Field-Sensitive Adaptive Encryption (FSAE) for IIOT-based Digital Twin Systems

1st Boyang Wang
dept. of Electrical and Computer Engineering
Northeastern University
Boston, United States
Wang.b2@northeastern.edu

2nd Vaidehi Ajitsinh Gohil
dept. of Electrical and Computer Engineering
Northeastern University
Boston, United States
gohil.vai@northeastern.edu

***Abstract*—Digital Twin (DT) systems are based on real-time telemetry streams to mirror physical assets and support low-latency control. However, encrypting entire telemetry payloads can introduce computational overhead that risks violating sub-millisecond latency budgets, whereas sending data in plaintext exposes sensitive operational information. This project proposes Field-Sensitive Adaptive Encryption (FSAE), a framework that protects only the most critical fields instead of encrypting every packet. At runtime, a "Sensitivity–Impact–Timeliness" (S-I-T) scoring model and a context-aware policy engine identify high-risk fields and selectively apply AES-256-GCM based on network conditions, user role, and system state. Our prototype and benchmarks show that FSAE preserves field-level confidentiality with negligible performance impact, maintaining the real-time requirements of industrial DT environments.**

***Index Terms*—Digital twins (DT), Industrial Internet of Things (IIoT), field-level encryption, context-aware security, real-time systems, edge computing, AES-GCM.**

I. Introduction

Digital Twin (DT) systems are progressively used in industrial environments to mirror physical assets in real time and support monitoring, analytics, and control. As these systems scale, they must handle continuous telemetry streams that are both delay-sensitive and security-critical, creating a non-trivial trade-off between latency and confidentiality. Recent surveys on Digital Twin security indicate that these systems introduce complex new attack surfaces that require dedicated protection mechanisms [4].

A. Background

The Industrial Internet of Things (IIoT) has enabled the integration of numerous sensors, actuators, and controllers with Digital Twin (DT) platforms. DTs rely on real-time telemetry, such as temperature, pressure, actuator positions, and alarm states, to create virtual models of physical assets or processes. This data enables system visualization, anomaly detection, failure prediction, and, in some cases, direct control.

Yet not all telemetry fields are equally important or sensitive. Non-critical data, like ambient temperature or standard status indicators, may be inconsequential if disclosed. In contrast, data such as operator IDs, specific actuator commands, or production line alerts could reveal trade secrets, safety information, or personal details. Furthermore, many DT applications operate on edge devices with limited resources and stringent latency requirements, particularly when controlling systems in real time. These varied sensitivities, combined with real-time and resource constraints, mean that securing DT telemetry requires nuanced solutions rather than blanket encryption.

B. Problem Statement

Conventional security mechanisms in IoT systems, such as end-to-end TLS tunnels or full-payload encryption, typically treat each message as a single opaque unit. While selective encryption strategies have been successfully optimized for high-bandwidth domains like CCTV video [6], applying similar fine-grained protection to real-time industrial telemetry remains an open challenge. Full-packet encryption requires applying cryptographic operations to every field in the message, including low-risk ones, thereby consuming CPU cycles and increasing processing time on edge devices. For latency-sensitive DT scenarios, the additional overhead and jitter can push end-to-end delays beyond acceptable thresholds.

Moreover, DT telemetry is often consumed by multiple classes of subscribers with distinct roles and privileges, including engineers, operators, management dashboards, and automated analytics services. A uniform encryption policy makes it difficult to expose only the necessary subset of fields to each consumer without either over-sharing or requiring costly decryption and re-encryption at intermediate components. Overall, there is a persistent tension between maintaining strong confidentiality for sensitive fields and preserving the real-time performance and flexibility needed by industrial Digital Twin systems.

C. Project Overview

This project addresses this methodological tension by designing and implementing a context-aware, field-level encryption framework tailored explicitly to Digital Twin telemetry data. Instead of encrypting entire messages indiscriminately, the proposed approach evaluates each telemetry field using a quantitative scoring model that captures its Sensitivity, operational Impact, and Timeliness requirements. Based on this "S–I–T" score and the current runtime context—such as network security posture, subscriber role, and system or event state—the framework selectively applies AES-256-GCM encryption only to fields deemed high risk.

By restricting cryptographic operations to sensitive data and tailoring decisions to the runtime environment, the system aims to maintain the sub-millisecond latency required by real-time DT applications while still ensuring meaningful confidentiality. The remainder of this work describes the design of the scoring model and policy engine, the implementation of the selective encryption pipeline, and an evaluation of its performance overhead in a representative industrial Digital Twin setting.

II. Related Work

Digital twin (DT) systems sit at the intersection of industrial IoT, cyber–physical systems, and real-time control; therefore, prior work spans both cryptographic access control for IoT data and surveys of DT security.

Jammula et al. [1] propose a hybrid lightweight cryptography with attribute-based encryption (LWC-ABE) framework for securing IoT data while keeping the overhead acceptable for resource-constrained devices. Their design combines a lightweight stream cipher, additional classical ciphers, and ABE to provide fine-grained access control and enhanced resistance to common attacks, and it achieves lower encryption/decryption time than conventional ABE schemes on generic IoT workloads. In this paper, we adopt the view that hybrid, multi-layer cryptographic designs can reconcile IoT constraints with fine-grained access control. However, their approach still treats whole messages as the basic unit of protection and does not reason about mixed-criticality fields within a single telemetry packet or about strict sub-millisecond latency budgets in DT feedback loops.

Sicari et al. [2] study attribute-based encryption and sticky policies for data access control in a smart-home IoT middleware. They compare CP-ABE (in which policies are embedded in ciphertexts) with "sticky policies" that travel with the data and are evaluated by a trusted authority, and measure storage, CPU load, and retrieval delay on a real testbed. Their results show a classic trade-off: sticky policies are lighter on CPU but heavier on storage and delay, whereas CP-ABE offers richer cryptographic enforcement at a higher computational cost. From this work, we take the notion of data-centric, policy-driven protection and the importance of aligning crypt choice with system-level performance. In contrast, our project does not rely on heavyweight ABE primitives; instead, we keep role-based access control in the application layer and move the fine-grained confidentiality decision down to which fields deserve AES-GCM, based on runtime context and S-I-T scoring.

Perazzo et al. [3] conduct a detailed performance evaluation of ABE on constrained IoT devices, such as ESP32 and RE-Mote boards, analyzing execution time, energy consumption, and memory usage across different ABE schemes and attribute counts. They show that ABE can be feasible on low-end devices only under carefully limited conditions (e.g., $\geq$ 10 attributes and hardware acceleration). Those naive "worst-case" assumptions in the literature significantly overestimate overhead. This paper strongly motivates our design constraint: pushing full ABE into the hot telemetry path of an industrial DT, where packets are frequent and small—is risky for sub-millisecond control. Instead of embedding complex policies into each ciphertext, our system keeps a simple symmetric primitive (AES-256-GCM) in the data plane. It distributes complexity to a lightweight scoring and policy engine that runs once per message, rather than once per key-policy evaluation.

On the DT side, Alcaraz and Lopez [5] provide a comprehensive survey of security threats in digital-twin systems, organizing attacks across four functional layers (data dissemination/acquisition, data management/synchronization, data modeling/services, and data visualization/access). They highlight that DTs are critical infrastructure, that the attack surface spans both physical and digital spaces, and that security controls must be carefully balanced with operational requirements such as latency, fidelity, and synchronization accuracy.

From this survey, we primarily derive two insights: (i) DT data streams are a particularly attractive target, because they often encode operational IP and real-time state, and (ii) naïve security mechanisms that introduce jitter or disrupt synchronization can be as harmful as a direct attack. Similarly, Qureshi et al. [4] discuss models and tools for enhancing security in Digital Twins, emphasizing the need for specialized defenses.

III. Field-Sensitive Adaptive Encryption (FSAE)

The proposed solution, Field-Sensitive Adaptive Encryption (FSAE), departs from traditional full-packet encryption. Instead of treating each telemetry message as an opaque blob, FSAE parses the JSON payload and selectively encrypts only those fields deemed sensitive by a quantitative model and the current runtime context. This approach allows the system to preserve confidentiality where it matters most, while avoiding unnecessary cryptographic overhead on low-risk fields.

A. Field Sensitivity Analysis (S–I–T Model)

At the core of FSAE is a field-level sensitivity model that assigns a score to every data field in the telemetry schema. Each field is evaluated along three dimensions:

- Sensitivity (S): the privacy or confidentiality impact if the field is exposed (0–3).
- Impact (I): the operational safety risk if the field is modified or forged (0–3).
- Timeliness (T): the importance of the field to real-time control decisions (0–3).

The overall S–I–T score for a field is calculated as a weighted sum of these components. In the current prototype, equal weights are used, so the score is given by

$$Score = 3S + 2I + T \tag{1}$$

This yields a maximum possible score of 9 for highly critical fields and a minimum of 0 for fields with negligible relevance

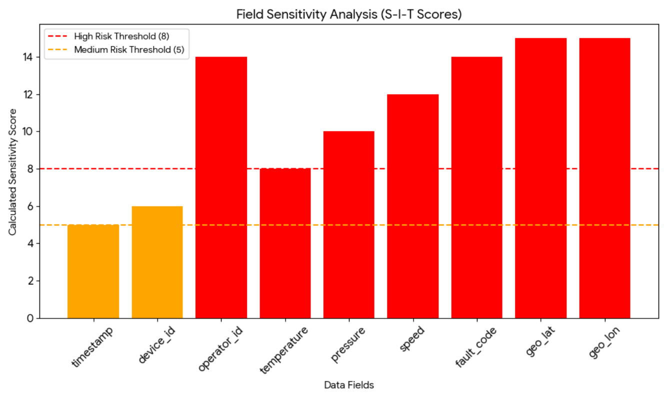

Fig. 1. Field Sensitivity Analysis (S-I-T Scores)

across all three dimensions. In, Figure 1 illustrates how representative fields are scored and categorized into risk classes. Fields such as `operator id` and `geo lat` obtain high S–I–T scores that exceed the "High Risk" threshold (Score ≥ 8), reflecting their potential to reveal personally identifiable information or precise physical locations. Therefore, these fields are always marked for encryption. In contrast, a generic timestamp often receives a much lower score and falls into the Medium or Low risk category, depending on the chosen S–I–T configuration. Such fields can be transmitted safely in plain text in many scenarios, thereby preserving observability and reducing cryptographic overhead.

In the current design, fields with a score of 8 or higher are classified as High Risk and must be encrypted. Fields with scores in the intermediate range (e.g., 5–7) are classified as Medium Risk and may or may not be encrypted, depending on the runtime context. Fields with scores below five are treated as Low Risk and are typically left in plain text unless the environment becomes particularly hostile.

B. Context-Aware Policy Engine

The S–I–T score provides a static measure of field sensitivity, but the actual encryption decision is made dynamically by a context-aware policy engine. This engine continuously evaluates three contextual variables associated with each message:

- The network risk level, categorized as LAN (trusted), Wi-Fi (semi-trusted), or Public (untrusted);
- The subscriber role, such as Engineer, Viewer, or Admin;
- The event state, distinguishing between normal operation and alert or fault conditions.

Given the S–I–T scores and the current context, the policy engine decides whether each field should be encrypted, left in plain text, or treated more conservatively than usual. For example, when the system enters an Alert state, the policy becomes stricter: all High- and Medium-risk fields are encrypted regardless of the underlying network type, ensuring that detailed fault or incident information is not exposed. On Public networks, the engine similarly enforces encryption for both High and medium-risk fields to protect against eavesdropping and traffic inspection. In contrast, on a trusted LAN during regular operation, the policy may only require encryption of high-risk fields, allowing medium-risk fields to remain in plain text to save CPU cycles and reduce latency.

By combining the static S–I–T scores with these contextual parameters, the policy engine effectively approximates fine-grained, attribute-based access control using only lightweight symmetric cryptography. This enables the system to adapt its protection level to the current threat posture and operational needs without incurring the high costs of more complex cryptographic schemes.

C. Cryptographic Mechanism

Once the policy engine has determined which fields should be protected, FSAE applies AES-256-GCM (Galois/Counter Mode) to those fields. AES-GCM is chosen because it provides both confidentiality and authenticated integrity in a single operation, which is essential for detecting tampering and eavesdropping in industrial settings.

Each encrypted field is associated with a unique 12-byte random nonce, generated per field and per message. This per-field nonce construction avoids nonce reuse even when the same field appears in many messages, and it enables independent decryption of each field by authorized subscribers. The resulting ciphertext, along with any necessary metadata (e.g., the nonce and authentication tag), is embedded directly within the JSON structure. This design preserves the validity of the overall JSON document: non-sensitive fields remain as plain values, while sensitive fields are replaced with structured objects that encode their encrypted form.

By keeping the message format JSON-compatible and applying AES-256-GCM only where required, FSAE integrates seamlessly into existing publish–subscribe pipelines while enforcing field-level confidentiality with minimal additional latency.

IV. System Implementation

A complete Digital Twin telemetry pipeline was implemented in Python to validate the FSAE framework. The prototype emulates an industrial IoT setting in which a machine publishes JSON telemetry over MQTT, a broker routes messages to multiple subscribers with different privileges, and a dashboard visualizes both the raw stream and the effect of the encryption policy. The FSAE logic is encapsulated in a reusable policy module that can be invoked by any publisher or subscriber in the pipeline.

A. System Architecture

The overall architecture follows a publish–subscribe model based on MQTT. A publisher process, representing a machine or edge gateway, generates synthetic telemetry records and invokes FSAE to selectively encrypt fields before publishing them to an MQTT topic. An Eclipse Mosquitto broker running on `localhost` handles message routing and decouples publishers from subscribers. On the subscriber side, three roles are implemented. The *Engineer* subscriber possesses the decryption key material and can reconstruct the full plain text view of each message, including all fields marked as sensitive by the policy. The *Viewer* subscriber is a least-privilege consumer that may view only low-sensitivity fields in clear text, while encrypted blobs are ignored. A third subscriber, the *Dashboard*, visualizes the stream for monitoring and debugging purposes. It renders live plots of selected telemetry values and shows which fields are currently being encrypted under the active policy.

B. Key Software Components

The FSAE prototype is organized into four main components.

The first component is the policy module, which contains the core logic. It defines the S-I-T configuration and implements the select_fields_to_encrypt function, which maps static scores and runtime context to a set of fields that must be protected. A dictionary specifies the S, I, and T values for each field, for example, temperature, speed, and operator_id. The module also exposes an encrypt_fields wrapper that checks a global ENABLE_CRYPTO flag and, when encryption is enabled, calls a helper function to perform AES-256-GCM encryption using the Python cryptography library. After the encryption decisions are applied, the module injects a small _enc_header structure into the JSON message to indicate the algorithm, policy version, context, and list of encrypted fields.

The second component is the publisher, which simulates an IoT device or gateway. It generates random telemetry values such as temperature, pressure, and speed at a rate of about 2 Hz, attaches a timestamp, and then calls the policy module to obtain the list of fields to encrypt for the current message. To demonstrate context awareness, the publisher periodically changes the network risk level in the context, for example, rotating between ”LAN” and ”WiFi” every 20 messages. This triggers different encryption decisions without modifying the application logic. A configuration flag ENABLE_CRYPTO allows the system to be run in a baseline mode with no encryption or in FSAE mode, enabling straightforward A/B comparisons of latency.

The third component is the Engineer subscriber. It receives messages from the broker, inspects the _enc_header, and uses the policy module to decrypt any encrypted fields and recover the original plain text. For each message, it computes one-way latency as the difference between the local receive time and the publisher's timestamp. The subscriber logs per-message latency and summary statistics, such as p50 and p95, to a CSV file for subsequent analysis.

The fourth component is the Viewer subscriber, which models a restricted role. It constructs a ”safe view” of each message that includes only a predefined subset of low-sensitivity fields, for example timestamp and temperature, and ignores any encrypted values. This demonstrates that FSAE can support least-privilege data access without requiring complex access-control logic inside the application.

Finally, the dashboard is implemented using Plotly Dash. It subscribes to an MQTT topic in a background thread, updates live time-series plots for selected telemetry fields, such as temperature and speed, and displays both the raw JSON message and the list of fields currently encrypted under the policy. This provides an interactive view of how changes in context or configuration affect the encryption behavior over time.

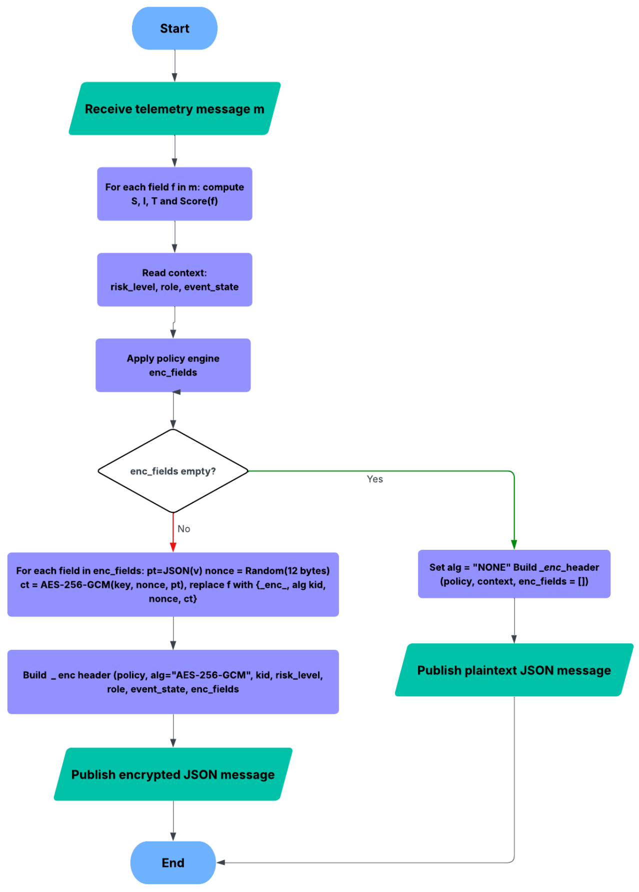
Start
Receive telemetry message m
For each field f in m: compute S, I, T and Score(f)
Read context: risk_level, role, event_state
Apply policy engine enc_fields
enc_fields empty?
Yes
No
For each field in enc_fields: pt=JSON(v) nonce = Random(12 bytes) ct = AES-256-GCM(key, nonce, pt), replace f with {_enc_, alg kid, nonce, ct}
Build _ enc header (policy, alg="AES-256-GCM", kid, risk_level, role, event_state, enc_fields
Publish encrypted JSON message
Set alg = "NONE" Build _enc_header (policy, context, enc_fields = [])
Publish plaintext JSON message
End

Fig. 2. Field-Sensitivity Adaptive Encryption Flowchart

C. FSAE Processing Algorithm

The end-to-end FSAE decision and encryption logic executed at the publisher is summarized in the flowchart shown in the Fig.2. When a telemetry message m is generated, the system first iterates over each field in the message to compute its S, I, and T values and the corresponding S-I-T score. It then reads the current runtime context, including the network risk level, subscriber role, and event state. The policy engine uses the S-I-T scores, along with this context, to produce a set of field names, enc_fields, that must be encrypted for this message.

If enc_fields is empty, the policy engine sets the algorithm in the header to ”NONE”, builds an _enc_header that records the context and an empty encrypted-field list, and publishes the original JSON message as plain text. If enc_fields is not empty, the publisher processes each selected field independently: it serializes the field value to JSON, generates a fresh 12-byte random nonce, computes the AES-256-GCM cipher text

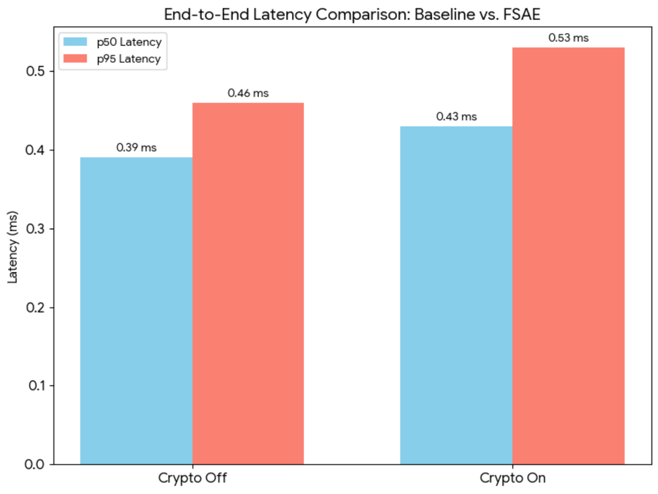

Fig. 3. End-to-End Latency Comparison

and authentication tag with the shared key, and replaces the original field value with a structured object containing the encrypted representation, including algorithm identifier, key identifier, nonce, and cipher text. After all selected fields have been transformed in this way, the publisher constructs an _enc_header indicating that AES-256-GCM is in use and listing the encrypted fields together with the current context, and then publishes the resulting encrypted JSON message.

This algorithm ensures that messages that do not require protection incur essentially no cryptographic overhead. In contrast, messages containing high-risk fields are selectively protected at the field level with authenticated encryption, without changing the overall JSON structure or the basic publish-subscribe interaction pattern.

V. Performance Evaluation

To verify the lightweight nature of FSAE, we evaluated the system in two operating modes: a baseline configuration with encryption disabled (crypto_off) and the DFSE configuration with FSAE enabled (crypto_on).

A. Methodology

The primary metric is end-to-end latency, defined as the time difference between the timestamp attached by the publisher at data generation and the timestamp recorded by the subscriber upon reception. The workload consists of a continuous telemetry stream at a fixed rate of 2 messages per second. Latency measurements are collected and aggregated in 10-second windows; for each window, we compute the median (P50) and the 95th-percentile (P95) latency. This allows us to capture both typical behavior and tail effects while keeping the workload and message structure identical across both modes.

B. Quantitative Results

Figure 3 summarizes the impact of enabling FSAE. In the baseline crypto_off mode, the median latency is 0.39 ms and the 95th-percentile latency is 0.46 ms, with throughput stable

Mode	*P50 Latency (ms)*	*P95 Latency (ms)*	*Throughput*
Crypto Off	0.39 ms	0.46 ms	2 msg/s
Crypto On	0.43 ms	0.53 ms	2 msg/s

at 2 messages per second. When FSAE is enabled (crypto_on), the median latency increases slightly to 0.43 ms and the 95th percentile to 0.53 ms, while throughput remains unchanged at two messages per second. The results indicate that introducing field-level AES-256-GCM encryption adds only a slight and consistent delay to the overall pipeline.

C. Analysis

The observed overhead is minimal: enabling FSAE increases median latency by approximately 0.04 ms and the 95th percentile by about 0.07 ms, keeping the additional cost well below 0.1 ms in both cases. Throughout the experiments, throughput remained constant at 2 messages per second, and no significant spikes in jitter or instability were observed in the latency distribution. Because the total end-to-end latency remains below 1 ms even with encryption enabled, the prototype satisfies the timing constraints typical of high-speed Digital Twin scenarios. It supports the claim that FSAE provides practical, real-time-compatible protection for sensitive telemetry fields.

VI. Discussion

From a confidentiality perspective, the use of AES-256 in GCM mode provides strong protection for sensitive fields, assuming proper key management and nonce uniqueness. Because only selected fields are encrypted, an attacker who intercepts the telemetry stream cannot recover the underlying values of high-risk fields without access to the appropriate keys, even if they can observe the overall message structure. A key advantage of FSAE over traditional tunnel-based approaches, such as TLS, is the granularity of protection. In a typical TLS deployment, data is decrypted at the network edge and then transmitted in plain text within the application layer, which means that any component within the trusted boundary can see the entire message. In contrast, FSAE preserves encryption for selected fields throughout the application. This enables different subscribers on the same MQTT topic, such as an Engineer and a Viewer, to receive identical messages but reconstruct different views: the Engineer can decrypt all sensitive fields. At the same time, the Viewer sees only low-sensitivity fields in clear text and opaque encrypted blobs for the rest. This design aligns more closely with least-privilege principles at the data level. The prototype also demonstrates that encryption decisions can adapt to changing conditions. When the publisher simulates a higher-risk context, such as moving from a LAN to a WiFi environment or entering an alert state, the policy engine automatically extends protection to Medium-risk fields that were transmitted in plain text under normal conditions. This dynamic adjustment of encryption targets shows that FSAE can respond to shifts in network risk and system state without requiring manual reconfiguration or code changes, and without altering the basic messaging pattern.

A. *Trade-Off*

The main cost of field-level encryption in this design is an increase in payload size. Each encrypted field carries additional metadata, including a nonce, an authentication tag, and small JSON structures such as the algorithm identifier and key identifier. As a result, the serialized JSON message is larger than its plain-text counterpart, notably when many fields are encrypted simultaneously. In the context of modern industrial networks, however, the resulting byte-level overhead is modest relative to available bandwidth and is offset by the benefits of granular, role-aware confidentiality. Given that the measured latency impact remains well below one millisecond and throughput is unchanged, this trade-off is acceptable for the class of Digital Twin applications targeted by this work.

VII. Conclusion

This project demonstrated that Field-Sensitive Adaptive Encryption (FSAE) is a practical alternative to traditional full-packet encryption for Digital Twin systems. By combining an S-I-T-based field-sensitivity model with a context-aware policy engine, the system can protect high-risk fields while leaving low-risk fields in plain text, thereby preserving observability and reducing unnecessary cryptographic operations. The prototype implementation delivered an end-to-end MQTT-based Digital Twin pipeline in Python, complete with policy module, publisher, role-aware subscribers, and a live dashboard. Experimental results showed that high-grade AES-256-GCM encryption can be selectively applied with only a small additional latency of approximately 0.04 milliseconds at the median and less than 0.1 milliseconds at the 95th percentile, without degrading throughput. Together, these results support the claim that FSAE can deliver fine-grained confidentiality while remaining compatible with the real-time constraints of industrial Digital Twin workloads.

A. *Future Work*

There are several directions in which this work can be extended. One avenue is to replace the static S-I-T scoring configuration with a data-driven or machine-learning-based model that infers field sensitivity from observed behavior, anomaly patterns, or policy feedback over time. Another is to explore hardware acceleration, for example, by using AES-NI on x86 platforms or ARM cryptographic extensions on embedded devices, to further reduce the already small encryption overhead. In parallel, integrating a dedicated key management solution, such as a Hardware Security Module, would strengthen key storage security and support automated key rotation in multi-tenant deployments. Finally, future experiments should scale the prototype to a multi-device environment with numerous publishers and subscribers to study how FSAE behaves under higher load and more complex traffic patterns, and to validate its applicability to larger industrial Digital Twin deployments.

References

[1] M. Jammula, V. M. Vakamulla, and S. K. Kondoju, "Hybrid lightweight cryptography with attribute-based encryption standard for secure and scalable IoT system," *Connection Science*, vol. 34, no. 1, pp. 2431-2447, 2022.

[2] S. Sicari *et al.*, "Attribute-based encryption and sticky policies for data access control in a smart home scenario: a comparison on networked smart object middleware," *Int. J. Inf. Secur.*, vol. 20, no. 5, pp. 695-713, 2021.

[3] P. Perazzo *et al.*, "Performance evaluation of attribute-based encryption on constrained IoT devices," *Comput. Commun.*, vol. 170, pp. 151-163, 2021.

[4] A. R. Qureshi *et al.*, "A survey on security enhancing Digital Twins: Models, applications and tools," *Comput. Commun.*, p. 108158, 2025.

[5] C. Alcaraz and J. Lopez, "Digital twin: A comprehensive survey of security threats," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 3, pp. 1475-1503, 2022.

[6] K. Kaur *et al.*, "IoT CCTV Video Security Optimization Using Selective Encryption and Compression," *Int. J. Adv. Comput. Sci. Appl.*, vol. 16, no. 2, 2025.