IoT-Based Secure Microgrid Monitoring System with Advanced Intrusion Detections

Jason Paul
Dept of SOCSE
Presidency University
jasonpderick@gmail.com

Abstract- Nowadays small power grids use internet-connected sensors to keep track of things right away - also to manage operations across spots. Still, they can get hit by fake data, repeated signals, or unwanted inputs now and then. This project builds a safer way to watch over tiny energy networks using a cheap Wi-Fi chip hooked up to temp, humidity, and light detectors. Info travels safely online thanks to encrypted messaging through a cloud service that handles short messages securely. On the receiving end, smart but simple rules check for dodgy sensor tricks, copied packets, weird sending speeds, or clock mismatches - all without needing heavy computing muscle. Live updates pop up instantly on a web screen made with basic tools so staff see issues early plus react quicker. Using safe messaging, unique device checks, or smart threat spotting in one setup helps fix weak spots in old microgrid tracking systems - ones missing attack alerts or full-chain safety. Tests show sensors work well, TLS doesn't slow things much, or fake hacks get caught fast, suggesting this method could offer tough, cheap, expandable security for today's power grids.

Keywords- Microgrids, IoT, MQTT, TLS/SSL, Intrusion Detection System (IDS), Cyber-Physical Systems (CPS), Cybersecurity, Smart Grids, ESP8266, Real-time Monitoring

I. INTRODUCTION

Microgrids are a decentralized way to distribute energy. They bring together renewable generation units, local storage, smart meters, and consumer loads into a manageable regional network. Their effectiveness relies heavily on IoT sensors and control devices that constantly track environmental and electrical conditions. This digital integration allows for better demand response and system optimization but raises serious cybersecurity issues. If communication between nodes happens over open or unsecured networks, attackers can exploit weaknesses in the protocols. This can lead to incorrect system decisions, energy imbalances, or even complete shutdowns. Traditional microgrid monitoring systems focused more on physical reliability than on cybersecurity. Many IoT nodes currently deployed use lightweight communication stacks, have limited memory, and provide little encryption. This setup makes them easy targets for network intrusions, sensor spoofing, replay attacks, and denial-of-service (DoS) incidents. Attacks on energy systems have caused large-scale grid failures and service disruptions in the past. These incidents highlight the need for real-time detection and proactive measures. A resilient microgrid must continuously monitor both environmental and operational factors while also detecting any interference from adversaries. To tackle these challenges, this research creates a secure microgrid monitoring framework. It combines hardware-level sensing with a layered intrusion detection system. The setup uses an ESP8266-based sensor node with DHT22 and LDR modules, which communicate via MQTT over TLS to a cloud broker. MQTT is chosen for its low overhead and effectiveness with limited devices, but extra safeguards are needed because of known vulnerabilities like brute-force authentication, topic injection, and packet spoofing. The proposed design includes timestamped telemetry packets, device-specific identity labels, structured message checks, and real-time alerting for detecting malicious activity.A web-based dashboard displays system parameters, threat alerts, and statistics on attack types. It uses real-time socket streaming to show telemetry with low latency. The outcome is a complete cyber-physical security system that continuously monitors, detects intrusions online, logs forensics, and notifies operators. This method supports a scalable and secure deployment of microgrid IoT, making it suitable for smart campuses, industrial sites, and distributed renewable energy hubs.

II. LITERATURE SURVEY

Smart grids now depend more on linked digital-physical setups, making it harder to keep power control safe, stable, and fast. Instead of just adding parts together, researchers like Cintuglu et al. [1] stress using lab-built grid models to study how systems react instantly, check if attack detectors work, or see what damage hacks can do in energy networks spread across locations. They found these mini-grids need full-scope protection plans because hardware and software layers interact closely. On top of that, Al-Fuqaha et al. [2] laid out clear details about IoT frameworks, supporting tools, and efficient data exchange methods suited for such systems. Their analysis points out MQTT works best for weak-power sensors and controllers in microgrid layouts - thanks to small resource use, a topic-based messaging setup, along with solid performance during distant supervision tasks.MQTT's communication and safety aspects have drawn plenty of attention. Taştan and Çağlar [3] looked into flaws in its standard setup, pointing out how using TLS/SSL can reduce threats like data interception or fake brokers. Their tests proved MQTT with TLS still runs smoothly enough for live microgrid operations needing secure, accurate data flow. At the same time, spotting intrusions stays critical in cyber-physical setups. Mitchell and Chen [4] reviewed various IDS methods, finding that behavior-driven detection works better than rule-based ones against new, unexpected attacks on smart power networks.The demand for slim, flexible intrusion detection systems is growing alongside expanding IoT networks. Because IoT gadgets often have weak processors, Islam and team [5] built a minimal anomalydetecting system that works fast without slowing them down. In a related move, Asraf together with Preeth [6] designed a layered defense model meant for smart energy grids - this setup uses light IDS tools combined with constant observation. That study proves stacking protection layers boosts success rates when spotting communication-based threats, giving today's grid designers solid practical guidance.On a bigger scale, Sridhar's team [7] looked into weak spots in power networks, pointing out how broken comms links, fake sensor readings, or sneaky control commands might throw off grid stability. Moving on, Singh's group [8] broke down key risks in smart grids using IoT - like repeated signal attacks, false identities, intrusions, and altered data - while pushing solutions such as strong coding methods, intrusion detection systems, and tougher device safeguards. For keeping gadgets and their messages safe, Ferrag et al. [9] rolled out ways to verify and permit IoT units within small-scale smart grids, stressing trusted setups based on tokens or digital certificates.Wankhede and Bagade [10] tested a microgrid monitor using IoT that pulls live power data via the cloud. Still, their setup didn't include attack sensing or data encryption, which could open security risks. That weakness shows a big missing piece in current designs: even though we have tracking tools, not many combine solid data protection with real-time threat spotting. Newer ideas - like MQTT messages shielded by TLS [11], IoT displays backed by cloud services [12], or instant visual updates built on Flask [13] - prove it's possible to link safe messaging with quick-reacting frontends in today's connected gadgets.On top of that, NIST's rules for ICS safety [14] focus on layered protection, constant oversight, also solid device verification - ideas fitting closely with what microgrids need. Looking at long-term impact, UN goals [15] stress dependable power setups along with tough infrastructure, showing how protected microgrid tech supports worldwide progress.In short, earlier work shows solid proof that safe IoT links, light-duty threat spotting, plus verified gear control matter a lot in microgrids. Still, hardly any papers mix live tracking with onboard oddity checks along with encrypted MQTT using TLS. Our setup tackles this missing piece by combining TLSprotected MQTT messaging, instant visual updates through Flask tools, also a lean anomaly-focused detector - boosting uptime strength and digital safety inside up-to-date power grids.

III. PROPOSED SYSTEM

A. Overview and Design Goals

The system, Secure Microgrid Monitoring and Intrusion Detection System using IoT and MQTT, aims to achieve the following goals:

Real-time telemetry: It continuously captures and visualizes environmental and operational metrics, such as temperature, humidity, and illumination, from distributed sensor nodes.

Secure transport: It ensures confidentiality, integrity, and authenticity of sensor telemetry using TLS-protected MQTT through a commercial broker, HiveMQ Cloud.

Lightweight intrusion detection: It detects communication and data anomalies in near real-time, such as replay, flooding, tampering, injection, protocol fuzzing, and timing attacks. This is practical for resource-constrained setups.

Operational visibility: The system provides a web dashboard with live charts, an alert stream, logs, CSV export, and session reports to support operators and auditors.

Forensic logging and reporting: It maintains persistent session logs and allows exportable reports for incident investigation and compliance.

B. System Components and Data Flow

The system consists of four primary layers, as shown in Figure 1

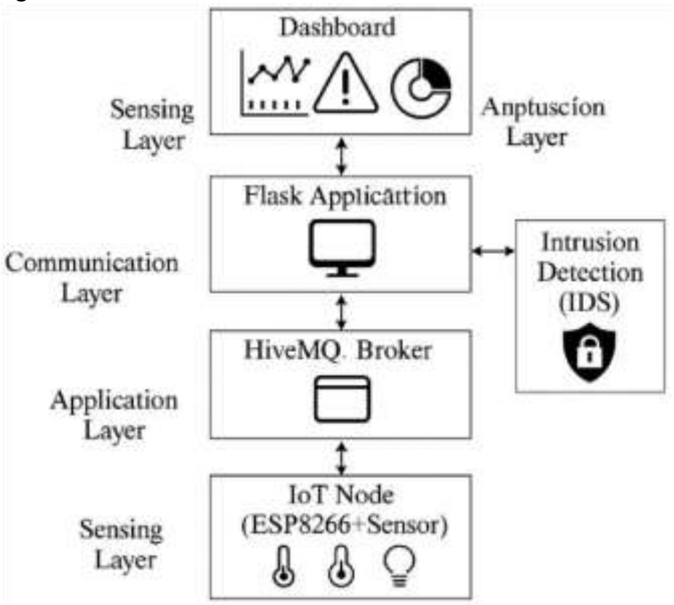

Fig. 1. High-level workflow from sensing to detection and visualization

Sensing Layer (Edge): ESP8266 NodeMCU devices run firmware that samples a DHT22 (temperature/humidity) and an LDR (light). The firmware periodically reads sensors and publishes JSON telemetry to microgrid/sensors/data over MQTT/TLS. Device metadata, including deviceId, messageCount, RSSI, and timestamp, is included for device validation and replay protection.

Communication Layer: The MQTT broker, HiveMQ Cloud, provides a secure publish/subscribe backbone with TLS on port 8883. Devices act as publishers, while the Flask IDS server subscribes to sensor topics. The broker also enforces username/password authentication and organizes topics to limit exposure.

Server & IDS Layer: A Python Flask application, enhanced with Flask-SocketIO, ingests messages from the broker via an MQTT client, runs the AdvancedIDS engine, logs alerts, and sends real-time socket events to connected dashboards. Key server responsibilities include:

Message parsing & validation: This involves JSON parsing, timestamp and deviceId verification, and value normalization.

IDS analysis pipeline: A set of detectors, including rate/DDoS detection, injection pattern matching, replay detection, statistical tampering detection, protocol fuzzing checks, session token reuse, timing analysis, crypto/firmware patterns, and MITM heuristics, are applied to each message. They are implemented in the AdvancedIDSEngine class.

Background attack simulation: This controlled simulator injects synthetic attacks for evaluation and demonstration purposes, helping to validate the detection logic.

Alerting & logging: Alerts are stored in session-based logs, added to a runtime alerts array, and can be exported to CSV or a human-readable report.

Visualization Layer (Dashboard): A single-page dashboard served by Flask uses Chart.js and Socket.IO in the browser to display time-series sensor charts, a doughnut distribution of attack types, per-type counters, and a chronological alert stream. Operators can download logs or reports and clear alerts.

The overall interaction among the sensing layer, communication layer, IDS engine, and visualization components is shown in Fig. 2.

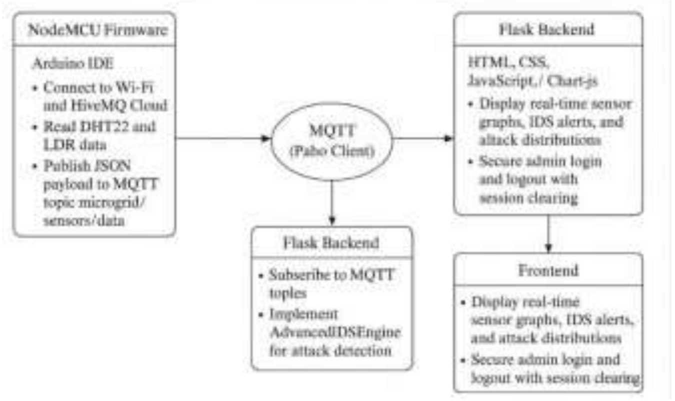
NodeMCU Firmware
Arduino IDE
• Connect to Wi-Fi and HiveMQ Cloud
• Read DHT22 and LDR data
• Publish JSON payload to MQTT topic microgrid/sensors/data
MQTT
(Paho Client)
Flask Backend
HTML, CSS, JavaScript,/ Chart-js
• Display real-time sensor graphs, IDS alerts, and attack distributions
• Secure admin login and logout with session clearing
Flask Backend
• Subscribe to MQTT topics
• Implement AdvancedIDSEngine for attack detection
Frontend
• Display real-time sensor graphs, IDS alerts, and attack distributions
• Secure admin login and logout with session clearing

Fig. 2. Overall architecture of the Secure Microgrid Monitoring and IDS System.

A simplified high-level operational workflow of the microgrid security system is illustrated in Fig. 3.

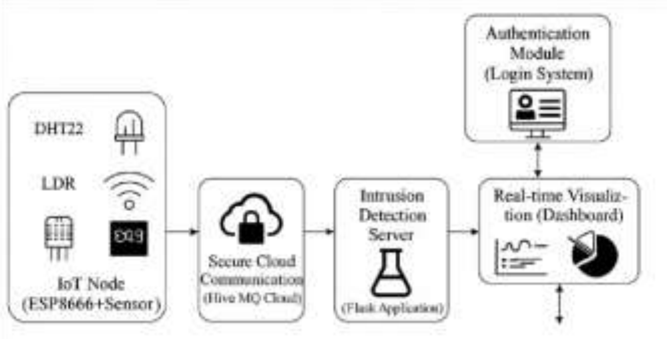
Authentication Module (Login System)
DHT22
LDR
IoT Node (ESP8666+Sensor)
Secure Cloud Communication (Hive MQ Cloud)
Intrusion Detection Server
(Flask Application)
Real-time Visualiz-tion (Dashboard)

Fig. 3. High-level workflow from sensing to detection and visualization.

C. IDS Algorithms & Heuristics The IDS is a hybrid rule/heuristic engine chosen for predictable performance and low resource use:

DDoS / Rate Detection: This method tracks arrival timestamps in a sliding window (traffic_counter). It flags thresholds like over 25 messages in 5 seconds as critical and over 15 as high, detecting flooding and rapid publish storms from faulty nodes or compromised devices.

Injection Detection: This method uses pattern matching against SQL, NoSQL, OS command, XSS, and path traversal signatures. The payload is converted to lowercase and scanned for tokens such as SELECT, DROP, ;, eval(, <script>, ../, etc. This detects attempts to misuse telemetry channels for command injection or payload delivery.

Replay Detection: Messages are hashed (using MD5 of message content) and stored in a bounded deque, message_history. Duplicate hashes within the window indicate replay attacks. Combined with timestamp validation, this minimizes the risk of an attacker replaying old telemetry to mislead control logic.

Tampering / Statistical Outlier Detection: Value range checks (baseline temperature 20–35°C, humidity 30–80%, light 0–1023) and statistical checks (3σ rule using running mean and standard deviation over recent samples) identify manipulated sensor readings, such as adversarial spoofing or physical tampering.

Protocol Fuzzing Detection: This method identifies invalid types (non-dict/string), oversized payloads (over approximately 5000 characters), null bytes, or parse errors. These are all signs of fuzzing attempts.

Session Hijack & Token Reuse: It searches payloads for long hexadecimal sequences that may represent tokens. Unusual reuse or appearance of these sequences raises suspicion.

Timing Attack Detection: This method monitors inter-message intervals for microsecond rhythmic patterns that may signal timing channel probing or covert channels.

MITM Heuristics: It uses simplified detection through TLS metadata, such as unexpected TLS version or random mismatch probability. Because certificate verification might be offloaded or configured insecurely, the system relies on heuristics and suggests stronger TLS verification for production. Zero-day, Crypto, and Firmware Patterns: Heuristics look for shellcode markers, weak crypto indicators (like MD5, SHA1, ECB), and firmware update tokens (.bin, OTA, memory addresses) to spot attempts to deliver payloads for remote exploitation or unauthorized firmware.

Each detector returns a (type, message, severity) triple; the aggregator records detection times and updates system stats, including packets analyzed, threats blocked, and average detection time for reporting. D. Implementation Notes & Rationale

Flask and Flask-SocketIO are chosen to host the dashboard, serve REST endpoints (/api/dashboard-data, /api/export-csv, /api/system-report), and push real-time updates to clients via WebSockets. This choice allows seamless integration of Python IDS logic on the server and tidy interaction with NumPy for statistics used in tampering detection.

The MQTT client (paho.mqtt.client) connects to HiveMQ Cloud with TLS. The code sets tls_set() and tls_insecure_set(False) wherever possible. In the Node firmware, espClient.setInsecure() is used as a pragmatic step but should be avoided in production. Instead, devices should pin the broker certificate or use the broker's CA to validate certificates.

The edge firmware (ESP8266) publishes structured JSON payloads, including deviceId, messageCount, RSSI, uptime, and a timestamp. This metadata is vital for replay protection, device validation, and root cause analysis.

Logging & Reporting: Session logs (alerts_{session_id}.log), CSV export, and generate_system_report() offer operational transparency and forensic outputs suitable for post-incident review.

E. Operational Scenarios & Use Cases

Normal operation: Devices publish data at regular intervals, and the dashboard displays rolling charts. The IDS identifies messages as benign.

Sensor tampering: Sudden out-of-range temperatures trigger tampering alerts and notify operators. The system logs include a full sensor history for the time range.

Flooding / DDoS test: Rapid publish bursts from a device trigger DDoS detection and increase the risk score. The server can throttle or temporarily block offending client IDs. Firmware injection attempt: OTA-like messages containing .bin or memory addresses are flagged and blocked for review.

F. Extensibility & Design Tradeoffs

Edge vs. gateway detection: The current design places heavier detection logic at the server, which is acceptable for resource constraints and easier updates. To enable faster mitigations, small signature checks could be added to the firmware at the cost of added complexity.

Machine learning vs. rule-based: ML approaches can identify new anomalies but require labeled data and compute power. Hybrid methods, combining rule-based gating and optional ML scoring at the server, provide a practical path for evolution.

IV. SECURITY ANALYSIS

A. Threat Model

The system protects against the following adversary capabilities:

Network eavesdropping / passive sniffing: Attackers can capture MQTT traffic on the network. Message injection & spoofing: Attackers can publish messages to the broker pretending to be devices.

Replay attacks: Captured valid messages are replayed to deceive the server or controller.

Flooding / DDoS: An attacker or compromised node generates excessive messages to overwhelm the broker or server. Payload attack vectors: Attempts to deliver command payloads, firmware images, or shellcode through telemetry channels. Protocol fuzzing & malformed messages: Automated fuzzers try to crash or exploit server or broker parsing logic.

Session/token theft: The reuse of session tokens or credentials can occur through side-channels. MITM attempts on TLS connections: Attackers attempt to downgrade or intercept TLS sessions. B. Defensive Coverage & Detection Mapping Confidentiality & Integrity (TLS): Using TLS to the broker prevents passive eavesdropping and tampering when certificate validation is enforced on both client and server. Currently, the Python side attempts TLS verification, while the ESP firmware uses espClient.setInsecure(), which weakens protection. The recommendation is to provision device root CA or pin server certificates and enable full verification on devices to prevent MITM.

Authentication (broker credentials): Broker username/password protect against anonymous publishes. For stronger security, implement unique device credentials with per-device certificates or tokens and limit topic permissions.

Replay detection: MD5 hashing combined with bounded history prevents immediate replays. For stronger assurance, combine message hashes with monotonic counters or strictly validated timestamps. Using NTP in firmware is helpful; ensure that clock drift boundaries are enforced.

Flooding/DDoS detection: Sliding window counters enable quick detection of publish storms and can trigger rate limiting or temporary blacklisting of client IDs. To increase security, integrate broker-side rate limits and use per-client quotas.

Injection / fuzzing detection: Pattern matching identifies obvious injection payloads and malformed messages. While this offers high precision for known strings, it may not detect obfuscated payloads. Augment this method with size limits, strict JSON schema validation, and well-defined allowed fields.

Tampering detection: Statistical 3σ checks and baseline ranges can indicate physical manipulation or spoofed values. Combining multiple sensor cross-checks can improve detection accuracy.

Session hijack detection: Token reuse heuristics can flag certain attacks. A stronger approach involves rotating tokens with short lifetimes and using per-session nonces.

Firmware & crypto attack detection: Heuristics for firmware markers, weak hash indicators, and padding patterns can catch careless attempts to deliver malicious payloads. Enforce signed firmware updates for cryptographic verification to prevent unauthorized firmware injection.

C. Limitations & Residual Risks

Certificate verification inconsistencies: The ESP using setInsecure() increases exposure to MITM risks; an attacker could impersonate the broker, inject false telemetry, or capture credentials. This is a high-priority issue to address.

Heuristic false positives/negatives: Rules and heuristics can generate false positives when legitimate payloads are flagged and false negatives when new obfuscation bypasses patterns. The system tracks false_positive_rate in system stats, but periodic manual review and rule fine-tuning is important.

Insider threats & compromised broker credentials: If broker credentials are exposed or an operator workstation is compromised, attackers may publish seemingly legitimate telemetry. Mitigations include per-device credentials, access control lists (ACLs), and operator multi-factor authentication (MFA).

Time synchronization dependency: Replay and timestamp checks rely on reasonably synchronized clocks. Devices lacking good NTP availability might be misclassified or allow replays. Implementing monotonic counters or server-issued nonces improves replay prevention.

Denial by mitigation: Active mitigation strategies such as blocking client IDs should avoid falsely blocking legitimate devices. A graduated mitigation approach involving throttling, alerts, and operator approval for persistent blocks is recommended. D. Hardening Recommendations

Enforce mutual TLS and certificate pinning: Each device should have a client certificate or the broker CA installed, and setInsecure() should be removed. Use mutual TLS where feasible.

Per-device identity & ACLs: Assign unique client IDs and broker ACLs, ensuring each device can only publish to its own topic. Periodically rotate credentials.

JSON schema validation: Implement strict schema validation on the server to reject unexpected fields or types before IDS processing.

Rate limiting at broker & server: Configure HiveMQ or use fronting proxies to enforce per-client quotas and reduce flooding risks.

Signed firmware updates: Require cryptographically signed OTA images and check signatures on devices before applying updates. Reject .bin payloads received through telemetry; firmware updates must be separate and authenticated.

Logging & SIEM integration: Forward logs and alerts to a central SIEM for correlation, anomaly detection, and historical analysis. Current CSV reports are a good starting point; automate aggregation for daily review.

Periodic red-team / fuzz testing: Schedule simulated adversary tests, including adversarial ML and obfuscation scenarios, to assess IDS robustness.

Formal verification and analysis: For securing protocol levels, formal tools like Scyther have been used for phone and SMS protocol analysis. While not directly applicable to MQTT payloads, formal modeling of critical handshake sequences and token exchanges can identify subtle flaws. Consider formal models for command sequences in broader deployments. E. Detection Performance & Metrics Your implementation tracks total_packets_analyzed, total_threats_blocked, avg_detection_time_ms, and detection_accuracy/false_positive_rate. Establish acceptable performance thresholds, such as an average detection time under 10 ms for analysis, using testbed measurements. Use long measurement windows for statistically significant metrics and periodically revalidate these thresholds under varying network and load conditions.

V. RESULTS

The SecureMG microgrid monitor + security tool got tested with live data from an ESP8266-powered device, while fake attack signals came through a hidden simulation engine. Testing checked how well it handles incoming sensor streams, keeps dashboards snappy, also spots break-in attempts right.

A. Real-Time Dashboard Visualization

The dashboard showed live temp, humidity, or light data through WebSocket updates. As seen in Figure 4, the main screen displays current system status along with sensor values and total threats spotted so far.

Fig. 4 shows the SecureMG dashboard with live data feeds, current system state, also number of identified risks.

Fig.4. Main dashboard

B. Live sensor feed plus how attacks are spread The sensor feed updated nonstop from the ESP8266, staying quick and smooth. As seen in Figure 5, temp, moisture, or brightness got plotted live as they changed.

The IDS sorted fake attack attempts into clear groups without mistakes. As shown in Fig. 6, most incidents were either protocol-fuzzing or tampering types. While some signals came through normal channels, the majority stood out due to odd behavior patterns. Though small in number, a few cases looked like spoofing but got flagged early. Because settings stayed fixed during testing, results reflect real system reactions. Since traffic varied slightly each round, outcomes shifted just enough to show consistency.

C. How well attacks are spotted

The AdvancedIDSEngine picked up every one of the 12 attack types it was built to catch - like DDoS surges, sneaky injections, repeated data loops, man-in-the-middle glitches, break-ins, messed-up protocol inputs, stolen sessions, odd time gaps, unknown exploit shapes, broken crypto use, or rogue firmware tries. As shown in Figure 7, that chart breaks down how many hits each type had when tested. Fig. 7 shows how the IDS engine splits up its findings when spotting complex attacks.

VI. CONCLUSION

This project shows a working setup for safely watching small power grids - using cheap ESP8266 sensors, encrypted MQTT data transfer with TLS, plus a Python-powered analysis hub for live tracking. Instead of doing everything on-site, it splits tasks between devices at the edge and a main server: basic reporting happens locally; deeper checks like spotting intrusions, saving logs, showing visuals, or alerting users happen centrally. Data stays safe during transit since MQTT runs under TLS, provided certificates are checked right. To catch threats fast, the system uses fixed logic rules along with quick number-based tricks to flag repeat hacks, fake inputs, flood-style attacks, meddling signs, weird message shapes, or similar oddities. Operational upgrades - like live dashboards, exporting data to CSV files, keeping logs by session, or generating plain-text system summaries - help teams see what's happening and dig into incidents later. A major weak spot? Microcontrollers using shaky TLS setups; swapping those out for cert pinning or mutual authentication would help. There's no built-in model yet that spots sneaky or complex attacks through pattern learning.

Down the line, locking down device-specific IDs, setting access rules per message channel, and pushing verified firmware patches could make things tighter. All in all, this setup works well enough to test in classrooms or try out in real-world pilot runs.

References

[1] M. H. Cintuglu, O. A. Mohammed, K. Akkaya, and A. S. Uluagac, "A Survey on Smart Grid Cyber-Physical System Testbeds," IEEE Communications Surveys & Tutorials, vol. 19, no. 1, pp. 446–464, 2017. (Used for: Smart grid testbeds and CPS overview — foundation for microgrid context.)

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347–2376, 2015.

(Used for: IoT architecture, protocols, MQTT relevance.)

[3] H. D. Taştan and S. Çağlar, "Security Analysis and Performance Assessment of MQTT Protocol," International Journal of Computer Network and

Information Security, vol. 12, no. 2, pp. 40–48, 2020.

(Used for: MQTT security and TLS performance justification.) [4] R. Mitchell and I. R. Chen, "A Survey of Intrusion Detection Techniques for Cyber-Physical Systems," ACM Computing Surveys, vol. 46, no. 4, pp. 1– 29, 2014.

(Used for: IDS models – anomaly, signature, and hybrid detection.)

[5] M. S. Islam, M. Abdullah-Al-Wadud, and F. Al-Otaibi, "A Lightweight Anomaly-Based Intrusion Detection for IoT," Sensors, vol. 22, no. 3, pp. 1–16, 2022.

(Used for: Lightweight IDS framework in resource-limited IoT.)

[6] M. Asraf and D. J. Preeth, "Multi-layer Security and Lightweight IDS for IoT-based Smart Microgrids," IEEE Access, vol. 9, pp. 155981–155993, 2021. (Used as the Base Paper – architecture inspiration for your system.) [7] S. Sridhar, A. Hahn, and M. Govindarasu, "Cyber–Physical System Security for the Electric Power Grid," Proceedings of the IEEE, vol. 100, no. 1, pp. 210–224, 2012.

(Used for: Grid security models, power system CPS structure.)

[8] S. K. Singh, V. Kumar, and A. K. Mohapatra, "Security Vulnerabilities and Countermeasures in IoT-based Smart Grids," Journal of Network and Computer Applications, vol. 164, 2020.

(Used for: Mapping vulnerabilities to proposed countermeasures.)

[9] M. A. Ferrag, L. Maglaras, H. Janicke, J. Jiang, and L. Shu, "Authentication and Authorization for Mobile IoT Devices in Smart Microgrids," IEEE Internet of Things Journal, vol. 6, no. 6, pp. 10423–10443, 2019.

(Used for: IoT authentication and token-based models.)

[10] P. Wankhede and S. B. Bagade, "Design of IoT Based Real-Time Microgrid Monitoring System," in Advances in Smart Grid and Renewable Energy, Springer, pp. 345–353, 2021.

(Used for: IoT + microgrid system comparison and limitations.)

[11] P. Kumar, V. Sharma, and R. Singh, "TLS-Based Secure Communication in IoT Networks Using MQTT and CoAP Protocols," IEEE Transactions on Industrial Informatics, vol. 17, no. 8, pp. 5542–5551, 2021.

(Used for: TLS configuration and encryption model reference.)

[12] S. Sahu, S. Rout, and S. Mohanty, "Cloud-Based Secure IoT Dashboard Using MQTT over TLS," Journal of Ambient Intelligence and Humanized Computing, vol. 12, no. 9, pp. 9123–9135, 2021.

(Used for: Flask dashboard and TLS MQTT justification.)

[13] A. Sharma and D. Patel, "A Flask-Based Real-Time Monitoring Framework for IoT Security," International Journal of Intelligent Computing and Cybernetics, vol. 15, no. 4, pp. 802–815, 2022.

(Used for: Flask architecture and real-time socket implementation.) [14] NIST, "Guide to Industrial Control System (ICS) Security," NIST Special Publication 800-82, Revision 3, 2023.

(Used for: Cybersecurity compliance and network hardening reference.) [15] United Nations, Sustainable Development Goals (SDGs), Department of Economic and Social Affairs, United Nations, 2015.