_speed_kbps||0).toFixed(2);} ";
  html +=
"document.getElementById('otaavg').textContent=(j.ota_cpu_avg_core0||0).
toFixed(1)+'% / '+(j.ota_cpu_avg_core1||0).toFixed(1)+'% /
'+(j.ota_cpu_avg_total||0).toFixed(1)+'%';";
  html +=
"document.getElementById('otamax').textContent=(j.ota_cpu_max_core0||0).
toFixed(1)+'% / '+(j.ota_cpu_max_core1||0).toFixed(1)+'% /
'+(j.ota_cpu_max_total||0).toFixed(1)+'%';";
  html +=
"document.getElementById('otasamp').textContent=j.ota_cpu_samples||0;";
  html +=
"document.getElementById('otarammin').textContent=fmtBytes(j.ota_ram_min
||0);";
  html +=
"document.getElementById('otaramavg').textContent=fmtBytes(j.ota_ram_avg
||0);";
  html +=
"document.getElementById('otarammax').textContent=fmtBytes(j.ota_ram_max
||0);";
  html += "}catch(e){}}";
  html += "tick(); setInterval(tick,1000);";
  html += "</script>";

  html += "</body></html>";
  server.send(200, "text/html", html);
}

// Endpoint JSON
void handleMetrics() {
  long freeHeap        = ESP.getFreeHeap();
  long totalHeap       = ESP.getHeapSize();
  long flashSize       = ESP.getFlashChipSize();
```

```
  long sketchSize      = ESP.getSketchSize();
  long freeSketchSpace = ESP.getFreeSketchSpace();

  float heapUsedPct  = totalHeap > 0 ? (float)(totalHeap - freeHeap) /
(float)totalHeap * 100.0f : 0.0f;
  float flashUsedPct = flashSize > 0 ? (float)sketchSize            /
(float)flashSize * 100.0f : 0.0f;

  String j; j.reserve(1100);
  j += "{";
  j += "\"status\":\"" + lastStatus + "\",";
  j += "\"firmware\":\"" CURRENT_VERSION "\",";
  j += "\"uptime_ms\":" + String(millis()) + ",";

  // suhu: null jika belum ada
  if (isnan(waterTempC)) j += "\"water_temp_c\":null,";
  else                   j += "\"water_temp_c\":" + String(waterTempC,
2) + ",";

  // CPU
  j += "\"cpu_core0\":" + String(cpuCore[0],1) + ",";
  j += "\"cpu_core1\":" + String(cpuCore[1],1) + ",";
  j += "\"cpu_total\":" + String(cpuTotal,1) + ",";

  // RAM
  j += "\"heap_total\":" + String(totalHeap) + ",";
  j += "\"heap_free\":"  + String(freeHeap) + ",";
  j += "\"heap_used\":"  + String(totalHeap - freeHeap) + ",";
  j += "\"heap_used_percent\":" + String(heapUsedPct, 2) + ",";

  // Flash
  j += "\"flash_total\":" + String(flashSize) + ",";
  j += "\"flash_sketch\":" + String(sketchSize) + ",";
  j += "\"flash_free_sketch\":" + String(freeSketchSpace) + ",";
  j += "\"flash_used_percent\":" + String(flashUsedPct, 2) + ",";

  // OTA
  j += "\"ota_time_ms\":" + String(lastOtaTime) + ",";
  j += "\"ota_speed_kbps\":" + String(lastOtaSpeed) + ",";

  // OTA CPU stats
  j += "\"ota_cpu_avg_core0\":" + String(lastOtaCpuAvgCore0, 1) + ",";
  j += "\"ota_cpu_avg_core1\":" + String(lastOtaCpuAvgCore1, 1) + ",";
  j += "\"ota_cpu_avg_total\":" + String(lastOtaCpuAvgTotal, 1) + ",";
  j += "\"ota_cpu_max_core0\":" + String(lastOtaCpuMaxCore0, 1) + ",";
  j += "\"ota_cpu_max_core1\":" + String(lastOtaCpuMaxCore1, 1) + ",";
  j += "\"ota_cpu_max_total\":" + String(lastOtaCpuMaxTotal, 1) + ",";
  j += "\"ota_cpu_samples\":"   + String(otaCpuSamples) + ",";

  // OTA RAM min/avg/max (bytes)
  j += "\"ota_ram_min\":" + String(minHeapDuringOTA) + ",";
  j += "\"ota_ram_avg\":" + String((long)lastOtaHeapAvg) + ",";
  j += "\"ota_ram_max\":" + String(maxHeapDuringOTA);

  j += "}";
  server.send(200, "application/json", j);
}

void handleReboot() {
```

```
  server.send(200, "text/plain", "Rebooting...");
  delay(1000);
  esp_restart();
}

// ---------- Task monitor CPU (idle-hook based) ----------
void cpuMonitorTask(void*){
  // Kalibrasi baseline 1 detik awal
  uint64_t s0 = idleCnt[0], s1 = idleCnt[1];
  vTaskDelay(pdMS_TO_TICKS(1000));
  idleBaseline[0] = max<uint64_t>(1, idleCnt[0] - s0);
  idleBaseline[1] = max<uint64_t>(1, idleCnt[1] - s1);
  Serial.printf("[CPU] Baseline idle: C0=%llu, C1=%llu (counts/1s)\n",
                (unsigned long long)idleBaseline[0], (unsigned long
long)idleBaseline[1]);

  for(;;){
    uint64_t c0s = idleCnt[0], c1s = idleCnt[1];
    vTaskDelay(pdMS_TO_TICKS(1000));
    uint64_t d0 = idleCnt[0] - c0s;
    uint64_t d1 = idleCnt[1] - c1s;

    float x0 = 100.0f - (float)d0 * 100.0f / (float)idleBaseline[0];
    float x1 = 100.0f - (float)d1 * 100.0f / (float)idleBaseline[1];

    if (x0 < 0) x0 = 0; if (x0 > 100) x0 = 100;
    if (x1 < 0) x1 = 0; if (x1 > 100) x1 = 100;

    cpuCore[0] = x0;
    cpuCore[1] = x1;
    cpuTotal   = (x0 + x1) * 0.5f;

    if (otaMeasuring) {
      otaCpuSumCore[0] += cpuCore[0];
      otaCpuSumCore[1] += cpuCore[1];
      otaCpuSumTotal   += cpuTotal;
      otaCpuSamples    += 1;

      if (cpuCore[0] > lastOtaCpuMaxCore0) lastOtaCpuMaxCore0 =
cpuCore[0];
      if (cpuCore[1] > lastOtaCpuMaxCore1) lastOtaCpuMaxCore1 =
cpuCore[1];
      if (cpuTotal   > lastOtaCpuMaxTotal) lastOtaCpuMaxTotal =
cpuTotal;

      // ===== (BARU) sampel RAM untuk avg =====
      long heapNow = ESP.getFreeHeap();
      otaHeapSum += heapNow;
      otaHeapSamples++;
    }
  }
}

// ---------- Recalibrate endpoint (opsional) ----------
void handleRecalibrate() {
  server.send(200, "text/plain", "Kalibrasi CPU 1 detik dimulai...");
  uint64_t s0 = idleCnt[0], s1 = idleCnt[1];
  delay(1000);
  idleBaseline[0] = max<uint64_t>(1, idleCnt[0] - s0);
```

```
  idleBaseline[1] = max<uint64_t>(1, idleCnt[1] - s1);
  Serial.printf("[CPU] Baseline baru: C0=%llu, C1=%llu\n",
                (unsigned long long)idleBaseline[0], (unsigned long
long)idleBaseline[1]);
}

void setup() {
  Serial.begin(115200);
  Serial.println("\n=== ESP32 OTA + Resource Dashboard (Realtime) —
IdleHook + DS18B20 @ GPIO33 ===");

  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print(".");
}
  Serial.printf("\nWiFi OK. IP: %s\n",
WiFi.localIP().toString().c_str());
  Serial.printf("Firmware saat ini: %s\n", CURRENT_VERSION);

  if (MDNS.begin(HOSTNAME)) {
    Serial.printf("mDNS aktif: http://%s.local\n", HOSTNAME);
    MDNS.addService("http", "tcp", 80);
  }

  // DS18B20 init (robust, non-blocking)
  delay(300);
  ds18.begin();
  ds18.setResolution(12);
  ds18.setWaitForConversion(false);
  Serial.printf("DS18B20 device count: %d (GPIO %d)\n",
ds18.getDeviceCount(), DS18B20_PIN);
  ds18.requestTemperatures();          // mulai konversi pertama
  dsConvStartMs = millis();
  dsConvPending = true;
  dsLastTickMs  = millis();

  // Idle hooks
  esp_register_freertos_idle_hook_for_cpu(idleHook0, 0);
  esp_register_freertos_idle_hook_for_cpu(idleHook1, 1);

  // Routes
  server.on("/",        handleRoot);
  server.on("/metrics", handleMetrics);
  server.on("/check", [](){ server.send(200, "text/plain", "Memeriksa
update..."); checkAndUpdate(); });
  server.on("/reboot",  handleReboot);
  server.on("/recalibrate", handleRecalibrate);
  server.begin();

  // CPU monitor task (Core 0)
  xTaskCreatePinnedToCore(cpuMonitorTask, "cpuMonitorTask", 4096,
nullptr, 1, &cpuMonTaskHandle, 0);
}

void loop() {
  server.handleClient();

  // ===== DS18B20 non-blocking scheduler =====
  unsigned long now = millis();
```

```
  // Jika sedang menunggu hasil konversi, baca setelah >= DS_CONV_MS
  if (dsConvPending && (now - dsConvStartMs >= DS_CONV_MS)) {
    float t = ds18.getTempCByIndex(0);
    if (t != DEVICE_DISCONNECTED_C && t > -55 && t < 125) {
      waterTempC = t;
    } else {
      Serial.println("DS18B20 read fail / not found");
    }
    dsConvPending = false; // selesai baca
  }

  // Periodik 10 detik: trigger konversi baru
  if (now - dsLastTickMs >= DS_PERIOD_MS) {
    ds18.requestTemperatures();     // mulai konversi berikutnya
    dsConvStartMs = now;
    dsConvPending = true;
    dsLastTickMs  = now;
  }
}
```

2. HMAC SHA-256

File Firmware Suhu

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Update.h>
#include <ArduinoJson.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include "esp_system.h"

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_freertos_hooks.h"

// ===== Tambahan untuk HMAC-SHA256 (digital token) =====
#include "mbedtls/md.h"

// ===== DS18B20 (GPIO33) =====
#include <OneWire.h>
#include <DallasTemperature.h>
#define DS18B20_PIN 33
OneWire oneWire(DS18B20_PIN);
DallasTemperature ds18(&oneWire);
float waterTempC = NAN;
```

```
const uint32_t DS_PERIOD_MS = 1000;
const uint32_t DS_CONV_MS   = 750;
unsigned long dsLastTickMs = 0;
unsigned long dsConvStartMs = 0;
bool dsConvPending = false;

// ===== KONFIG =====
const char* WIFI_SSID = "POCO F4";
const char* WIFI_PASS = "11111111";
const char* CHECK_URL =
"http://10.248.38.135/digitaltoken/check_update.php";
const char* HOSTNAME  = "esp32-dashboard";
#define CURRENT_VERSION "v1.0.1"

// ── Kunci rahasia HMAC (SAMAKAN dengan server $K_MAC) ──
static const char* K_MAC = "kelompokTA09";

// (Opsional) Validasi window waktu ts (butuh NTP agar akurat).
// Set 1 untuk aktif, 0 untuk nonaktif (default nonaktif karena belum
NTP).
#define ENABLE_TS_WINDOW 0
#define TS_WINDOW_SEC    300  // ±5 menit

// ===== WEB =====
WebServer server(80);

// ===== Status OTA =====
String lastStatus = "Idle";
unsigned long lastOtaTime = 0;
float lastOtaSpeed = 0.0f;
long minHeapDuringOTA = 0;
long maxHeapDuringOTA = 0;

// ===== CPU & RAM monitor (idle-hook) =====
static volatile uint64_t idleCnt[portNUM_PROCESSORS] = {0, 0};
static uint64_t idleBaseline[portNUM_PROCESSORS]     = {1, 1};
static float cpuCore[portNUM_PROCESSORS]             = {0.0f, 0.0f};
static float cpuTotal                                = 0.0f;
TaskHandle_t cpuMonTaskHandle                        = nullptr;
bool idleHook0() { idleCnt[0]++; return true; }
```

```
bool idleHook1() { idleCnt[1]++; return true; }

// ===== OTA CPU stats =====
static volatile bool otaMeasuring = false;
static float  otaCpuSumCore[2] = {0,0};
static float  otaCpuSumTotal   = 0;
static uint32_t otaCpuSamples  = 0;
static float lastOtaCpuAvgCore0  = 0, lastOtaCpuAvgCore1 = 0,
lastOtaCpuAvgTotal = 0;
static float lastOtaCpuMaxCore0  = 0, lastOtaCpuMaxCore1 = 0,
lastOtaCpuMaxTotal = 0;

// ===== OTA RAM avg =====
static uint64_t otaHeapSum = 0;
static uint32_t otaHeapSamples = 0;
static float    lastOtaHeapAvg = 0;

String formatBytes(long bytes){
  if (bytes < 1024) return String(bytes) + " B";
  else if (bytes < (1024 * 1024)) return String(bytes / 1024.0, 2) + "
KB";
  else return String(bytes / 1024.0 / 1024.0, 2) + " MB";
}

// ===== Helpers OTA stats =====
static inline void otaStatsStart(){
  otaMeasuring = true;
  otaCpuSumCore[0] = otaCpuSumCore[1] = 0;
  otaCpuSumTotal = 0;
  otaCpuSamples  = 0;
  lastOtaCpuMaxCore0 = lastOtaCpuMaxCore1 = lastOtaCpuMaxTotal = 0;
  minHeapDuringOTA = ESP.getFreeHeap();
  maxHeapDuringOTA = minHeapDuringOTA;
  otaHeapSum = 0;
  otaHeapSamples = 0;
  lastOtaHeapAvg = 0;
}
static inline void otaStatsFinish(){
  otaMeasuring = false;
  if (otaCpuSamples > 0) {
```

```
    lastOtaCpuAvgCore0 = otaCpuSumCore[0] / otaCpuSamples;
    lastOtaCpuAvgCore1 = otaCpuSumCore[1] / otaCpuSamples;
    lastOtaCpuAvgTotal = otaCpuSumTotal   / otaCpuSamples;
  } else {
    lastOtaCpuAvgCore0 = lastOtaCpuAvgCore1 = lastOtaCpuAvgTotal = 0;
  }
  if (otaHeapSamples > 0) lastOtaHeapAvg = (float)otaHeapSum /
(float)otaHeapSamples;
  else lastOtaHeapAvg = 0;
}

// ===== HMAC util =====
String hmac_sha256_hex(const String& msg, const char* key) {
  uint8_t out[32];
  mbedtls_md_context_t ctx;
  const mbedtls_md_info_t* info =
mbedtls_md_info_from_type(MBEDTLS_MD_SHA256);
  mbedtls_md_init(&ctx);
  mbedtls_md_setup(&ctx, info, 1 /* HMAC */);
  mbedtls_md_hmac_starts(&ctx, (const unsigned char*)key, strlen(key));
  mbedtls_md_hmac_update(&ctx, (const unsigned char*)msg.c_str(),
msg.length());
  mbedtls_md_hmac_finish(&ctx, out);
  mbedtls_md_free(&ctx);
  char hex[65];
  for (int i=0;i<32;i++) sprintf(hex+2*i, "%02x", out[i]); // hex
lowercase
  hex[64]='\0';
  return String(hex);
}

// ===== OTA =====
bool doOTAFromURL(const String& binURL){
  lastStatus = "Mengunduh firmware...";
  otaStatsStart();

  HTTPClient http; http.setTimeout(60000);
  if (!http.begin(binURL)) { lastStatus = "http.begin gagal";
otaStatsFinish(); return false; }
  int code = http.GET();
```

```
  if (code != HTTP_CODE_OK) { lastStatus = "HTTP error " + String(code);
http.end(); otaStatsFinish(); return false; }

  int contentLength = http.getSize();
  if (!Update.begin(contentLength > 0 ? contentLength :
UPDATE_SIZE_UNKNOWN)) {
    lastStatus = "Update.begin gagal"; http.end(); otaStatsFinish();
return false;
  }

  WiFiClient* stream = http.getStreamPtr();
  uint8_t buf[2048];
  size_t totalWritten = 0;
  unsigned long t0 = millis();

  while (true) {
    int n = stream->readBytes((char*)buf, sizeof(buf));
    if (n < 0) { lastStatus = "read error"; Update.abort(); http.end();
otaStatsFinish(); return false; }
    if (n == 0) { if (!stream->available()) break; delay(1); continue; }
    if (Update.write(buf, n) != (size_t)n) {
      lastStatus = "Update.write error"; Update.abort(); http.end();
otaStatsFinish(); return false;
    }
    totalWritten += n;

    long heapNow = ESP.getFreeHeap();
    if (heapNow < minHeapDuringOTA) minHeapDuringOTA = heapNow;
    if (heapNow > maxHeapDuringOTA) maxHeapDuringOTA = heapNow;
  }
  http.end();

  if (!Update.end() || !Update.isFinished()) {
    lastStatus = "Update gagal, err=" + String(Update.getError());
    otaStatsFinish();
    return false;
  }

  unsigned long dt = millis() - t0;
  lastOtaTime  = dt;
```

```
  lastOtaSpeed = (totalWritten / 1024.0) / (dt / 1000.0);
  lastStatus   = "Update sukses, silakan klik Reboot";

  otaStatsFinish();

  Serial.printf("[OTA] Total: %u bytes, Durasi: %.2f s, Kecepatan: %.2f
KB/s\n",
                (unsigned)totalWritten, dt / 1000.0, lastOtaSpeed);
  Serial.printf("[RAM] OTA -> Min: %.2f KB, Avg: %.2f KB, Max: %.2f
KB\n",
                minHeapDuringOTA/1024.0, lastOtaHeapAvg/1024.0,
maxHeapDuringOTA/1024.0);
  Serial.printf("[CPU] OTA avg C0=%.1f C1=%.1f Tot=%.1f | max C0=%.1f
C1=%.1f Tot=%.1f | sample=%u\n",
                lastOtaCpuAvgCore0, lastOtaCpuAvgCore1,
lastOtaCpuAvgTotal,
                lastOtaCpuMaxCore0, lastOtaCpuMaxCore1,
lastOtaCpuMaxTotal, otaCpuSamples);
  return true;
}

// ===== Manifest + Digital Token (HMAC) =====
void checkAndUpdate(){
  lastStatus = "Memeriksa manifest...";
  HTTPClient http;
  if (!http.begin(CHECK_URL)) { lastStatus = "Manifest gagal dibuka";
return; }
  int code = http.GET();
  if (code != HTTP_CODE_OK) { lastStatus = "Manifest HTTP " +
String(code); http.end(); return; }
  String body = http.getString(); http.end();

  // Buffer lebih besar untuk field token
  StaticJsonDocument<768> doc;
  if (deserializeJson(doc, body)) { lastStatus = "JSON parse error";
return; }

  String ver   = doc["version"] | "";
  String fwurl = doc["url"]     | "";
  uint32_t size= doc["size"]    | 0;
```

```
  String nonce = doc["nonce"]   | "";
  uint32_t ts  = doc["ts"]      | 0;
  String token = doc["token"]   | "";

  if (ver=="" || fwurl=="" || size==0 || nonce=="" || ts==0 ||
token=="") {
    lastStatus = "Manifest tidak lengkap"; return;
  }

  // (Opsional) Validasi window waktu ts (butuh NTP). Default OFF.
#if ENABLE_TS_WINDOW
  {
    // Contoh placeholder (bila belum NTP): gunakan millis() sebagai
pseudo time → umumnya TIDAK akurat.
    // Disarankan aktifkan hanya jika sudah sync NTP (time.h).
    uint32_t nowEpoch = (uint32_t)(millis()/1000);
    int32_t  skew = (int32_t)nowEpoch - (int32_t)ts;
    if (skew < -TS_WINDOW_SEC || skew > TS_WINDOW_SEC) {
      lastStatus = "Token kadaluarsa / jam tidak sinkron";
      return;
    }
  }
#endif

  // Anti-rollback sederhana
  if (ver == CURRENT_VERSION) { lastStatus = "Sudah versi terbaru";
return; }

  // Susun pesan persis seperti di server (URUTAN & FORMAT FIX)
  // ver=<versi>|size=<size>|url=<url>|nonce=<nonce>|ts=<ts>
  String msg = "ver=" + ver
             + "|size=" + String(size)
             + "|url=" + fwurl
             + "|nonce=" + nonce
             + "|ts=" + String(ts);

  // Hitung HMAC lokal (hex lowercase) dan bandingkan
  String tloc = hmac_sha256_hex(msg, K_MAC);
  if (tloc != token) {
    lastStatus = "HMAC mismatch"; return;
```

```
  }

  // Token valid → lanjut OTA
  lastStatus = "Versi baru " + ver + " (token OK), update...";
  doOTAFromURL(fwurl);
}

// ===== UI =====
void handleRoot(){
  String html;
  html.reserve(9000);
  html  = "<!doctype html><html><head><meta charset='utf-8'><title>ESP32
OTA + Resource</title>";
  html += "<meta name='viewport' content='width=device-width,initial-
scale=1'>";
  html += "<style>body{font-family:system-
ui,Arial;margin:16px}.k{color:#555}.row{margin:6px 0}.card{border:1px
solid #ddd;border-radius:12px;padding:12px;margin:12px 0;box-shadow:0
1px 3px rgba(0,0,0,.05)}.h{font-
weight:700}.bar{height:10px;background:#eee;border-
radius:8px;overflow:hidden}.fill{height:100%;width:0%}.mono{font-
family:ui-monospace,Consolas,monospace}</style>";
  html += "</head><body><h2>ESP32 OTA + Monitoring (Realtime)</h2>";

  html += "<div class='card'><div class='row'><span class='k'>Status:
</span><span id='status'>-</span></div>";
  html += "<div class='row'><span class='k'>Firmware: </span><b
id='fw'>-</b></div>";
  html += "<div class='row'><span class='k'>IP: </span><b>" +
WiFi.localIP().toString() + "</b></div>";
  html += "<div class='row'><span class='k'>Uptime: </span><span
id='uptime'>-</span></div>";
  html += "<div class='row'><span class='k'>Suhu Air (DS18B20 @GPIO33):
</span><b id='ds18'>-</b> °C</div>";
  html += "<div class='row'><a href='/check'>Cek Update</a> | <a
href='/reboot'>Reboot</a></div></div>";

  html += "<div class='card'><div class='h'>CPU</div>";
  html += "<div class='row'>Core0: <b id='c0'>0%</b><div
class='bar'><div id='c0b' class='fill'></div></div></div>";
```

```
  html += "<div class='row'>Core1: <b id='c1'>0%</b><div
class='bar'><div id='c1b' class='fill'></div></div></div>";
  html += "<div class='row'>Total: <b id='ct'>0%</b><div
class='bar'><div id='ctb' class='fill'></div></div></div></div>";

  html += "<div class='card'><div class='h'>Memori</div>";
  html += "<div class='row'>Heap: <span id='heap'>-</span> (<b
id='heapp'>-</b>)<div class='bar'><div id='heapb'
class='fill'></div></div></div>";
  html += "<div class='row'>Flash: <span id='flash'>-</span> (<b
id='flashp'>-</b>)<div class='bar'><div id='flashb'
class='fill'></div></div></div></div>";

  html += "<div class='card'><div class='h'>Ringkasan OTA
terakhir</div>";
  html += "<div class='row mono'>Durasi: <b id='otat'>-</b> s,
Kecepatan: <b id='otas'>-</b> KB/s</div>";
  html += "<div class='row mono'>CPU avg (C0/C1/Tot): <b id='otaavg'>-
</b></div>";
  html += "<div class='row mono'>CPU max (C0/C1/Tot): <b id='otamax'>-
</b> | Sampel: <b id='otasamp'>-</b></div>";
  html += "<div class='row mono'>RAM selama OTA: Min <b id='otarammin'>-
</b>, Avg <b id='otaramavg'>-</b>, Max <b id='otarammax'>-
</b></div></div>";

  html += "<script>";
  html += "function fmtBytes(x){if(x<1024)return x+'
B';if(x<1048576)return (x/1024).toFixed(2)+' KB';return
(x/1048576).toFixed(2)+' MB'};";
  html += "function fmtUp(ms){let
s=Math.floor(ms/1000),m=Math.floor(s/60),h=Math.floor(m/60),d=Math.floor
(h/24);return (d?d+' hari, ':'')+((h%24)?(h%24)+' jam,
':'')+((m%60)?(m%60)+' menit, ':'')+(s%60)+' detik'};";
  html += "async function tick(){try{const r=await
fetch('/metrics');const j=await r.json();";
  html += "document.getElementById('status').textContent=j.status;";
  html += "document.getElementById('fw').textContent=j.firmware;";
  html +=
"document.getElementById('uptime').textContent=fmtUp(j.uptime_ms);";
```

```
  html +=
"if(j.water_temp_c!==null){document.getElementById('ds18').textContent=j
.water_temp_c.toFixed(2);}";

  html += "const
c0=j.cpu_core0||0,c1=j.cpu_core1||0,ct=j.cpu_total||0;";
  html +=
"document.getElementById('c0').textContent=c0.toFixed(1)+'%';document.ge
tElementById('c1').textContent=c1.toFixed(1)+'%';document.getElementById
('ct').textContent=ct.toFixed(1)+'%';";
  html +=
"document.getElementById('c0b').style.width=c0+'%';document.getElementBy
Id('c1b').style.width=c1+'%';document.getElementById('ctb').style.width=
ct+'%';";

  html += "const heapUsed=j.heap_total-j.heap_free;";
  html +=
"document.getElementById('heap').textContent=fmtBytes(heapUsed)+' /
'+fmtBytes(j.heap_total);";
  html +=
"document.getElementById('heapp').textContent=(j.heap_used_percent||0).t
oFixed(2)+'%';";
  html +=
"document.getElementById('heapb').style.width=(j.heap_used_percent||0)+'
%';";

  html +=
"document.getElementById('flash').textContent=fmtBytes(j.flash_sketch)+'
/ '+fmtBytes(j.flash_total)+' (free sketch
'+fmtBytes(j.flash_free_sketch)+')';";
  html +=
"document.getElementById('flashp').textContent=(j.flash_used_percent||0)
.toFixed(2)+'%';";
  html +=
"document.getElementById('flashb').style.width=(j.flash_used_percent||0)
+'%';";

  html +=
"if(j.ota_time_ms>0){document.getElementById('otat').textContent=(j.ota_
```

```
time_ms/1000).toFixed(2);document.getElementById('otas').textContent=(j.
ota_speed_kbps||0).toFixed(2);} ";
  html +=
"document.getElementById('otaavg').textContent=(j.ota_cpu_avg_core0||0).
toFixed(1)+'% / '+(j.ota_cpu_avg_core1||0).toFixed(1)+'% /
'+(j.ota_cpu_avg_total||0).toFixed(1)+'%';";
  html +=
"document.getElementById('otamax').textContent=(j.ota_cpu_max_core0||0).
toFixed(1)+'% / '+(j.ota_cpu_max_core1||0).toFixed(1)+'% /
'+(j.ota_cpu_max_total||0).toFixed(1)+'%';";
  html +=
"document.getElementById('otasamp').textContent=j.ota_cpu_samples||0;";
  html +=
"document.getElementById('otarammin').textContent=fmtBytes(j.ota_ram_min
||0);";
  html +=
"document.getElementById('otaramavg').textContent=fmtBytes(j.ota_ram_avg
||0);";
  html +=
"document.getElementById('otarammax').textContent=fmtBytes(j.ota_ram_max
||0);";

  html +=
"}catch(e){}};tick();setInterval(tick,1000);</script></body></html>";
  server.send(200, "text/html", html);
}

// ===== /metrics JSON =====
void handleMetrics(){
  long freeHeap        = ESP.getFreeHeap();
  long totalHeap       = ESP.getHeapSize();
  long flashSize       = ESP.getFlashChipSize();
  long sketchSize      = ESP.getSketchSize();
  long freeSketchSpace = ESP.getFreeSketchSpace();
  float heapUsedPct  = totalHeap > 0 ? (float)(totalHeap - freeHeap) /
(float)totalHeap * 100.0f : 0.0f;
  float flashUsedPct = flashSize > 0 ? (float)sketchSize           /
(float)flashSize * 100.0f : 0.0f;

  String j; j.reserve(1100);
```

```
  j += "{";
  j += "\"status\":\"" + lastStatus + "\",";
  j += "\"firmware\":\"" CURRENT_VERSION "\",";
  j += "\"uptime_ms\":" + String(millis()) + ",";

  if (isnan(waterTempC)) j += "\"water_temp_c\":null,";
  else                   j += "\"water_temp_c\":" + String(waterTempC,
2) + ",";

  j += "\"cpu_core0\":" + String(cpuCore[0],1) + ",";
  j += "\"cpu_core1\":" + String(cpuCore[1],1) + ",";
  j += "\"cpu_total\":" + String(cpuTotal,1) + ",";

  j += "\"heap_total\":" + String(totalHeap) + ",";
  j += "\"heap_free\":"  + String(freeHeap) + ",";
  j += "\"heap_used\":"  + String(totalHeap - freeHeap) + ",";
  j += "\"heap_used_percent\":" + String(heapUsedPct, 2) + ",";

  j += "\"flash_total\":" + String(flashSize) + ",";
  j += "\"flash_sketch\":" + String(sketchSize) + ",";
  j += "\"flash_free_sketch\":" + String(freeSketchSpace) + ",";
  j += "\"flash_used_percent\":" + String(flashUsedPct, 2) + ",";

  j += "\"ota_time_ms\":" + String(lastOtaTime) + ",";
  j += "\"ota_speed_kbps\":" + String(lastOtaSpeed) + ",";

  j += "\"ota_cpu_avg_core0\":" + String(lastOtaCpuAvgCore0, 1) + ",";
  j += "\"ota_cpu_avg_core1\":" + String(lastOtaCpuAvgCore1, 1) + ",";
  j += "\"ota_cpu_avg_total\":" + String(lastOtaCpuAvgTotal, 1) + ",";
  j += "\"ota_cpu_max_core0\":" + String(lastOtaCpuMaxCore0, 1) + ",";
  j += "\"ota_cpu_max_core1\":" + String(lastOtaCpuMaxCore1, 1) + ",";
  j += "\"ota_cpu_max_total\":" + String(lastOtaCpuMaxTotal, 1) + ",";
  j += "\"ota_cpu_samples\":"   + String(otaCpuSamples) + ",";

  j += "\"ota_ram_min\":" + String(minHeapDuringOTA) + ",";
  j += "\"ota_ram_avg\":" + String((long)lastOtaHeapAvg) + ",";
  j += "\"ota_ram_max\":" + String(maxHeapDuringOTA);
  j += "}";
  server.send(200, "application/json", j);
}
```

```
void handleReboot(){
  server.send(200, "text/plain", "Rebooting...");
  delay(1000);
  esp_restart();
}

// ===== CPU monitor task (1 Hz) =====
void cpuMonitorTask(void*){
  uint64_t s0 = idleCnt[0], s1 = idleCnt[1];
  vTaskDelay(pdMS_TO_TICKS(1000));
  idleBaseline[0] = max<uint64_t>(1, idleCnt[0] - s0);
  idleBaseline[1] = max<uint64_t>(1, idleCnt[1] - s1);
  Serial.printf("[CPU] Baseline idle: C0=%llu, C1=%llu (counts/1s)\n",
                (unsigned long long)idleBaseline[0], (unsigned long
long)idleBaseline[1]);

  for(;;){
    uint64_t c0s = idleCnt[0], c1s = idleCnt[1];
    vTaskDelay(pdMS_TO_TICKS(1000));
    uint64_t d0 = idleCnt[0] - c0s;
    uint64_t d1 = idleCnt[1] - c1s;

    float x0 = 100.0f - (float)d0 * 100.0f / (float)idleBaseline[0];
    float x1 = 100.0f - (float)d1 * 100.0f / (float)idleBaseline[1];
    if (x0 < 0) x0 = 0; if (x0 > 100) x0 = 100;
    if (x1 < 0) x1 = 0; if (x1 > 100) x1 = 100;

    cpuCore[0] = x0; cpuCore[1] = x1; cpuTotal = (x0 + x1) * 0.5f;

    if (otaMeasuring) {
      otaCpuSumCore[0] += cpuCore[0];
      otaCpuSumCore[1] += cpuCore[1];
      otaCpuSumTotal   += cpuTotal;
      otaCpuSamples++;

      if (cpuCore[0] > lastOtaCpuMaxCore0) lastOtaCpuMaxCore0 =
cpuCore[0];
      if (cpuCore[1] > lastOtaCpuMaxCore1) lastOtaCpuMaxCore1 =
cpuCore[1];
```

```
      if (cpuTotal   > lastOtaCpuMaxTotal) lastOtaCpuMaxTotal =
cpuTotal;

      long heapNow = ESP.getFreeHeap();
      otaHeapSum += heapNow;
      otaHeapSamples++;
    }
  }
}

// ===== Recalibrate (opsional) =====
void handleRecalibrate(){
  server.send(200, "text/plain", "Kalibrasi CPU 1 detik dimulai...");
  uint64_t s0 = idleCnt[0], s1 = idleCnt[1];
  delay(1000);
  idleBaseline[0] = max<uint64_t>(1, idleCnt[0] - s0);
  idleBaseline[1] = max<uint64_t>(1, idleCnt[1] - s1);
  Serial.printf("[CPU] Baseline baru: C0=%llu, C1=%llu\n",
                (unsigned long long)idleBaseline[0], (unsigned long
long)idleBaseline[1]);
}

void setup(){
  Serial.begin(115200);
  Serial.println("\n=== ESP32 OTA + Resource Dashboard — IdleHook +
DS18B20 @ GPIO33 ===");

  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print(".");
}
  Serial.printf("\nWiFi OK. IP: %s\n",
WiFi.localIP().toString().c_str());
  Serial.printf("Firmware saat ini: %s\n", CURRENT_VERSION);

  if (MDNS.begin(HOSTNAME)) { Serial.printf("mDNS aktif:
http://%s.local\n", HOSTNAME); MDNS.addService("http", "tcp", 80); }

  // DS18B20
  delay(300);
  ds18.begin();
```

```
  ds18.setResolution(12);
  ds18.setWaitForConversion(false);
  Serial.printf("DS18B20 device count: %d (GPIO %d)\n",
ds18.getDeviceCount(), DS18B20_PIN);
  ds18.requestTemperatures();
  dsConvStartMs = millis();
  dsConvPending = true;
  dsLastTickMs  = millis();

  // Idle hooks
  esp_register_freertos_idle_hook_for_cpu(idleHook0, 0);
  esp_register_freertos_idle_hook_for_cpu(idleHook1, 1);

  // Routes
  server.on("/", handleRoot);
  server.on("/metrics", handleMetrics);
  server.on("/check", [](){ server.send(200, "text/plain", "Memeriksa
update..."); checkAndUpdate(); });
  server.on("/reboot", handleReboot);
  server.on("/recalibrate", handleRecalibrate);
  server.begin();

  // CPU monitor task (Core 0)
  xTaskCreatePinnedToCore(cpuMonitorTask, "cpuMonitorTask", 4096,
nullptr, 1, &cpuMonTaskHandle, 0);
}

void loop(){
  server.handleClient();

  // DS18B20 non-blocking
  unsigned long now = millis();
  if (dsConvPending && (now - dsConvStartMs >= DS_CONV_MS)) {
    float t = ds18.getTempCByIndex(0);
    if (t != DEVICE_DISCONNECTED_C && t > -55 && t < 125) waterTempC =
t;
    dsConvPending = false;
  }
  if (now - dsLastTickMs >= DS_PERIOD_MS) {
    ds18.requestTemperatures();
```

```
    dsConvStartMs = now;
    dsConvPending = true;
    dsLastTickMs  = now;
  }
}
```

3. SHA 256

File Firmware SHA-256

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Update.h>
#include <ArduinoJson.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include "esp_system.h"

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_freertos_hooks.h"

// ===== DS18B20 (GPIO33) =====
#include <OneWire.h>
#include <DallasTemperature.h>
#define DS18B20_PIN 33
OneWire oneWire(DS18B20_PIN);
DallasTemperature ds18(&oneWire);
float waterTempC = NAN;
const uint32_t DS_PERIOD_MS = 1000;
const uint32_t DS_CONV_MS   = 750;
unsigned long dsLastTickMs = 0;
unsigned long dsConvStartMs = 0;
bool dsConvPending = false;

// ===== KONFIG =====
const char* WIFI_SSID = "POCO F4";
const char* WIFI_PASS = "11111111";
const char* CHECK_URL = "http://10.51.124.135/SHA/check_update.php";
const char* HOSTNAME  = "esp32-dashboard";
```

```
#define CURRENT_VERSION "v1.0.0"

// ===== WEB =====
WebServer server(80);

// ===== Status OTA =====
String lastStatus = "Idle";
unsigned long lastOtaTime = 0;
float lastOtaSpeed = 0.0f;
long minHeapDuringOTA = 0;
long maxHeapDuringOTA = 0;

// ===== CPU & RAM monitor (idle-hook) =====
static volatile uint64_t idleCnt[portNUM_PROCESSORS] = {0, 0};
static uint64_t idleBaseline[portNUM_PROCESSORS]     = {1, 1};
static float cpuCore[portNUM_PROCESSORS]             = {0.0f, 0.0f};
static float cpuTotal                                = 0.0f;
TaskHandle_t cpuMonTaskHandle                        = nullptr;
bool idleHook0() { idleCnt[0]++; return true; }
bool idleHook1() { idleCnt[1]++; return true; }

// ===== OTA CPU stats =====
static volatile bool otaMeasuring = false;
static float  otaCpuSumCore[2] = {0,0};
static float  otaCpuSumTotal   = 0;
static uint32_t otaCpuSamples  = 0;
static float lastOtaCpuAvgCore0  = 0, lastOtaCpuAvgCore1 = 0,
lastOtaCpuAvgTotal = 0;
static float lastOtaCpuMaxCore0  = 0, lastOtaCpuMaxCore1 = 0,
lastOtaCpuMaxTotal = 0;

// ===== OTA RAM avg =====
static uint64_t otaHeapSum = 0;
static uint32_t otaHeapSamples = 0;
static float    lastOtaHeapAvg = 0;   // bytes

// ===== SHA-256 (mbedTLS) =====
#include "mbedtls/sha256.h"
static String expectedSHA = "";   // dari manifest (64 hex lower-case)
static size_t expectedSize = 0;   // dari manifest (optional)
```

```
static String lastCalcSHA = "";   // hasil hash firmware terakhir

// ===== Utils =====
String formatBytes(long bytes){
  if (bytes < 1024) return String(bytes) + " B";
  else if (bytes < (1024 * 1024)) return String(bytes / 1024.0, 2) + "
KB";
  else return String(bytes / 1024.0 / 1024.0, 2) + " MB";
}
static String toHexString(const uint8_t* data, size_t len) {
  static const char* hex = "0123456789abcdef";
  String out; out.reserve(len * 2);
  for (size_t i = 0; i < len; i++) { out += hex[(data[i] >> 4) & 0x0F];
out += hex[data[i] & 0x0F]; }
  return out;
}

// ===== Helpers OTA stats =====
static inline void otaStatsStart(){
  otaMeasuring = true;
  otaCpuSumCore[0] = otaCpuSumCore[1] = 0;
  otaCpuSumTotal = 0;
  otaCpuSamples  = 0;
  lastOtaCpuMaxCore0 = lastOtaCpuMaxCore1 = lastOtaCpuMaxTotal = 0;
  minHeapDuringOTA = ESP.getFreeHeap();
  maxHeapDuringOTA = minHeapDuringOTA;
  otaHeapSum = 0;
  otaHeapSamples = 0;
  lastOtaHeapAvg = 0;
}
static inline void otaStatsFinish(){
  otaMeasuring = false;
  if (otaCpuSamples > 0) {
    lastOtaCpuAvgCore0 = otaCpuSumCore[0] / otaCpuSamples;
    lastOtaCpuAvgCore1 = otaCpuSumCore[1] / otaCpuSamples;
    lastOtaCpuAvgTotal = otaCpuSumTotal   / otaCpuSamples;
  } else {
    lastOtaCpuAvgCore0 = lastOtaCpuAvgCore1 = lastOtaCpuAvgTotal = 0;
  }
```

```
  if (otaHeapSamples > 0) lastOtaHeapAvg = (float)otaHeapSum /
(float)otaHeapSamples;
  else lastOtaHeapAvg = 0;
}

// ===== OTA (dengan verifikasi SHA-256) =====
bool doOTAFromURL(const String& binURL){
  lastStatus = "Mengunduh firmware...";
  otaStatsStart();

  HTTPClient http;
  http.setTimeout(60000);
  http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);

  if (!http.begin(binURL)) { lastStatus = "http.begin gagal";
otaStatsFinish(); return false; }
  int code = http.GET();
  if (code != HTTP_CODE_OK) { lastStatus = "HTTP error " + String(code);
http.end(); otaStatsFinish(); return false; }

  int contentLength = http.getSize(); // -1 kalau chunked
  if (!Update.begin(contentLength > 0 ? contentLength :
UPDATE_SIZE_UNKNOWN)) {
    lastStatus = "Update.begin gagal"; http.end(); otaStatsFinish();
return false;
  }

  WiFiClient* stream = http.getStreamPtr();
  uint8_t buf[2048];
  size_t totalWritten = 0;
  unsigned long t0 = millis();

  // init SHA
  mbedtls_sha256_context shaCtx;
  mbedtls_sha256_init(&shaCtx);
  mbedtls_sha256_starts(&shaCtx, 0); // 0 = SHA-256

  while (true) {
    int n = stream->readBytes((char*)buf, sizeof(buf));
```

```
    if (n < 0) { lastStatus = "read error"; Update.abort(); http.end();
mbedtls_sha256_free(&shaCtx); otaStatsFinish(); return false; }
    if (n == 0) { if (!stream->available()) break; delay(1); continue; }

    if (Update.write(buf, n) != (size_t)n) {
      lastStatus = "Update.write error"; Update.abort(); http.end();
mbedtls_sha256_free(&shaCtx); otaStatsFinish(); return false;
    }
    totalWritten += n;

    mbedtls_sha256_update(&shaCtx, buf, n);

    long heapNow = ESP.getFreeHeap();
    if (heapNow < minHeapDuringOTA) minHeapDuringOTA = heapNow;
    if (heapNow > maxHeapDuringOTA) maxHeapDuringOTA = heapNow;
  }
  http.end();

  // finalize hash
  uint8_t digest[32];
  mbedtls_sha256_finish(&shaCtx, digest);
  mbedtls_sha256_free(&shaCtx);
  lastCalcSHA = toHexString(digest, sizeof(digest)); // lower-case

  // verifikasi SHA (dan size opsional)
  if (expectedSHA.length() > 0 && lastCalcSHA != expectedSHA) {
    Serial.printf("[OTA] SHA mismatch!\nExpected: %s\nGot     : %s\n",
expectedSHA.c_str(), lastCalcSHA.c_str());
    lastStatus = "SHA256 mismatch, OTA dibatalkan";
    Update.abort();  // JANGAN finalize image
    otaStatsFinish();
    return false;
  }
  if (expectedSize > 0 && totalWritten != expectedSize) {
    Serial.printf("[OTA] Size mismatch: expected=%u got=%u\n",
(unsigned)expectedSize, (unsigned)totalWritten);
    lastStatus = "Ukuran tidak sesuai manifest";
    Update.abort();
    otaStatsFinish();
    return false;
```

```
  }
  if (contentLength > 0 && totalWritten != (size_t)contentLength) {
    Serial.printf("[OTA] Incomplete download: contentLength=%d
written=%u\n", contentLength, (unsigned)totalWritten);
    lastStatus = "Bytes terunduh tidak lengkap";
    Update.abort();
    otaStatsFinish();
    return false;
  }

  // finalize image valid
  if (!Update.end(true)) { lastStatus = "Update.end err=" +
String(Update.getError()); otaStatsFinish(); return false; }
  if (!Update.isFinished()) { lastStatus = "Update belum selesai";
otaStatsFinish(); return false; }

  unsigned long dt = millis() - t0;
  lastOtaTime  = dt;
  lastOtaSpeed = (totalWritten / 1024.0) / (dt / 1000.0);
  lastStatus   = "Update sukses, silakan klik Reboot";

  otaStatsFinish();

  Serial.printf("[OTA] Total: %u bytes, Durasi: %.2f s, Kecepatan: %.2f
KB/s\n",
                (unsigned)totalWritten, dt / 1000.0, lastOtaSpeed);
  Serial.printf("[SHA] Expected: %s\n[SHA] Calculated: %s\n",
expectedSHA.c_str(), lastCalcSHA.c_str());
  Serial.printf("[RAM] OTA -> Min: %.2f KB, Avg: %.2f KB, Max: %.2f
KB\n",
                minHeapDuringOTA/1024.0, lastOtaHeapAvg/1024.0,
maxHeapDuringOTA/1024.0);
  Serial.printf("[CPU] OTA avg C0=%.1f C1=%.1f Tot=%.1f | max C0=%.1f
C1=%.1f Tot=%.1f | sample=%u\n",
                lastOtaCpuAvgCore0, lastOtaCpuAvgCore1,
lastOtaCpuAvgTotal,
                lastOtaCpuMaxCore0, lastOtaCpuMaxCore1,
lastOtaCpuMaxTotal, otaCpuSamples);
  return true;
}
```

```
void checkAndUpdate(){
  lastStatus = "Memeriksa manifest...";
  HTTPClient http;
  http.setTimeout(20000);
  http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);

  if (!http.begin(CHECK_URL)) { lastStatus = "Manifest gagal dibuka";
return; }
  int code = http.GET();
  if (code != HTTP_CODE_OK) { lastStatus = "Manifest HTTP " +
String(code); http.end(); return; }
  String body = http.getString(); http.end();

  StaticJsonDocument<512> doc;
  if (deserializeJson(doc, body)) { lastStatus = "JSON parse error";
return; }

  String latest = doc["version"] | "";
  String fwurl  = doc["url"]     | "";
  expectedSHA   = doc["sha256"]  | "";
  expectedSize  = doc["size"]    | 0;

  expectedSHA.trim(); expectedSHA.toLowerCase();

  // validasi manifest
  if (latest == "" || fwurl == "" || expectedSHA == "" ||
expectedSHA.length() != 64) {
    lastStatus = "Manifest tidak lengkap";
    Serial.printf("[Manifest] version='%s' url='%s' sha='%s' size=%u\n",
                  latest.c_str(), fwurl.c_str(), expectedSHA.c_str(),
(unsigned)expectedSize);
    return;
  }
  if (latest == CURRENT_VERSION) { lastStatus = "Sudah versi terbaru";
return; }

  lastStatus = "Versi baru " + latest + " tersedia, update...";
  doOTAFromURL(fwurl);
}
```

```
// ===== UI =====
void handleRoot(){
  String html;
  html.reserve(9500);
  html  = "<!doctype html><html><head><meta charset='utf-8'><title>ESP32
OTA + Resource</title>";
  html += "<meta name='viewport' content='width=device-width,initial-
scale=1'>";
  html += "<style>body{font-family:system-
ui,Arial;margin:16px}.k{color:#555}.row{margin:6px 0}.card{border:1px
solid #ddd;border-radius:12px;padding:12px;margin:12px 0;box-shadow:0
1px 3px rgba(0,0,0,.05)}.h{font-
weight:700}.bar{height:10px;background:#eee;border-
radius:8px;overflow:hidden}.fill{height:100%;width:0%}.mono{font-
family:ui-monospace,Consolas,monospace}</style>";
  html += "</head><body><h2>ESP32 OTA + Monitoring (Realtime)</h2>";

  html += "<div class='card'><div class='row'><span class='k'>Status:
</span><span id='status'>-</span></div>";
  html += "<div class='row'><span class='k'>Firmware: </span><b
id='fw'>-</b></div>";
  html += "<div class='row'><span class='k'>IP: </span><b>" +
WiFi.localIP().toString() + "</b></div>";
  html += "<div class='row'><span class='k'>Uptime: </span><span
id='uptime'>-</span></div>";
  html += "<div class='row'><span class='k'>Suhu Air (DS18B20 @GPIO33):
</span><b id='ds18'>-</b> °C</div>";
  html += "<div class='row mono'>SHA Expected: <b id='shaexp'>-
</b></div>";
  html += "<div class='row mono'>SHA Calculated: <b id='shacal'>-
</b></div>";
  html += "<div class='row'><a href='/check'>Cek Update</a> | <a
href='/reboot'>Reboot</a></div></div>";

  html += "<div class='card'><div class='h'>CPU</div>";
  html += "<div class='row'>Core0: <b id='c0'>0%</b><div
class='bar'><div id='c0b' class='fill'></div></div></div>";
  html += "<div class='row'>Core1: <b id='c1'>0%</b><div
class='bar'><div id='c1b' class='fill'></div></div></div>";
```

```
  html += "<div class='row'>Total: <b id='ct'>0%</b><div
class='bar'><div id='ctb' class='fill'></div></div></div></div>";

  html += "<div class='card'><div class='h'>Memori</div>";
  html += "<div class='row'>Heap: <span id='heap'>-</span> (<b
id='heapp'>-</b>)<div class='bar'><div id='heapb'
class='fill'></div></div></div>";
  html += "<div class='row'>Flash: <span id='flash'>-</span> (<b
id='flashp'>-</b>)<div class='bar'><div id='flashb'
class='fill'></div></div></div></div>";

  html += "<div class='card'><div class='h'>Ringkasan OTA
terakhir</div>";
  html += "<div class='row mono'>Durasi: <b id='otat'>-</b> s,
Kecepatan: <b id='otas'>-</b> KB/s</div>";
  html += "<div class='row mono'>CPU avg (C0/C1/Tot): <b id='otaavg'>-
</b></div>";
  html += "<div class='row mono'>CPU max (C0/C1/Tot): <b id='otamax'>-
</b> | Sampel: <b id='otasamp'>-</b></div>";
  html += "<div class='row mono'>RAM selama OTA: Min <b id='otarammin'>-
</b>, Avg <b id='otaramavg'>-</b>, Max <b id='otarammax'>-
</b></div></div>";

  html += "<script>";
  html += "function fmtBytes(x){if(x<1024)return x+'
B';if(x<1048576)return (x/1024).toFixed(2)+' KB';return
(x/1048576).toFixed(2)+' MB'};";
  html += "function fmtUp(ms){let
s=Math.floor(ms/1000),m=Math.floor(s/60),h=Math.floor(m/60),d=Math.floor
(h/24);return (d?d+' hari, ':'')+((h%24)?(h%24)+' jam,
':'')+((m%60)?(m%60)+' menit, ':'')+(s%60)+' detik'};";
  html += "async function tick(){try{const r=await
fetch('/metrics');const j=await r.json();";
  html += "document.getElementById('status').textContent=j.status;";
  html += "document.getElementById('fw').textContent=j.firmware;";
  html +=
"document.getElementById('uptime').textContent=fmtUp(j.uptime_ms);";
  html +=
"if(j.water_temp_c!==null){document.getElementById('ds18').textContent=j
```

```
.water_temp_c.toFixed(2);}else{document.getElementById('ds18').textConte
nt='-';}";
  html +=
"document.getElementById('shaexp').textContent=j.ota_expect_sha||'-';";
  html +=
"document.getElementById('shacal').textContent=j.ota_calc_sha||'-';";

  html += "const
c0=j.cpu_core0||0,c1=j.cpu_core1||0,ct=j.cpu_total||0;";
  html +=
"document.getElementById('c0').textContent=c0.toFixed(1)+'%';document.ge
tElementById('c1').textContent=c1.toFixed(1)+'%';document.getElementById
('ct').textContent=ct.toFixed(1)+'%';";
  html +=
"document.getElementById('c0b').style.width=c0+'%';document.getElementBy
Id('c1b').style.width=c1+'%';document.getElementById('ctb').style.width=
ct+'%';";

  html += "const heapUsed=j.heap_total-j.heap_free;";
  html +=
"document.getElementById('heap').textContent=fmtBytes(heapUsed)+' /
'+fmtBytes(j.heap_total);";
  html +=
"document.getElementById('heapp').textContent=(j.heap_used_percent||0).t
oFixed(2)+'%';";
  html +=
"document.getElementById('heapb').style.width=(j.heap_used_percent||0)+'
%';";

  html +=
"document.getElementById('flash').textContent=fmtBytes(j.flash_sketch)+'
/ '+fmtBytes(j.flash_total)+' (free sketch
'+fmtBytes(j.flash_free_sketch)+')';";
  html +=
"document.getElementById('flashp').textContent=(j.flash_used_percent||0)
.toFixed(2)+'%';";
  html +=
"document.getElementById('flashb').style.width=(j.flash_used_percent||0)
+'%';";
```

```
  html +=
"if(j.ota_time_ms>0){document.getElementById('otat').textContent=(j.ota_
time_ms/1000).toFixed(2);document.getElementById('otas').textContent=(j.
ota_speed_kbps||0).toFixed(2);} ";
  html +=
"document.getElementById('otaavg').textContent=(j.ota_cpu_avg_core0||0).
toFixed(1)+'% / '+(j.ota_cpu_avg_core1||0).toFixed(1)+'% /
'+(j.ota_cpu_avg_total||0).toFixed(1)+'%';";
  html +=
"document.getElementById('otamax').textContent=(j.ota_cpu_max_core0||0).
toFixed(1)+'% / '+(j.ota_cpu_max_core1||0).toFixed(1)+'% /
'+(j.ota_cpu_max_total||0).toFixed(1)+'%';";
  html +=
"document.getElementById('otasamp').textContent=j.ota_cpu_samples||0;";
  html +=
"document.getElementById('otarammin').textContent=fmtBytes(j.ota_ram_min
||0);";
  html +=
"document.getElementById('otaramavg').textContent=fmtBytes(j.ota_ram_avg
||0);";
  html +=
"document.getElementById('otarammax').textContent=fmtBytes(j.ota_ram_max
||0);";

  html +=
"}catch(e){}};tick();setInterval(tick,1000);</script></body></html>";
  server.send(200, "text/html", html);
}

// ===== /metrics JSON =====
void handleMetrics(){
  long freeHeap        = ESP.getFreeHeap();
  long totalHeap       = ESP.getHeapSize();
  long flashSize       = ESP.getFlashChipSize();
  long sketchSize      = ESP.getSketchSize();
  long freeSketchSpace = ESP.getFreeSketchSpace();
  float heapUsedPct  = totalHeap > 0 ? (float)(totalHeap - freeHeap) /
(float)totalHeap * 100.0f : 0.0f;
  float flashUsedPct = flashSize > 0 ? (float)sketchSize           /
(float)flashSize * 100.0f : 0.0f;
```

```
  String j; j.reserve(1300);
  j += "{";
  j += "\"status\":\"" + lastStatus + "\",";
  j += "\"firmware\":\"" CURRENT_VERSION "\",";
  j += "\"uptime_ms\":" + String(millis()) + ",";

  if (isnan(waterTempC)) j += "\"water_temp_c\":null,";
  else                   j += "\"water_temp_c\":" + String(waterTempC,
2) + ",";

  j += "\"cpu_core0\":" + String(cpuCore[0],1) + ",";
  j += "\"cpu_core1\":" + String(cpuCore[1],1) + ",";
  j += "\"cpu_total\":" + String(cpuTotal,1) + ",";

  j += "\"heap_total\":" + String(totalHeap) + ",";
  j += "\"heap_free\":"  + String(freeHeap) + ",";
  j += "\"heap_used\":"  + String(totalHeap - freeHeap) + ",";
  j += "\"heap_used_percent\":" + String(heapUsedPct, 2) + ",";

  j += "\"flash_total\":" + String(flashSize) + ",";
  j += "\"flash_sketch\":" + String(sketchSize) + ",";
  j += "\"flash_free_sketch\":" + String(freeSketchSpace) + ",";
  j += "\"flash_used_percent\":" + String(flashUsedPct, 2) + ",";

  j += "\"ota_time_ms\":" + String(lastOtaTime) + ",";
  j += "\"ota_speed_kbps\":" + String(lastOtaSpeed) + ",";

  j += "\"ota_cpu_avg_core0\":" + String(lastOtaCpuAvgCore0, 1) + ",";
  j += "\"ota_cpu_avg_core1\":" + String(lastOtaCpuAvgCore1, 1) + ",";
  j += "\"ota_cpu_avg_total\":" + String(lastOtaCpuAvgTotal, 1) + ",";
  j += "\"ota_cpu_max_core0\":" + String(lastOtaCpuMaxCore0, 1) + ",";
  j += "\"ota_cpu_max_core1\":" + String(lastOtaCpuMaxCore1, 1) + ",";
  j += "\"ota_cpu_max_total\":" + String(lastOtaCpuMaxTotal, 1) + ",";
  j += "\"ota_cpu_samples\":"   + String(otaCpuSamples) + ",";

  j += "\"ota_ram_min\":" + String(minHeapDuringOTA) + ",";
  j += "\"ota_ram_avg\":" + String((long)lastOtaHeapAvg) + ",";
  j += "\"ota_ram_max\":" + String(maxHeapDuringOTA) + ",";
```

```
  // ringkasan SHA
  j += "\"ota_expect_sha\":\"" + expectedSHA + "\",";
  j += "\"ota_calc_sha\":\""   + lastCalcSHA + "\"";

  j += "}";
  server.send(200, "application/json", j);
}

void handleReboot(){
  server.send(200, "text/plain", "Rebooting...");
  delay(1000);
  esp_restart();
}

// ===== CPU monitor task (1 Hz) =====
void cpuMonitorTask(void*){
  uint64_t s0 = idleCnt[0], s1 = idleCnt[1];
  vTaskDelay(pdMS_TO_TICKS(1000));
  idleBaseline[0] = max<uint64_t>(1, idleCnt[0] - s0);
  idleBaseline[1] = max<uint64_t>(1, idleCnt[1] - s1);
  Serial.printf("[CPU] Baseline idle: C0=%llu, C1=%llu (counts/1s)\n",
                (unsigned long long)idleBaseline[0], (unsigned long
long)idleBaseline[1]);

  for(;;){
    uint64_t c0s = idleCnt[0], c1s = idleCnt[1];
    vTaskDelay(pdMS_TO_TICKS(1000));
    uint64_t d0 = idleCnt[0] - c0s;
    uint64_t d1 = idleCnt[1] - c1s;

    float x0 = 100.0f - (float)d0 * 100.0f / (float)idleBaseline[0];
    float x1 = 100.0f - (float)d1 * 100.0f / (float)idleBaseline[1];
    if (x0 < 0) x0 = 0; if (x0 > 100) x0 = 100;
    if (x1 < 0) x1 = 0; if (x1 > 100) x1 = 100;

    cpuCore[0] = x0; cpuCore[1] = x1; cpuTotal = (x0 + x1) * 0.5f;

    if (otaMeasuring) {
      otaCpuSumCore[0] += cpuCore[0];
      otaCpuSumCore[1] += cpuCore[1];
```

```
      otaCpuSumTotal   += cpuTotal;
      otaCpuSamples++;

      if (cpuCore[0] > lastOtaCpuMaxCore0) lastOtaCpuMaxCore0 =
cpuCore[0];
      if (cpuCore[1] > lastOtaCpuMaxCore1) lastOtaCpuMaxCore1 =
cpuCore[1];
      if (cpuTotal   > lastOtaCpuMaxTotal) lastOtaCpuMaxTotal =
cpuTotal;

      long heapNow = ESP.getFreeHeap();
      otaHeapSum += heapNow;
      otaHeapSamples++;
    }
  }
}

// ===== Recalibrate (opsional) =====
void handleRecalibrate(){
  server.send(200, "text/plain", "Kalibrasi CPU 1 detik dimulai...");
  uint64_t s0 = idleCnt[0], s1 = idleCnt[1];
  delay(1000);
  idleBaseline[0] = max<uint64_t>(1, idleCnt[0] - s0);
  idleBaseline[1] = max<uint64_t>(1, idleCnt[1] - s1);
  Serial.printf("[CPU] Baseline baru: C0=%llu, C1=%llu\n",
                (unsigned long long)idleBaseline[0], (unsigned long
long)idleBaseline[1]);
}

void setup(){
  Serial.begin(115200);
  Serial.println("\n=== ESP32 OTA (SHA-256 Verified) + Resource
Dashboard — IdleHook + DS18B20 @ GPIO33 ===");

  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print(".");
}
  Serial.printf("\nWiFi OK. IP: %s\n",
WiFi.localIP().toString().c_str());
  Serial.printf("Firmware saat ini: %s\n", CURRENT_VERSION);
```

```
  if (MDNS.begin(HOSTNAME)) { Serial.printf("mDNS aktif:
http://%s.local\n", HOSTNAME); MDNS.addService("http", "tcp", 80); }

  // DS18B20
  delay(300);
  ds18.begin();
  ds18.setResolution(12);
  ds18.setWaitForConversion(false);
  Serial.printf("DS18B20 device count: %d (GPIO %d)\n",
ds18.getDeviceCount(), DS18B20_PIN);
  ds18.requestTemperatures();
  dsConvStartMs = millis();
  dsConvPending = true;
  dsLastTickMs  = millis();

  // Idle hooks
  esp_register_freertos_idle_hook_for_cpu(idleHook0, 0);
  esp_register_freertos_idle_hook_for_cpu(idleHook1, 1);

  // Routes
  server.on("/", handleRoot);
  server.on("/metrics", handleMetrics);
  server.on("/check", [](){ server.send(200, "text/plain", "Memeriksa
update..."); checkAndUpdate(); });
  server.on("/reboot", handleReboot);
  server.on("/recalibrate", handleRecalibrate);
  server.begin();

  // CPU monitor task (Core 0)
  xTaskCreatePinnedToCore(cpuMonitorTask, "cpuMonitorTask", 4096,
nullptr, 1, &cpuMonTaskHandle, 0);
}

void loop(){
  server.handleClient();

  // DS18B20 non-blocking
  unsigned long now = millis();
  if (dsConvPending && (now - dsConvStartMs >= DS_CONV_MS)) {
```

```
    float t = ds18.getTempCByIndex(0);
    if (t != DEVICE_DISCONNECTED_C && t > -55 && t < 125) waterTempC =
t;
    dsConvPending = false;
  }
  if (now - dsLastTickMs >= DS_PERIOD_MS) {
    ds18.requestTemperatures();
    dsConvStartMs = now;
    dsConvPending = true;
    dsLastTickMs  = now;
  }
}
```