Journal of Applied Informatics and Computing (JAIC)
Vol.x, No.x, Juli 201x, pp. xx~xx
e-ISSN: 2548-6861

Perancangan dan Implementasi Mekanisme Autentikasi Pesan MQTT Berbasis HMAC-SHA256 pada Arsitektur Industrial Control System Berbiaya Rendah

Muhamad Hanif Widagdo[1]*, Hamdani Arif [2]**,**
[1, 2], Rekayasa Keamanan Siber, Politeknik Negeri Batam
widagdohanif@gmail.com [1], hamdani.arif@polibatam.ac.id [2],

Article Info

Article history:

Received …
Revised …
Accepted …

Keyword:

Arduino Uno R3, HMAC-SHA256, Industrial Control System, MQTT, XOR Checksum.

ABSTRACT

The convergence of Information Technology (IT) and Operational Technology (OT) through the Industrial Internet of Things (IIoT) has transformed manufacturing by enabling **real-time** monitoring and remote machine control. However, cost considerations in **medium-scale** industries often lead to the use of **low-cost** Industrial Control Systems (ICS) that utilize **resource-constrained** devices, such as the Arduino Uno R3, as edge nodes. This architecture introduces severe security vulnerabilities, particularly due to the transmission of unencrypted MQTT messages over port 1883, making the system highly susceptible to Man-in-the-Middle (MitM) and command forgery attacks. Traditional transport layer security mitigations like Mutual TLS (mTLS) are infeasible due to the heavy computational overhead and the Arduino's limited SRAM (2 KB), which risks triggering heap exhaustion. This study proposes an application layer security mechanism using the HMAC-SHA256 algorithm to guarantee payload authenticity and integrity on the Web–Broker–Bridge path, coupled with a bitwise XOR Checksum on the Serial UART path to detect physical transmission errors. The proposed design is implemented using a .NET Framework 4.8 Bridge and an Arduino Uno R3 firmware, and evaluated within an isolated physical testbed. The experimental results demonstrate that the proposed mechanism successfully achieves a 100% block rate against spoofing attacks and a 100% detection rate against tampering on state files. Furthermore, the security overhead remains highly acceptable; the HMAC-SHA256 computation, offloaded to the Gateway layer, incurs an average processing latency of only 0.236 ms with a median of 0.070 ms (well below the 200 ms operational threshold), while the integrity validation layer on the edge device adds only 170 bytes of Flash ROM and a transient 20-byte SRAM stack usage, imposing no persistent memory burden on the Arduino Uno microcontroller, ensuring that the physical control of the machinery remains stable and **real-time**.

I. Pendahuluan

Tren integrasi antara Information Technology (IT) dan Operational Technology (OT) yang difasilitasi oleh Industrial Internet of Things (IIoT) telah mengubah cara sistem manufaktur beroperasi dari yang semula tertutup dan terisolasi, kini terhubung melalui antarmuka web dan jaringan komunikasi yang jauh lebih terbuka. Konsekuensi dari keterbukaan ini turut dirasakan pada lingkup Industrial Control System (ICS) berskala kecil dan berbiaya rendah, di mana keterbatasan sumber daya perangkat keras seperti pada mikrokontroler Arduino Uno R3 menjadi pertimbangan utama dalam memilih teknologi pendukung. Dalam konteks inilah protokol Message Queuing Telemetry Transport (MQTT) menjadi pilihan yang relevan, karena sifatnya yang ringan dari sisi komputasi dan relatif sederhana untuk diimplementasikan pada perangkat dengan spesifikasi terbatas.. Salah satu implementasinya terdapat pada sistem *Cutting and Forming*, di mana dashboard web mengirimkan perintah kendali melalui broker MQTT, lalu diproses oleh aplikasi bridge sebelum diteruskan ke mesin melalui komunikasi serial.

e-ISSN: 2548-6861

Namun, arsitektur tersebut memiliki risiko keamanan yang signifikan. Penggunaan MQTT pada port 1883 tanpa enkripsi menyebabkan pertukaran data berlangsung dalam bentuk plaintext, sehingga membuka peluang terjadinya penyadapan, man-in-the-middle, dan pemalsuan perintah oleh aktor di jaringan lokal [4][6]. Dalam konteks ICS, kelemahan ini tidak hanya berdampak pada kerahasiaan data, tetapi juga dapat memengaruhi integritas operasional mesin fisik apabila perintah yang tidak sah berhasil dieksekusi [5][6].

Pendekatan pengamanan yang umum digunakan adalah Transport Layer Security (TLS) atau Mutual TLS, namun mekanisme tersebut tidak selalu sesuai untuk perangkat resource-constrained [7][10]. Arduino Uno R3 hanya memiliki SRAM sebesar 2 KB, sehingga beban komputasi dan memori dari protokol keamanan berbasis sertifikat berpotensi menimbulkan latensi tinggi, kegagalan alokasi memori, atau gangguan pada fungsi kontrol. Kondisi ini menciptakan kesenjangan antara kebutuhan keamanan dan keterbatasan perangkat yang belum sepenuhnya dijawab oleh literatur yang ada.

Beberapa penelitian terdahulu tentang keamanan komunikasi IoT dan IIoT menunjukkan bahwa sebagian besar pendekatan masih berfokus pada enkripsi kanal atau kriptografi ringan pada perangkat dengan sumber daya terbatas, namun belum mengkaji secara spesifik mekanisme autentikasi payload MQTT yang ringan untuk mencegah command forgery dan tampering pada arsitektur ICS legacy berbiaya rendah dengan mesin fisik. Penelitian ini mengisi celah tersebut dengan mengusulkan autentikasi payload berbasis HMAC-SHA256 pada lapisan aplikasi bridge yang dikombinasikan dengan validasi XOR checksum pada jalur serial, sementara Tabel 1 merangkum perbandingan fokus, hasil utama, dan keterbatasan studi-studi terdahulu.

Tabel 1. Perbandingan studi terdahulu (state of the art)

Peneliti / Tahun	**Fokus Penelitian**	**Hasil Utama**	**Keterbatasan / Gap**
Jacinto et al. (2024)	End-to-End Encryption pada MQTT	Kerahasiaan data terjaga	Beban memori tinggi, kurang sesuai untuk perangkat terbatas
Noman & Abu-Sharkh (2023)	Analisis command injection pada IoT wireless	Mengidentifikasi risiko transmisi plaintext	Tidak menyediakan mitigasi kriptografis
Kurniawan et al. (2022)	Kriptografi ringan Simon/Speck pada Arduino IoT	Efisien untuk perangkat terbatas sumber daya	Tidak spesifik pada autentikasi payload MQTT untuk ICS
Erroutbi et al. (2019)	HMAC mutual authentication untuk IoT-fog	Overhead komputasi rendah	Hanya diuji secara simulatif, belum pada mesin fisik
Khan et al. (2018)	STRIDE-based threat modeling for CPS	Pemetaan ancaman IT-OT komprehensif	Masih konseptual, belum diikuti implementasi mitigasi

Berdasarkan studi terdahulu, terdapat kesenjangan pada mekanisme autentikasi payload MQTT yang ringan, spesifik untuk mencegah command forgery dan tampering, serta divalidasi langsung pada arsitektur ICS legacy berbiaya rendah dengan mesin fisik. Penelitian ini mengisi celah tersebut melalui implementasi autentikasi payload berbasis HMAC-SHA256 pada lapisan aplikasi bridge, disertai validasi XOR checksum pada jalur serial. Pendekatan ini dipilih untuk menjaga integritas dan autentikasi pesan tanpa menambah beban komputasi berlebihan pada perangkat dengan sumber daya terbatas. Ruang lingkup penelitian dibatasi pada command forgery dan tampering; replay attack tidak termasuk karena HMAC-SHA256 menjamin authenticity dan integrity, bukan freshness. Selain itu, PSK pada mekanisme autentikasi masih disimpan secara statis sehingga rotasi kunci dinamis belum diimplementasikan.

Meskipun node edge (Arduino) terisolasi dari potensi kebocoran PSK karena hanya memverifikasi checksum XOR pada jalur serial, kompromi terhadap berkas konfigurasi di sisi Bridge atau Dashboard tetap dapat memaparkan PSK kepada penyerang. Meskipun dampak kompromi kunci ini dibatasi oleh mekanisme validasi timestamp freshness window selama 30 detik, serta otentikasi tingkat jaringan MQTT, untuk penerapan skema rotasi kunci dinamis otomatis berbasis protokol ringan berada di luar cakupan penelitian ini dan menjadi fokus untuk penelitian lanjutan.

Berdasarkan kesenjangan yang telah diuraikan, penelitian ini merumuskan pertanyaan penelitian sebagai berikut:

1. RQ1: Bagaimana rancangan mekanisme autentikasi payload MQTT berbasis *HMAC-SHA256* pada lapisan aplikasi bridge, disertai validasi *XOR checksum* pada komunikasi serial, yang sesuai dengan keterbatasan

sumber daya perangkat low-cost seperti Arduino Uno R3?
2. RQ2: Sejauh mana mekanisme autentikasi *HMAC-SHA256* dan validasi *XOR checksum* efektif mencegah command forgery dan tampering pada arsitektur ICS *Cutting and Forming*?
3. RQ3: Bagaimana pengaruh penerapan mekanisme autentikasi tersebut terhadap performa sistem, misalnya latensi komunikasi dan kestabilan fungsi kontrol, dibandingkan dengan arsitektur tanpa autentikasi?

Berdasarkan kesenjangan tersebut, penelitian ini mengusulkan mekanisme autentikasi pesan MQTT berbasis HMAC-SHA256 pada lapisan aplikasi bridge, serta validasi XOR checksum pada jalur komunikasi serial menuju mikrokontroler. HMAC-SHA256 dipilih karena mampu memverifikasi keaslian dan integritas payload tanpa perlu mengenkripsi seluruh saluran komunikasi, sehingga overhead komputasi dapat dijaga seminimal mungkin [8][13]. XOR checksum diterapkan sebagai lapisan deteksi kesalahan transmisi pada komunikasi serial antara bridge dan Arduino, yang tidak memerlukan tambahan memori signifikan. Berdasarkan rumusan masalah di atas, penelitian ini bertujuan untuk:

1. Merancang dan mengimplementasikan mekanisme autentikasi payload MQTT berbasis HMAC-SHA256 pada lapisan aplikasi bridge, disertai validasi XOR checksum pada komunikasi serial, yang sesuai dengan keterbatasan sumber daya perangkat low-cost seperti Arduino Uno R3.
2. Mengevaluasi efektivitas mekanisme autentikasi dalam mencegah command forgery dan tampering pada arsitektur ICS Cutting and Forming melalui pengujian keamanan pada lingkungan dengan mesin fisik.
3. Menganalisis pengaruh penerapan mekanisme autentikasi terhadap performa sistem, mencakup latensi komunikasi dan kestabilan fungsi kontrol, dibandingkan dengan arsitektur tanpa autentikasi.

II. Metode

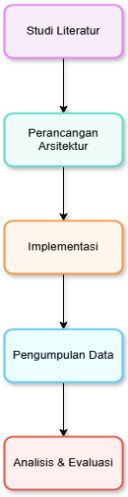
A. *Pendekatan Penelitian*

Penelitian ini menggunakan pendekatan kuantitatif eksperimental dengan sifat terapan, bertujuan untuk merancang, mengimplementasikan, dan mengevaluasi mekanisme autentikasi pesan MQTT berbasis HMAC-SHA256 pada arsitektur *Industrial Control System* (ICS) berbiaya rendah. Pendekatan ini dipilih karena penelitian berfokus pada pengukuran dampak langsung penerapan mekanisme keamanan terhadap tingkat perlindungan sistem dan overhead kinerja yang ditimbulkan.

Variabel independen dalam penelitian ini adalah penerapan autentikasi payload MQTT menggunakan HMAC-SHA256 pada lapisan aplikasi Bridge serta validasi XOR checksum pada jalur komunikasi serial menuju mikrokontroler. Variabel dependen meliputi rasio blokir terhadap serangan spoofing, rasio deteksi terhadap manipulasi data pada berkas state.json, overhead latensi komputasi, dan konsumsi SRAM pada perangkat edge. Variabel kontrol meliputi ukuran payload, topik MQTT, baud rate komunikasi serial, spesifikasi perangkat keras, serta kondisi jaringan pengujian yang dibuat terisolasi.

B. *Tahapan Penelitian*

Penelitian dilaksanakan melalui lima tahapan utama: (1) studi literatur dan analisis kerentanan, (2) perancangan arsitektur keamanan, (3) implementasi solusi, (4) pengujian dan pengumpulan data, serta (5) evaluasi hasil. Pada tahap analisis awal, kerentanan dipetakan menggunakan kerangka STRIDE untuk alur komunikasi Machine-to-Machine, dengan fokus pada spoofing di jalur MQTT dan tampering terhadap data persistensi lokal aplikasi Bridge . Tahap perancangan menghasilkan dua mekanisme utama: autentikasi payload MQTT berbasis HMAC-SHA256 yang diverifikasi di sisi Bridge, serta validasi integritas jalur serial antara Bridge dan Arduino menggunakan XOR checksum. Pada tahap implementasi, sistem dibangun menggunakan aplikasi dashboard dan bridge berbasis .NET Framework 4.8, firmware Arduino Uno R3, dan broker MQTT Eclipse Mosquitto.



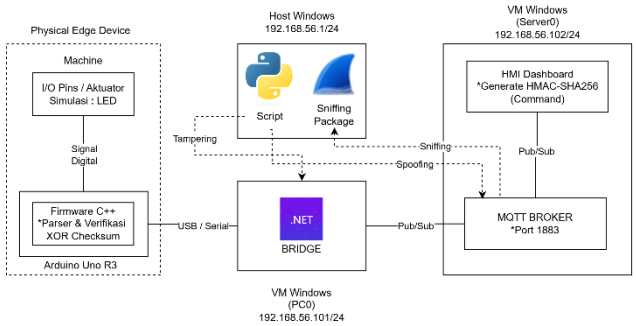
Gambar 1. Diagram Tahapan Penelitian

Tahap pengujian dilakukan pada lingkungan isolated testbed agar hasil eksperimen tidak dipengaruhi oleh lalu lintas jaringan eksternal. Tahap terakhir adalah evaluasi hasil berdasarkan parameter keamanan, performa, dan penggunaan

e-ISSN: 2548-6861

sumber daya, kemudian dibandingkan antara kondisi baseline dan kondisi proposed.

Eksperimen dilakukan pada lingkungan Isolated Testbed yang dipisahkan secara fisik dari jaringan operasional untuk menghindari gangguan terhadap sistem produksi.



Gambar 2. Arsitektur Isolated Testbed.

Testbed menggunakan perangkat yang merepresentasikan sistem aktual, yaitu dashboard web, broker MQTT Eclipse Mosquitto, aplikasi Bridge berbasis .NET Framework 4.8, serta mikrokontroler Arduino Uno R3 sebagai edge node dengan komunikasi serial UART pada baud rate 9600. Komunikasi antara dashboard dan Bridge menggunakan port MQTT 1883 tanpa TLS untuk mensimulasikan kondisi sistem legacy yang masih rentan terhadap pertukaran data plaintext.

C. Lingkungan Pengujian

Pengujian dalam penelitian ini dilakukan di dalam sebuah *Isolated Testbed* (lingkungan pengujian terisolasi) guna memastikan bahwa seluruh data yang diekstraksi valid, terkontrol, dan tidak terpengaruh oleh gangguan lalu lintas jaringan eksternal (network noise). Spesifikasi perangkat keras dan perangkat lunak yang digunakan diatur secara ketat untuk meniru arsitektur komunikasi sistem kontrol industri (ICS) pada umumnya.

Infrastruktur pengujian dibangun menggunakan kombinasi perangkat keras fisik dan mesin virtual (VM) untuk memisahkan peran masing-masing node. Spesifikasi detail perangkat keras yang digunakan pada pengujian ini dirangkum pada Tabel 2.

Tabel 2. Spesifikasi Perangkat Keras Pengujian

Device	Processor	RAM	ROM	Keterangan
Arduino Uno R3	(ATmega328P)	2 KB	32 KB	Bertindak sebagai *edge node* / perangkat industri di lapangan.
Host	AMD Ryzen 5 Pro 4650U (12 CPU)	32 GB	512 GB	Mesin fisik utama yang menjalankan virtualisasi.
Server0 (VM)	2 Core (Virtual vCPU)	4 GB	80 GB	Berjalan di VirtualBox, bertindak sebagai *Bridge / Gateway*.
PC0 (VM)	2 Core (Virtual vCPU)	4 GB	80 GB	Berjalan di VirtualBox, bertindak sebagai *Client / Dashboard*.

Dari sisi perangkat lunak, sistem aplikasi dashboard dan bridge dikembangkan menggunakan kerangka kerja .NET Framework v4.8. Sementara itu, skrip untuk mengeksekusi simulasi serangan dibangun menggunakan bahasa pemrograman Python 3. Pertukaran data antara node dalam infrastruktur virtual ini disalurkan melalui antarmuka jaringan *HostOnly Adapter* dengan rentang alamat IP 192.168.56.0/24. Pendekatan jaringan terisolasi ini mengeliminasi variabel latensi eksternal, sehingga hasil pengukuran waktu respons menjadi sangat akurat.

D. Skenario Kondisi Pengujian

Untuk membuktikan efektivitas sistem keamanan yang diusulkan, eksperimen dilakukan dengan membandingkan dua kondisi arsitektur:

a) **Baseline (sistem legacy),** yang merepresentasikan sistem kontrol industri konvensional dengan pertukaran data JSON dalam bentuk plaintext tanpa enkripsi, autentikasi pesan, maupun validasi checksum, sehingga rentan terhadap manipulasi.

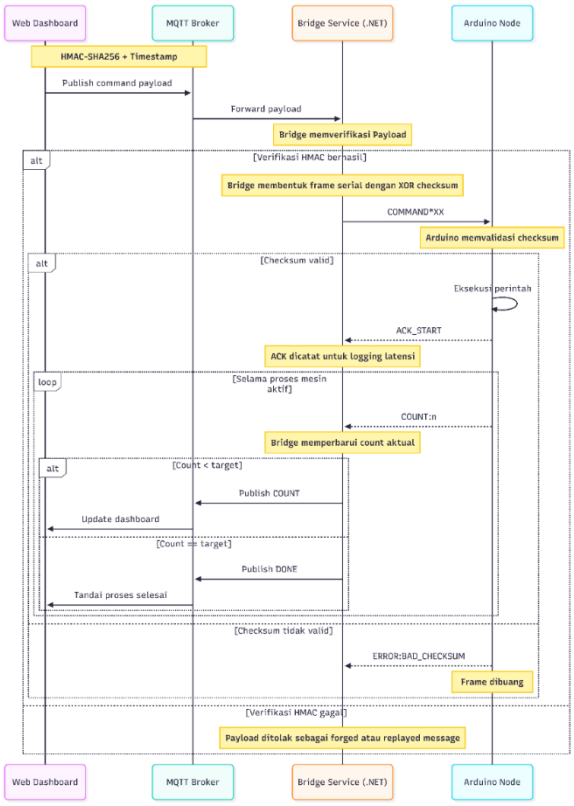
b) **Proposed (sistem usulan),** yaitu arsitektur dengan penguatan lapisan keamanan yang mengimplementasikan algoritma autentikasi pesan HMAC-SHA256 pada payload JSON, dikombinasikan dengan algoritma XOR sebagai checksum dinamis untuk memvalidasi integritas pada level transmisi ringan.

Evaluasi keamanan sistem mencakup tiga skenario pengujian utama: (1) ketahanan terhadap upaya *spoofing*, diuji dengan mengirimkan seribu pesan MQTT tidak sah tanpa HMAC valid ke topik kontrol; pengujian dinyatakan berhasil apabila Bridge menolak pesan tersebut dan mencatat status penolakan pada log keamanan. (2) ketahanan terhadap *tampering*, diuji melalui dua ratus kali modifikasi berkas *state.json*, baik secara manual maupun lewat simulasi *crash recovery*, guna menilai sejauh mana sistem mampu mendeteksi manipulasi pada data yang tersimpan. (3) potensi *false positive*, diuji dengan mengalirkan pesan-pesan sah

secara beruntun, untuk memastikan mekanisme autentikasi tidak keliru menolak pesan yang seharusnya diterima.

Untuk pengukuran performa, dilakukan 1.000 iterasi overhead HMAC pada sisi Bridge, 100 iterasi overhead XOR pada sisi Arduino, 100 iterasi pengukuran latensi end-to-end, serta 30 sampel penggunaan SRAM; jumlah iterasi disesuaikan dengan karakteristik masing-masing objek uji agar hasil pengukuran stabil dan representatif terhadap sifat proses yang dievaluasi.

Alur mekanisme autentikasi dan validasi integritas pada sistem usulan ditunjukkan pada **Gambar 3**.



Gambar 3. Sequence Diagram Mekanisme Autentikasi HMAC-SHA256

Berdasarkan alur tersebut, pengumpulan data dibagi ke dalam dua objek uji utama, yaitu HMAC-SHA256 pada lapisan aplikasi dan XOR checksum pada lapisan serial. Pengujian *spoofing attack* dijalankan melalui skrip otomatis berbasis Python yang memublikasikan pesan-pesan MQTT palsu ke topik kontrol tanpa sertaan HMAC valid, dengan volume pengiriman yang disesuaikan agar mencerminkan kondisi pembanjiran pesan pada skenario nyata; apakah respons Bridge menolak atau justru mengeksekusi, kemudian direkam untuk dianalisis. Pengujian *tampering* dilakukan dengan pendekatan berbeda: berkas state.json dimodifikasi secara manual maupun melalui simulasi *crash recovery* dalam siklus berulang, karena sifatnya yang *stateful*, setiap iterasi menuntut tahap modifikasi diikuti verifikasi secara berurutan, sehingga metodologinya tidak dapat disamakan dengan pendekatan injeksi otomatis pada pengujian *spoofing*.

Kecukupan 200 iterasi ini juga divalidasi secara statistik menggunakan metode *Rule of Three* yang berlaku pada kasus proporsi sempurna (*zero-failure observation*). Dengan seluruh 200 skenario tampering berhasil terdeteksi (p = 1,0), batas bawah Confidence Interval 95% dihitung sebagai:

$$LB\ CI = 1 - \left(\frac{3}{n}\right) = 1\ \left(\frac{3}{200}\right) = 0.985 = 98{,}5\%$$

Hasil ini membuktikan bahwa rasio deteksi sesungguhnya, dengan keyakinan 95%, berada di atas 98,5%, melampaui target minimal 95% yang ditetapkan. Dengan demikian, 200 iterasi terbukti secara statistik cukup untuk memberikan estimasi yang valid dan dapat dipercaya [11]. Pada objek uji XOR checksum, data dikumpulkan melalui 1.000 iterasi injeksi kesalahan pada instruksi serial untuk mengukur konsistensi deteksi error transmisi.

E. Formulasi Kriptografi dan Deteksi Error

Mekanisme pengamanan arsitektur komunikasi pada penelitian ini menerapkan pendekatan pertahanan berlapis (*Defense-in-Depth*) yang memisahkan antara proteksi integritas siber pada jaringan terbuka (*Application-Layer Security*) dan deteksi kesalahan transmisi fisik pada saluran komunikasi serial (*Serial Transmission Integrity*).

E.1 Mekanisme Autentikasi Payload berbasis HMAC-SHA256

Untuk mengamankan integritas payload JSON yang dikirim melalui jalur MQTT antara Dashboard dan Bridge, sistem ini mengadopsi mekanisme otentikasi pesan HMAC-SHA256 sesuai RFC 2104 [8][13], sebagaimana dirumuskan pada Persamaan (1):

$$\boldsymbol{HMAC(K, msg)} = \boldsymbol{H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, msg))} \quad (1)$$

Secara garis besar, proses ini melibatkan fungsi hash SHA-256 (dilambangkan H) yang diterapkan dua kali secara berurutan. Kunci rahasia K: yaitu Pre-Shared Key (PSK) yang disepakati bersama antara Dashboard dan Bridge, terlebih dahulu disesuaikan panjangnya agar sama dengan ukuran blok internal SHA-256, yakni 64 byte, sebelum diproses lebih lanjut. Kunci tersebut kemudian dikombinasikan melalui operasi XOR (dilambangkan $\oplus$) dengan dua konstanta byte statis yang telah baku ditetapkan dalam RFC 2104, yaitu ipad untuk lapisan hashing bagian dalam dan opad untuk lapisan hashing bagian luar.

Pada tahap pertama, hasil XOR antara K dan ipad digabungkan (dilambangkan dengan operator ||) dengan data payload asli yang hendak dikirim (msg), lalu keseluruhan

e-ISSN: 2548-6861

gabungan ini di-hash menggunakan SHA-256. Hasil hash tahap pertama tersebut kemudian digabungkan kembali dengan hasil XOR antara K dan opad, dan seluruh rangkaian ini di-hash sekali lagi untuk menghasilkan nilai HMAC akhir. Struktur dua lapis inilah yang membuat HMAC tahan terhadap serangan ekstensi panjang (length-extension attack) yang menjadi kelemahan pada penggunaan hash tunggal biasa.

E.2 Deteksi Kesalahan Transmisi Serial berbasis XOR-8 Checksum

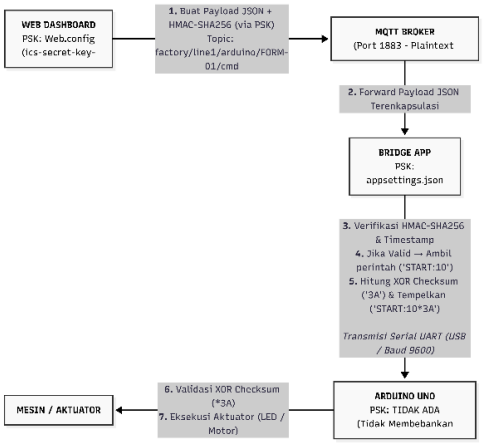
Untuk jalur komunikasi serial UART antara Bridge dan mikrokontroler Arduino Uno, validasi integritas data fisik menggunakan operasi XOR-8 Checksum linier analog dengan mekanisme checksum yang digunakan pada NMEA 0183 (IEC 61162-1). Nilai checksum dihitung melalui Persamaan (2):

$$\boldsymbol{C = b_1 \oplus b_2 \oplus \cdots \oplus b_n} \qquad (2)$$

Dengan b_i menyatakan byte ASCII ke-*i* dari string perintah dan C adalah nilai checksum 8-bit. Aplikasi Bridge mengkalkulasi Persamaan (2) terhadap perintah yang telah lolos verifikasi HMAC, kemudian menyisipkan nilai C sebagai dua karakter heksadesimal di akhir string setelah pembatas asterisk (format: *PERINTAH*XX*, contoh: "*START:10*3A*"). Firmware Arduino mengeksekusi fungsi validateChecksum() untuk memverifikasi ulang nilai C; jika hasil XOR tidak sesuai akibat gangguan kebisingan elektromagnetik (noise) pada kabel serial, instruksi dianggap rusak dan diabaikan.

E.3 Arsitektur Pertahanan Berlapis dan Pemetaan Penyimpanan PSK

Kriptografi berbasis PSK diterapkan secara bilateral hanya antara Dashboard dan Bridge, sedangkan Arduino Uno beroperasi tanpa menyimpan kunci. Pemetaan lokasi penyimpanan PSK dirangkum dalam alur kerjanya yang diilustrasikan pada Gambar 4.



Gambar 4. Diagram Alur Pertahanan Berlapis Dan Distribusi Kunci

F. Instrumen dan Parameter Pengukuran

Pengukuran dalam penelitian ini difokuskan pada tiga aspek utama, yaitu efektivitas keamanan, overhead komputasi, dan efisiensi penggunaan sumber daya. Untuk memperoleh data yang terukur dan dapat dibandingkan secara langsung, setiap parameter dievaluasi menggunakan instrumen yang disesuaikan dengan karakteristik objek uji.

Pengukuran latensi overhead HMAC-SHA256 pada sisi Bridge dilakukan menggunakan *System.Diagnostics.Stopwatch* pada lingkungan *.NET*. Parameter ini merepresentasikan waktu murni yang dibutuhkan untuk menghitung dan memverifikasi nilai HMAC terhadap payload MQTT yang diterima. Sementara itu, pengukuran latensi *overhead XOR checksum* pada sisi Arduino dilakukan menggunakan fungsi *micros()* pada firmware untuk memperoleh resolusi waktu dalam satuan mikrodetik.

Latensi end-to-end diukur sebagai total waktu sejak perintah dikirim dari dashboard hingga respons dari perangkat fisik diterima kembali melalui Bridge. Pengukuran ini dilakukan menggunakan pencatatan timestamp pada aplikasi untuk menangkap dampak akumulatif dari proses autentikasi, transmisi, dan eksekusi instruksi. Selain itu, konsumsi SRAM pada Arduino diukur untuk menilai kelayakan implementasi pada perangkat dengan sumber daya terbatas.

Rincian instrumen dan parameter pengukuran ditunjukkan pada Tabel 3.

Tabel 3. Instrumen dan Parameter Pengukuran

Jenis Latensi	Definisi	Instrumen	Iterasi

Latensi Overhead HMAC	Waktu murni komputasi HMAC-SHA256 pada Bridge (ms)	System.Diagnostics.Stopwatch (.NET)	1.000
Latensi Overhead XOR	Waktu murni komputasi XOR Checksum pada Arduino (µs)	Fungsi micros() (firmware)	100
Latensi End-to-end	Total waktu dari klik tombol Dashboard hingga mesin merespons (ms)	Timestamp aplikasi	100

Untuk evaluasi keamanan, indikator yang digunakan meliputi rasio blokir terhadap serangan spoofing, rasio deteksi terhadap manipulasi state.json, dan tingkat false positive pada pengiriman pesan valid. Sementara itu, pada aspek efisiensi implementasi, evaluasi difokuskan pada latensi tambahan yang ditimbulkan oleh mekanisme keamanan serta konsumsi SRAM tambahan pada perangkat edge. Definisi operasional dari masing-masing indikator evaluasi dirangkum pada Tabel 4.

Tabel 4. Definisi Operasional dan Instrumen Pengukuran

Indikator Evaluasi	**Objek Uji**	**Kondisi AsIs (Baseline)**	**Target**
Rasio Blokir Spoofing Attack	HMAC-SHA256	0% — Seluruh pesan MQTT palsu dieksekusi tanpa validasi.	≥ 95% pesan tidak sah ditolak Bridge
Rasio Deteksi Tampering pada state.json	HMAC-SHA256	0% — Modifikasi tidak terdeteksi; data palsu dimuat saat restart.	≥ 95% skenario manipulasi terdeteksi
Latensi Overhead HMAC (Bridge .NET)	HMAC-SHA256	N/A — Tidak ada komputasi keamanan (0 ms).	Rata-rata ≤ 200 ms; tidak ada outlier kritis
Latensi Overhead XOR (Arduino)	XOR Checksum	N/A — Instruksi serial diproses tanpa validasi (passthrough).	≤ 5 ms per instruksi serial.
Konsumsi SRAM Tambahan (Arduino)	XOR Checksum	N/A — Ukuran runtime baseline firmware tanpa validasi.	Tambahan SRAM ≤ 200 bytes

G. Analisis Data

Analisis data dilakukan secara komparatif antara kondisi baseline dan kondisi proposed untuk menilai pengaruh penerapan mekanisme keamanan terhadap efektivitas proteksi dan performa sistem. Pendekatan ini digunakan agar perubahan yang muncul dapat diinterpretasikan secara langsung sebagai konsekuensi dari penambahan autentikasi HMAC-SHA256 dan validasi *checksum* XOR.

Untuk data performa, analisis dilakukan menggunakan statistik deskriptif yang meliputi nilai rata-rata, median, standar deviasi, minimum, dan maksimum. Statistik ini digunakan untuk menggambarkan kecenderungan pusat dan sebaran data pada pengukuran latensi overhead HMAC, latensi overhead XOR, dan latensi end-to-end. Dengan demikian, evaluasi tidak hanya berfokus pada nilai rata-rata, tetapi juga mempertimbangkan kestabilan distribusi hasil pengukuran.

Untuk data keamanan yang menghasilkan proporsi sempurna (*zero-failure observation*), estimasi batas bawah CI 95% untuk data proporsi sempurna dihitung menggunakan metode *Rule of Three* sebagaimana dijelaskan pada subbab sebelumnya. Melalui pendekatan ini, tingkat kepercayaan terhadap hasil deteksi atau pemblokiran sempurna tetap dapat dinyatakan secara statistik.

Efektivitas keamanan sistem dievaluasi melalui tiga indikator utama, yaitu rasio blokir terhadap serangan spoofing, rasio deteksi terhadap tampering pada state.json, dan tingkat false positive pada pesan valid. Sementara itu, kelayakan implementasi pada perangkat dengan keterbatasan sumber daya dievaluasi melalui konsumsi SRAM tambahan dan besarnya overhead komputasi yang ditimbulkan. Evaluasi akhir tidak hanya mempertimbangkan keberhasilan proteksi, tetapi juga memastikan bahwa mekanisme keamanan tidak mengganggu operasi normal sistem kontrol industri.

III. Hasil dan Pembahasan

3.1 Skenario dan Parameter Pengujian

Pengujian dijalankan pada dua kondisi arsitektur, *baseline* dan *proposed*, dengan jumlah iterasi yang disesuaikan terhadap karakteristik masing-masing skenario agar setiap aspek dapat diamati secara proporsional. Rincian pembagian iterasi ditetapkan sebagai berikut:

- **1.000 iterasi** pengiriman pesan ilegal, digunakan untuk mengevaluasi ketahanan jalur komunikasi MQTT terhadap serangan *spoofing*.
- **200 iterasi** injeksi muatan, digunakan untuk mensimulasikan skenario manipulasi data (*tampering*).
- **1.000 iterasi** pengiriman pesan valid, digunakan untuk memantau pergeseran latensi transmisi *end-to-end* secara komparatif antar-arsitektur.
- **30 sampel konstan** digunakan untuk mengekstraksi jejak utilisasi memori (SRAM) pada mikrokontroler.

Pembagian iterasi tersebut disusun agar selaras dengan karakteristik masing-masing objek uji. Skenario spoofing dan

e-ISSN: 2548-6861

pengiriman pesan valid memerlukan jumlah iterasi yang lebih besar karena melibatkan proses komunikasi berulang yang relatif ringan, sedangkan skenario tampering dan pengukuran SRAM memerlukan pendekatan yang lebih terbatas karena sifat pengujian yang stateful dan berfokus pada kondisi sistem tertentu.

3.2 Evaluasi Keamanan Sistem

Pengujian pada lapisan keamanan bertujuan untuk memvalidasi seberapa tangguh arsitektur yang diusulkan dalam menahan berbagai vektor serangan siber industri, sekaligus memastikan perlindungan tersebut tidak mengganggu ketersediaan layanan (availability) dari data yang sah. Evaluasi ini diukur melalui tiga skenario utama: serangan spoofing, manipulasi data (tampering), dan pengujian false positive. Rangkuman komparasi hasil pengujian keamanan antara sistem baseline dan proposed dapat dilihat pada Tabel 5.

Tabel 5. Rangkuman Hasil Pengujian Keamanan Sistem

Skenario Pengujian	**Jumlah Iterasi**	**Kondisi Baseline (Sistem Legacy)**	**Kondisi Proposed (Sistem Usulan)**	**Tindakan / Mekanisme Sistem**
Spoofing	1.000	0% Diblokir (Aktuator bergerak liar)	**100% Diblokir** (*Block Rate* Penuh)	*Gateway* (Bridge) menolak pesan dan mencatat HMAC Invalid/Missing pada *alert log*.
Manipulasi Data (*Tampering*)	200	0% Terdeteksi (Muatan palsu dieksekusi)	**100% Terdeteksi** & STATE_REJECTED	*Silent Rejection*; menahan propagasi instruksi palsu dan mempertahankan *state* aman terakhir.
False Positive	1.000	N/A (Tidak ada sistem verifikasi)	**0% Pesan Valid Terblokir**	Pesan valid beruntun diteruskan dengan sukses tanpa memicu kegagalan *timeout*.

3.2.1 Ketahanan terhadap Serangan Spoofing

Dari seluruh percobaan *spoofing* yang dilakukan, sistem *Baseline* terbukti sama sekali tidak mampu membendung serangan: seluruh muatan *plaintext* berbahaya (100%) berhasil menembus gateway dan diteruskan langsung ke perangkat *edge* (Arduino) akibat absennya mekanisme autentikasi. Konsekuensi dari lolosnya instruksi ilegal ini bersifat fatal dalam konteks operasional fisik—aktuator seperti motor atau mesin berpotensi menerima perintah acak dan bergerak tidak terkendali, misalnya melakukan *restart* paksa berulang dari nilai 0 hingga 999.

Sebaliknya, pada implementasi sistem yang diusulkan (*Proposed*), mekanisme keamanan terbukti memberikan proteksi absolut dengan tingkat pemblokiran (*Block Rate*) mencapai 100%. Setiap muatan pesan *spoofing* yang masuk berhasil ditahan pada lapisan aplikasi di *Bridge*. Sistem tidak hanya sekadar membuang (*drop*) pesan tak berizin tersebut, tetapi juga memiliki kapabilitas visibilitas ancaman yang baik dengan mencatat aktivitas tersebut ke dalam *alert log* dengan status penolakan *"HMAC Invalid/Missing"*. Hal ini mencegah instruksi berbahaya menjangkau *controller* industri fisik.

3.2.2. Deteksi Manipulasi Data (Tampering)

Pada seluruh skenario manipulasi data mencakup penyuntikan nilai pada atribut krusial seperti *TargetQuantity*, *CurrentCount*, dan *WorkOrder*, sistem usulan secara konsisten berhasil mendeteksi ketidaksesuaian nilai *hash* pada setiap upaya serangan, dan langsung mengubah status muatan menjadi *STATE_REJECTED*.

Berdasarkan temuan tersebut, dapat disimpulkan bahwa mekanisme keamanan sistem bertindak sebagai *"Fail-Safe / Silent Defender"*. Ketika terjadi *tampering*, mekanisme kriptografi bekerja secara reaktif memblokir propagasi data yang dimanipulasi pada *layer gateway* (Bridge), sehingga berhasil melindungi *controller* (Arduino) dari instruksi operasional yang tidak sah. Lebih jauh lagi, sistem melakukan tindakan protektif ini tanpa membebani trafik jaringan internal dengan rentetan respons *error* (*Silent Rejection*). Sebagai hasilnya, perangkat keras di lapangan tetap beroperasi secara stabil dengan mempertahankan *state* (status) operasional terakhir yang terkonfirmasi aman.

3.2.3 Analisis Positif Palsu (False Positive)

Selain memastikan bahwa ancaman diblokir, sistem keamanan dinilai fungsional jika tidak mengganggu aliran data yang sah. Hasil pengujian *False Positive* dengan 1.000 iterasi pesan valid secara intensif (dengan jeda 50 ms) menghasilkan 0% *Block Rate*. Hal ini tidak hanya membuktikan keandalan logika verifikasi HMAC, tetapi juga mengonfirmasi efisiensi arsitektur komputasi secara keseluruhan.

Keberhasilan pengiriman transmisi beruntun tanpa adanya *drop payload* mengindikasikan dua hal krusial terkait *overhead* waktu komputasi: Pertama, proses kalkulasi algoritma HMAC-SHA256 pada *Bridge* (berbasis C# .NET) terbukti cukup cepat sehingga tidak menyebabkan fenomena *bottleneck* pada antrean pesan. Jika komputasi berjalan lambat, tumpukan antrean pesan berisiko kedaluwarsa dan

akan tertolak secara otomatis oleh mekanisme *Freshness Check* (*Replay Window* 30 detik berdasarkan standar RFC 6238) milik sistem itu sendiri.

Kedua, proses dekripsi algoritma XOR pada level mikrokontroler (Arduino) berjalan sangat ringan. Karakteristik ini memastikan Arduino mampu memproses seluruh bagian *payload* dan merespons melalui antarmuka komunikasi serial secara instan. Kesigapan respons ini sangat penting untuk menghindari kegagalan sinkronisasi atau pemutusan komunikasi yang dapat diakibatkan oleh terlampauinya batas *Serial ReadTimeout* (dengan batas waktu 1 detik) pada sisi *Bridge*. Rincian kuantitatif mengenai profil latensi spesifik dari penambahan komputasi HMAC dan XOR ini akan dibahas lebih mendalam pada subbab terkait Evaluasi Overhead di bagian berikutnya.

3.3 Evaluasi Performance & Resource Analysis

Sebuah sistem keamanan kontrol industri yang ideal tidak hanya harus tangguh menahan serangan siber, tetapi juga harus mampu mempertahankan ketersediaan sistem (*availability*) tanpa membebani perangkat secara berlebihan. Pada subbab ini, evaluasi difokuskan pada pengukuran *overhead* waktu komputasi dari penambahan lapisan kriptografi, pergeseran latensi transmisi *end-to-end*, serta jejak utilisasi memori (SRAM) pada perangkat *edge* mikrokontroler.

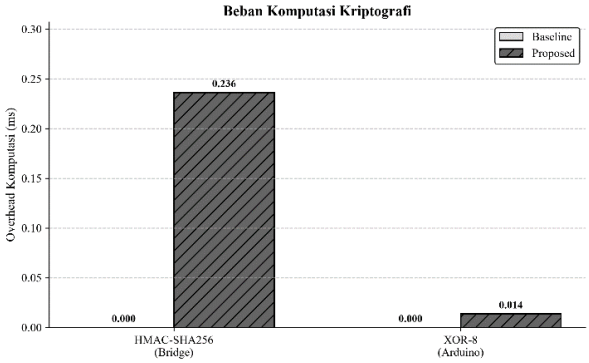
Tabel 6. Statistik Deskriptif Performa dan Overhead Sistem

Metrik	n	Mean	Median	Standar Deviasi	Min.	Maks.
Overhead HMAC-SHA256 (Bridge)	1.000	0,236 ms	0,070 ms	1,558 ms	0,050 ms	25,257 ms
Overhead XOR-8 Checksum (Arduino)	100	13,720 µs	12,000 µs	2,070 µs	12,000 µs	20,000 µs
Latensi End-to-End Proposed	1.000	114,07 ms	110,94 ms	22,74 ms	75,59 ms	441,26 ms
Latensi End-to-End Baseline	100	80,37 ms	74,55 ms	30,00 ms	44,08 ms	250,36 ms

**Catatan: Data baseline diekstraksi secara spesifik dari respons operasional murni (ACK_START) untuk menjamin ekuivalensi komparasi terhadap kondisi proposed.*

3.3.1 Analisis Overhead Kriptografi

Penambahan mekanisme keamanan pada arsitektur yang diusulkan melibatkan dua operasi komputasi utama: kalkulasi nilai *hash* HMAC-SHA256 pada lapisan *Gateway* (Bridge) dan operasi dekripsi XOR pada lapisan *Edge* (Arduino). Distribusi waktu komputasi dari kedua operasi ini divisualisasikan pada **Gambar 3.1**



Gambar 3.1 Kriptografi Overhead

Berdasarkan ekstraksi data statistik deskriptif pada Tabel 6, penambahan mekanisme keamanan pada arsitektur terdistribusi ini melibatkan dua operasi utama: kalkulasi HMAC-SHA256 pada lapisan Gateway (Bridge) dan dekripsi XOR pada lapisan Edge (Arduino).

Proses kalkulasi verifikasi HMAC murni pada CutAndFormBridge yang berjalan di lingkungan .NET C# mencatatkan nilai rata-rata (mean) sebesar 0,236 ms. Secara empiris, performa waktu komputasi ini sangat impresif dengan nilai tengah (median) hanya sebesar 0,070 ms. Konsistensi waktu pemrosesan yang berada jauh di bawah angka 1 ms ini secara praktis meniadakan risiko terjadinya bottleneck pada antrean pesan di sisi server. Adapun munculnya nilai maksimum 25,257 ms dan standar deviasi sebesar 1,558 ms lebih disebabkan oleh anomali sporadis akibat siklus pembersihan memori (Garbage Collection) pada runtime .NET dan tidak merepresentasikan performa operasional mayoritas.

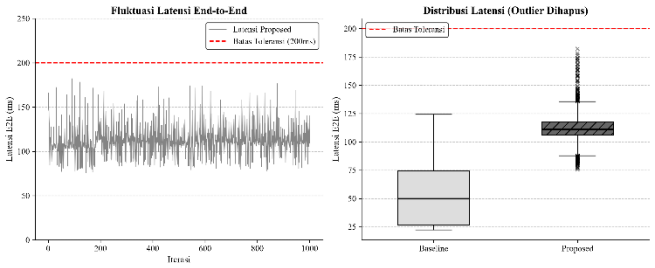
Di sisi lain, proses dekripsi XOR yang dibebankan pada mikrokontroler Arduino Uno mencatatkan beban waktu (overhead) rata-rata sebesar 13,72 µs dengan nilai maksimum hanya 20 µs. Skala komputasi pada level mikrosekon ini membuktikan bahwa algoritma XOR sangat ringan dan tidak menghambat siklus pemrosesan instruksi aktuator pada controller fisik.

3.3.2 Dampak terhadap Latensi End-to-end

Metrik kinerja yang paling krusial dalam lingkungan *Industrial Control System* (ICS) adalah latensi komunikasi *end-to-end*, yaitu total waktu yang dibutuhkan sejak pesan dikirim oleh *Dashboard* hingga respons dari perangkat keras (*actuator*) diterima kembali melalui *Bridge*. Komparasi

e-ISSN: 2548-6861

distribusi latensi antara kondisi *Baseline* dan *Proposed* diilustrasikan pada **Gambar 3.2**



Gambar 3.2 Latency End to End

Merujuk pada **Tabel 6**, hasil pengujian sampel iteratif menunjukkan bahwa sistem usulan (*Proposed*) mencatatkan waktu latensi E2E rata-rata (*mean*) sebesar **114,07 ms** (median: 110,94 ms), sedangkan kondisi sistem lama (*Baseline*) mencatatkan rata-rata sebesar **80,37 ms** (median: 74,55 ms). Terdapat selisih (*delta*) latensi absolut sebesar **+33,70 ms** (+42,0%). Catatan metodologis penting yang perlu ditegaskan adalah bahwa data *Baseline* yang digunakan dalam komparasi ini telah disaring secara eksplisit hanya pada respons operasional *ACK_START* untuk menjamin kesetaraan jenis operasi (*apple-to-apple*) terhadap kondisi *Proposed*.

Untuk mengevaluasi sumber utama dari penambahan latensi sebesar +33,70 ms tersebut, dilakukan analisis dekomposisi antara komputasi murni per-*isolated micro-benchmarks* dan latensi sistem secara makro. Hasil pengukuran independen menunjukkan bahwa waktu komputasi murni HMAC-SHA256 pada lapisan *Bridge* (0,236 ms) dan validasi XOR-8 Checksum pada Arduino (0,0137 ms) secara akumulatif hanya menyumbang sebesar **0,25 ms atau sekitar 0,74%** dari total penambahan latensi E2E.

Porsi terbesar dari penambahan latensi, yakni sebesar **33,45 ms (99,26%)**, diakibatkan oleh faktor-faktor overhead non-komputasi arsitektural. Faktor-faktor tersebut meliputi: (1) pembesaran ukuran muatan data (*payload*) JSON akibat penambahan atribut tanda tangan kriptografi dan *timestamp*, (2) overhead waktu transmisi fisik pada antarmuka serial UART (baud rate 9600 bps), (3) proses enkapsulasi dan serialisasi/deserialisasi objek JSON di sisi aplikasi, serta (4) pemrosesan antrean *stack* pada broker MQTT. Temuan ini membuktikan bahwa penambahan fungsi enkripsi dan validasi kriptografi itu sendiri tidak menjadi beban utama dalam jalur komunikasi.

Di sisi lain, analisis sebaran data mengindikasikan bahwa latensi sistem Proposed memiliki karakteristik yang lebih stabil dibandingkan Baseline. Hal ini tercermin dari nilai standar deviasi yang lebih rendah pada sistem Proposed (22,74 ms) dibandingkan Baseline (30,00 ms), yang mengindikasikan variansi latensi transmisi yang lebih terkontrol dan konsisten antar-percobaan [11]. Secara keseluruhan, latensi rata-rata total sistem usulan (~114 ms) berada jauh di bawah ambang toleransi latensi yang ditetapkan pada pengujian ini, yaitu 200 ms, sebuah batasan yang merujuk pada pertimbangan keandalan waktu-nyata dalam operasional sistem kendali industri [13]. Temuan ini memperkuat indikasi bahwa arsitektur keamanan terdistribusi yang diusulkan mampu menyeimbangkan antara penguatan proteksi siber (*security*) dan pemeliharaan ketersediaan kendali mesin secara waktu-nyata (*availability*), sejalan dengan tantangan keseimbangan *security–availability* yang umum dihadapi pada sistem *cyber-physical* [10].

3.3.3 Utilisasi Memori (SRAM) pada Arduino

Evaluasi konsumsi memori pada perangkat edge node (Arduino Uno / ATmega328P) dilakukan menggunakan pendekatan *static source code analysis* melalui skrip profiling otomatis yang membandingkan *binary output* dari kedua versi *firmware* secara *byte-demi-byte*. Tabel 3.4 merangkum hasil perbandingan konsumsi memori antara kondisi *Baseline* dan *Proposed* secara komprehensif.

Tabel 7. Perbandingan Konsumsi Memori Firmware Arduino

Region Memori	Baseline	Proposed	Delta	Deskripsi
Flash ROM Terpakai	4.472 bytes	4.642 bytes	+170 bytes	Kode mesin fungsi validateChecksum(), instrumentasi micros(), dan string literal
Flash ROM Sisa	27.784 bytes	27.614 bytes	-170 bytes	Kapasitas program yang masih tersedia
Utilisasi Flash (%)	13,86%	14,39%	+0,53%	Dari total kapasitas 32.256 bytes
SRAM Statis	319 bytes	319 bytes	0 bytes	Variabel global identik di kedua versi
SRAM Boot	1.729 bytes	1.729 bytes	0 bytes	Nilai freeMemory() di setup()
Stack Transien	0 bytes	20 bytes	+20 bytes	Alokasi sementara saat validateChecksum() dieksekusi

SRAM Bebas saat Eksekusi	1.729 bytes	1.709 bytes	-20 bytes	Headroom minimum saat pesan divalidasi

Overhead ini terdiri dari: (1) kode *assembly* fungsi *validateChecksum()* sebesar 141 bytes (82,94%), (2) instrumentasi pengukuran timing menggunakan *micros()* sebesar 26 bytes (15,29%), dan (3) selisih bersih string literal *PROGMEM* sebesar 3 bytes (1,77%). Dalam konteks total kapasitas *Flash ATmega328P* (32.256 bytes), penambahan 170 bytes ini hanya merepresentasikan peningkatan utilisasi Flash sebesar +0,53%, suatu beban yang secara praktis dapat diabaikan.

Kondisi *Proposed* mendefinisikan set variabel global yang identik dengan kondisi *Baseline*, sehingga segmen ***.data*** dan ***.bss*** kedua versi sama-sama berukuran 319 *bytes*. Hal ini menjelaskan mengapa pengukuran *freeMemory()* yang dipanggil di dalam fungsi *setup()* selalu menghasilkan nilai yang sama (1.729 *bytes*) pada kedua kondisi. Fungsi *validateChecksum()* dieksekusi secara eksklusif di dalam *loop()*, yaitu setelah *setup()* selesai, sehingga alokasi stack-nya tidak terukur oleh metode pengukuran *boot-time* tersebut.

Analisis statis terhadap struktur stack frame pada fungsi *validateChecksum()* menunjukkan bahwa alokasi memori lokal tersebut hanya berlangsung secara transaksional selama fungsi aktif dieksekusi pada rentang waktu komputasi (*runtime*). Komposisi stack frame tersebut meliputi variabel penunjuk (dua penunjuk *star* dan *hexStr* masing-masing berukuran 2 *byte*, serta penunjuk iterator p berukuran 2 *byte*), variabel indeks perulangan (i berukuran 2 *byte*), variabel penampung nilai *checksum* (calc dan expected masing-masing berukuran 1 byte), serta variabel manipulasi karakter sementara (*ch* dan *nibble* masing-masing berukuran 1 byte). Ditambah dengan beban *overhead* berupa simpanan alamat kembalian (*return address*), *frame pointer*, *register internal*, dan penyelarasan instruksi (*compiler padding*) sebesar 8 *byte*, total alokasi puncak pada area stack mencapai 20 *byte*.

Pada sisi efisiensi memori, seluruh variabel yang digunakan dalam proses kriptografi dirancang bersifat lokal dengan lingkup terbatas (block scope), sehingga ruang memori yang dialokasikan untuknya otomatis dibebaskan (deallocated) begitu eksekusi fungsi selesai. Dengan pendekatan ini, margin memori bebas (SRAM headroom) pada kondisi beban kerja puncak tetap terjaga pada angka 1.709 byte, atau setara 83,45% dari total kapasitas SRAM ATmega328P sebesar 2.048 byte. Capaian ini berada jauh di atas ambang minimum yang digunakan sebagai acuan kelayakan pada penelitian ini, yaitu 20% dari total kapasitas SRAM, batas konservatif yang lazim diterapkan pada perangkat mikrokontroler AVR 8-bit untuk mengantisipasi fragmentasi memori dan kebutuhan proses tambahan di masa mendatang.

Secara keseluruhan, temuan ini mengonfirmasi kelayakan arsitektur keamanan yang diusulkan. Pendelegasian komputasi kriptografi berat (HMAC-SHA256) ke lapisan gateway (bridge) serta pembatasan beban perangkat edge hanya pada fungsi validasi integritas ringan (XOR *Checksum*) terbukti tidak menimbulkan akumulasi beban memori statis (0 bytes overhead SRAM) dan hanya mengonsumsi 0,53% dari total kapasitas flash ROM yang tersedia.

3.4 Pembahasan Ekstensif (Tradeoff Analysis)

Evaluasi secara menyeluruh terhadap arsitektur keamanan yang diusulkan tidak hanya berfokus pada keberhasilan teknis dalam skala laboratorium, tetapi juga pada bagaimana sistem ini diposisikan di dalam konteks standar industri dan kelayakan praktis. Pembahasan ekstensif ini mensintesis tiga aspek utama: batasan ruang lingkup triade keamanan, komparasi beban sumber daya terhadap protokol standar, dan nilai kontribusi praktis arsitektur bagi industri skala menengah.

3.4.1 Keseimbangan Postur Keamanan (AIA vs CIA)

Dalam merancang sistem pertahanan siber untuk lingkungan dengan sumber daya sangat terbatas (resource-constrained), kejujuran dalam memetakan kapabilitas sistem adalah hal yang esensial. Meskipun triade CIA (Confidentiality, Integrity dan Availability) merupakan standar emas dalam keamanan informasi [6][10], arsitektur yang diusulkan pada penelitian ini dirancang secara spesifik untuk memberikan jaminan AIA (Authentication, Integrity, dan Availability).

Sistem ini secara sadar tidak menerapkan perlindungan kerahasiaan (*Confidentiality*). Penggunaan algoritma XOR pada mikrokontroler diimplementasikan secara murni sebagai operasi **XOR* Checksum** linier untuk validasi integritas (*error-detecting code*), bukan sebagai *cipher* atau blok enkripsi. Hal ini berarti muatan pesan (*payload*) dasar tetap ditransmisikan dalam format *plaintext*. Keputusan ini diambil karena implementasi standar enkripsi yang aman (seperti AES atau ChaCha20) secara persisten akan melampaui sisa kapasitas 2 KB SRAM pada mikrokontroler kelas bawah (seperti Arduino Uno). Dengan demikian, penambahan fungsionalitas enkripsi penuh (*Confidentiality*) diidentifikasi sebagai pengembangan di masa depan (*future work*) yang memerlukan transisi ke modul perangkat keras dengan kapabilitas memori yang lebih besar, seperti ESP32 atau arsitektur ARM 32-bit.

3.4.2 Komparasi terhadap Protokol Standar Industri (TLS/SSL)

e-ISSN: 2548-6861

Secara konseptual, sistem arsitektur kontrol industri idealnya dilindungi oleh protokol keamanan standar seperti TLS 1.3. Namun, memaksakan standar tersebut pada perangkat keras 8-bit merupakan pendekatan yang tidak layak secara fundamental. Penerapan TLS membutuhkan proses handshake yang wajib mengalokasikan memori untuk pertukaran sertifikat, pembentukan buffer kriptografi, dan penyimpanan konteks sesi secara bersamaan. Sebagai perbandingan kuantitatif, pustaka mbedTLS yang dikonfigurasi pada titik paling minimal sekalipun tetap membutuhkan sekitar 20 hingga 50 KB RAM secara dinamis [7].

Kebutuhan tersebut secara teknis akan langsung memicu fenomena *immediate heap exhaustion* (sistem mati mendadak karena kehabisan memori) pada Arduino Uno yang hanya memiliki total ketersediaan 2.048 *byte* SRAM. Sebaliknya, arsitektur inovatif yang diusulkan pada penelitian ini memecahkan masalah tersebut melalui strategi *HMAC offloading*. Dengan membebankan seluruh kalkulasi beban berat HMAC-SHA256 pada lapisan *bridge* (sistem C# berbasis PC yang tidak memiliki batasan memori restriktif), mikrokontroler di lapangan hanya perlu mengeksekusi *XOR8 Checksum*. Seperti yang telah dibuktikan pada pengujian sebelumnya, pendekatan ini terbukti hanya mengonsumsi beban *overhead* SRAM sebesar nol byte persisten dan tambahan waktu komputasi rata-rata yang sangat ringan, yakni 13,72 µs.

3.4.3 Implikasi Praktis: Solusi Retrofitting Berbiaya Rendah

Kontribusi paling signifikan dari desain keamanan yang diusulkan terletak pada kesesuaiannya dengan kebutuhan implementasi di lapangan. Arsitektur ini dibangun di atas prinsip *backward-compatible security layer* suatu lapisan keamanan yang dapat disematkan pada infrastruktur *legacy* tanpa perlu mengganggu arsitektur fisik yang sudah berjalan. Pendekatan semacam ini menjadi krusial khususnya bagi pelaku UKM manufaktur, mengingat opsi migrasi ke PLC modern berdukungan kriptografi bawaan seperti Siemens S7-1500 atau perangkat berbasis IEC 62443 yang umumnya berada di luar jangkauan finansial mereka. Kesenjangan ini tergambar jelas dari data pasar: biaya kepatuhan bagi UKM mencapai ratusan ribu dolar AS, sangat kontras dengan perusahaan multinasional yang mampu mengalokasikan belasan juta dolar [14]. Ketimpangan sumber daya semacam ini turut menjadi kendala nyata yang dihadapi UKM dalam melakukan *retrofit* keamanan pada infrastruktur lama mereka [15], sekaligus tantangan yang secara luas telah disorot dalam literatur keamanan ICS dan adopsi teknologi digital [16].

Melalui model penempatan logika *Freshness Check* dan HMAC-SHA256 terpusat pada Bridge (berjalan pada PC industri eksisting), mesin-mesin legacy tetap dapat dipertahankan sekaligus memperoleh kemampuan membedakan instruksi komando yang sah dari yang termanipulasi. Pendekatan ini sejalan dengan tren riset retrofit ICS kontemporer seperti RePeL, yang menyisipkan data autentikasi ke field protokol tak terpakai dan dapat dilimpahkan ke perangkat *bump-in-the-wire* untuk mengurangi beban komputasi perangkat lapangan [17]. Dengan demikian, fasilitas manufaktur dapat bertransformasi dari lingkungan *zero-security* menuju ekosistem yang terautentikasi dan integritasnya tervalidasi tanpa mengorbankan anggaran penggantian perangkat keras.

Trade-off berupa penambahan latensi ~104 ms perlu dinilai secara proporsional. Sebagai pembanding, sejumlah solusi *bump-in-the-wire* sejenis melaporkan overhead jauh lebih kecil F2-Pro di bawah 1 ms per hop [D], ACRIC jauh di bawah 1 ms [18], dan kerangka autentikasi inline IEC 61850 di bawah 3 ms [19], selisih yang wajar mengingat arsitektur ini memproses verifikasi secara terpusat di PC industri, bukan pada perangkat *embedded* khusus. CISA sendiri mencatat bahwa retrofit enkripsi pada protokol legacy secara umum berisiko menambah latensi dan memerlukan pengujian menyeluruh agar tidak melanggar parameter operasional [20]. Untuk kelas proses manufaktur diskrit yang umumnya bertoleransi terhadap siklus kontrol puluhan hingga ratusan milidetik, overhead 104 ms tetap berada dalam batas yang dapat diterima secara operasional dan menjadi kompensasi rasional untuk memperoleh proteksi autentikasi serta integritas komando tingkat industri tanpa biaya penggantian perangkat keras.

IV. Kesimpulan

Penelitian ini berhasil menjawab tantangan mengenai risiko keamanan siber berupa kerentanan protokol MQTT pada arsitektur Industrial Control System (ICS) berbiaya rendah yang pada kondisi standar mentransmisikan data operasional dalam bentuk plaintext pada port 1883. Sebagai solusi atas masalah tersebut, sebuah arsitektur pertahanan berlapis telah diimplementasikan tanpa mengubah spesifikasi perangkat keras utama. Solusi ini mengintegrasikan mekanisme autentikasi menggunakan algoritma HMAC-SHA256 pada lapisan aplikasi yang dikombinasikan secara hibrida dengan metode validasi ringan berupa operasi XOR *checksum* pada jalur komunikasi transmisi serial menuju perangkat lapangan.

Hasil evaluasi eksperimental secara kuantitatif membuktikan keandalan arsitektur keamanan yang diusulkan melalui parameter metrik yang presisi. Pada aspek pengujian keamanan siber, mekanisme yang diterapkan sukses mencapai rasio pemblokiran penuh sebesar 100% terhadap rentetan serangan injeksi pesan (*spoofing*) ilegal serta memberikan tingkat deteksi manipulasi (*tampering*) data operasional sebesar 100% pada pemantauan berkas status

(*state file*). Sementara itu, hasil pengujian kinerja menegaskan efisiensi arsitektur keamanan terdistribusi yang diusulkan. Beban komputasi kriptografi yang berat berhasil dialihkan (offloaded) sepenuhnya ke lapisan Gateway (Bridge), sebagaimana ditunjukkan oleh latensi komputasi murni HMAC pada Bridge yang sangat rendah—rata-rata 0,236 ms dengan median 0,070 ms—jauh di bawah ambang toleransi operasional 200 ms yang telah ditetapkan sebelumnya. Di sisi lain, lapisan validasi integritas (*XOR Checksum*) pada perangkat keras *edge* terbukti sangat ringan, di mana ia hanya menambah *overhead* penyimpanan program (*Flash ROM*) sebesar +170 bytes (setara 0,53% dari total kapasitas *flash*) dan alokasi memori dinamis transien (SRAM) sebesar +20 bytes tanpa memicu risiko kebocoran memori (memory leak) pada mikrokontroler Arduino Uno. Capaian metrik kinerja tersebut memastikan bahwa ketersediaan komunikasi dan stabilitas operasional mesin dalam melayani kendali industri secara real-time tetap dapat dipertahankan dengan stabil.

Meskipun sistem usulan terbukti efektif dalam memitigasi serangan manipulasi dan pemalsuan data, penelitian ini masih memiliki keterbatasan fungsional karena belum mencakup pemenuhan mekanisme proteksi terhadap serangan pertunjukan ulang (*anti-replay attack*). Keterbatasan ini merupakan konsekuensi yang disengaja dari desain metodologi penelitian, di mana cakupan evaluasi dibatasi agar dampak overhead HMAC dapat diukur secara terisolasi tanpa variabel timestamp atau nonce. Berdasarkan keterbatasan tersebut, penelitian selanjutnya disarankan untuk menambahkan parameter pengaman tambahan berupa skema penanda waktu (*timestamp*) ataupun implementasi nomor urut (*sequence number*) di dalam payload untuk menangkal celah replay. Di samping itu, pengujian perlu perluasan validasi dengan mengimplementasikan arsitektur keamanan terdistribusi ini pada platform mikrokontroler modern yang memiliki kapasitas memori dan komputasi lebih tinggi, seperti ESP32, guna menganalisis skalabilitas sistem pada ekosistem IoT industri yang lebih luas. Di luar itu, penelitian lanjutan juga perlu mengkaji solusi manajemen kunci dinamis guna menggantikan preshared key statis yang saat ini rentan terhadap risiko kompromi akibat *firmware dumping*, misalnya dengan mengadopsi skema *key derivation function* (KDF) atau protokol pertukaran kunci ringan yang kompatibel dengan perangkat *resource-constrained*.

Daftar Pustaka

[1] Boyes, H., Hallaq, B., Cunningham, J., & Watson, T., "The industrial internet of things (IIoT): An analysis framework," Computers in Industry, vol. 101, pp. 112, 2018.

[2] Qiu, F., Kumar, A., Hu, J., Sharma, P., Tang, Y. B., Xiang, Y. X., & Hong, J., "A Review on Integrating IoT, IIoT, and Industry 4.0: A Pathway to Smart Manufacturing and Digital Transformation," IET Information Security, 2025, doi: 10.1049/ise2/9275962.

[3] Amirkhanova, G., Adilkyzy, S., Amirkhanov, B., Baizhanova, D., & Chen, S., "A Digital Twin Architecture for Integrating Lean Manufacturing with Industrial IoT and Predictive Analytics," Information, vol. 17, no. 2, p. 196, 2026, doi: 10.3390/info17020196.

[4] Liebl, S., et al., "Analyzing the Attack Surface and Threats of Industrial Internet of Things Devices," in 2021 IEEE 26th International Conference on Emerging Technologies and Factory Automation (ETFA), 2021.

[5] Jbair, M., et al., "Threat modelling for industrial cyber physical systems in the era of smart manufacturing," Computers in Industry, vol. 137, p. 103611, 2022.

[6] Yaacoub, J.P. A., Salman, O., Noura, H. N., Kaaniche, N., Chehab, A., & Malli, M., "Cyber physical systems security: Limitations, issues and future trends," Microprocessors and Microsystems, vol. 77, p. 103201, 2020.

[7] Kurniawan, R. A. Z., et al., "Secure Communication Protocol for Arduino-based IoT Using Lightweight Cryptography," in 2022 14th International Conference on Information Technology and Electrical Engineering (ICITEE), 2022.

[8] Erroutbi, A., et al., "Secure and Lightweight HMAC Mutual Authentication Protocol for Communication between IoT Devices and Fog Nodes," Procedia Computer Science, vol. 160, 2019.

[9] Khan, R., et al., "STRIDE-based Threat Modelling for CyberPhysical Systems," in 2018 IEEE International Symposium on Systems Engineering (ISSE), 2018.

[10] Humayed, A., Lin, J., Li, F., & Luo, B., "Cyberphysical systems security—A survey," IEEE Internet of Things Journal, vol. 4, no. 6, pp. 18021831, 2017.

[11] Field, A., Discovering Statistics Using IBM SPSS Statistics, 5th ed. London: Sage, 2018.

[12] Forouzan, B. A., Data Communications and Networking, 5th ed. New York: McGrawHill, 2013.

[13] Stouffer, K., Pillitteri, V., Lightman, S., Abrams, M., & Hahn, A., Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82 Revision 2, Gaithersburg, MD: National Institute of Standards and Technology, 2015.

[14] IEC 62443 Compliance Services Market Research Report 2034, data biaya kepatuhan tahunan UKM vs Fortune 500.

[15] Securing Industrial Automation: A Comprehensive Guide to IEC 62443-4-2, iiot-world.com, kendala sumber daya UKM dalam retrofit legacy.

[16] Wagner et al., Efficient Message Authentication in the Industrial Internet of Things (RePeL), RWTH Aachen COMSYS, 2026.

[17] Modelling and Machine-Checking Bump-in-the-Wire Security for ICS (F2-Pro), Springer/ResearchGate.

[18] Securing Legacy Communication Networks via Authenticated Cyclic Redundancy Integrity Check (ACRIC), ResearchGate, 2024.

[19] Quantum-Resistant Crypto-Agile Inline Authentication and Encryption Framework for IEC 61850 Digital Substations, ACM Sustainability Week 2026.