



Blockchain-Enabled Zero Trust Architecture for Privacy-Preserving Cybersecurity in Industrial IoT Environments

Dr. Naresh Thoutam ¹, Dr. Ankita V. Karale ¹,
Shivam Nagendra Yadav ¹, Mayuri Vijay Nikam ¹,
Nitin Mansingh Bhole ¹, Pratik Shivaji Darvesh ¹

¹Department of Computer Engineering, Sandip Institute of Technology and Research Centre, Nashik, India

naresh.thoutam@sitrc.org, ankita.karale@sitrc.org, shivamyadav81808@gmail.com, nikamayuri11001@gmail.com,
bholenitin324@gmail.com, pratikdarvesh2@gmail.com

Corresponding author- Dr. Naresh Thoutam

DOI: 10.5281/zenodo.23633299

Abstract: Industrial Internet of Things (IIoT) deployments interconnect sensors, programmable logic controllers, supervisory control and data acquisition nodes, industrial gateways, databases, and administrative applications. This connectivity improves automation and operational visibility, but it also enlarges the attack surface and exposes industrial environments to stolen credentials, compromised devices, replayed requests, excessive privilege, and alteration of centralized security logs. This paper presents a synopsis-aligned blockchain-enabled Zero Trust architecture for privacy-preserving cybersecurity in software-simulated IIoT environments. A FastAPI gateway acts as the policy decision and policy enforcement point and evaluates every request using device registration, device state, JSON Web Token validity, role permission, timestamp freshness, endpoint authorization, and payload plausibility. Authorized telemetry is retained in MongoDB for efficient querying, whereas compact security events are written through Web3.py to a Solidity smart contract deployed on a local Ganache Ethereum network. A React dashboard provides telemetry monitoring, device and role administration, revocation, security alerts, and blockchain-backed audit verification. The architecture deliberately separates high-volume operational data from tamper-evident audit metadata, thereby reducing unnecessary on-chain disclosure and transaction overhead. The paper specifies the system model, formal access rule, on-chain/off-chain data model, algorithms, threat-control mapping, implementation plan, and a reproducible evaluation methodology for blocking effectiveness, false acceptance, false rejection, latency, throughput, blockchain overhead, and tamper detection. No unmeasured experimental performance claims are reported.

Keywords: Blockchain, Industrial Internet of Things, Zero Trust architecture, privacy preservation, FastAPI, MongoDB, Solidity, Ganache, Web3.py, JWT, RBAC, tamper-evident audit.

I INTRODUCTION

Industrial systems increasingly depend on connected sensing, control, and monitoring components. Temperature and pressure sensors, programmable logic controllers (PLCs), supervisory control and data acquisition (SCADA) nodes, industrial cameras, and gateways exchange operational data with local servers, edge platforms, and cloud applications. The resulting visibility supports predictive maintenance, process optimization, and rapid incident response. At the same time, convergence between operational technology and conventional information technology creates new paths for identity abuse, lateral movement, unauthorized commands, data manipulation, and concealment of forensic evidence. OT security guidance therefore stresses that industrial environments must preserve

availability and safety while improving identity, network, and audit controls [15].

Traditional perimeter security gives disproportionate weight to network location. Once a user or device is admitted to an internal segment, subsequent requests may receive broad or long-lived access. That assumption is unsafe when a device has been cloned, a token has been stolen, an internal account is misused, or an endpoint that was previously healthy becomes compromised. Zero Trust Architecture (ZTA) removes implicit trust based on ownership or location and requires explicit authentication and authorization of the subject and device before access to a protected resource [9]. For IIoT, this principle must be translated into request-level checks that are

computationally practical and understandable to administrators.

A separate weakness arises from editable security logs. Access decisions, policy changes, revocations, and suspicious-device alerts are commonly stored in the same database that holds operational records. An attacker or privileged insider who acquires database permissions may alter or delete these entries, making incident reconstruction incomplete. Blockchain can provide an append-only and independently verifiable record, but placing all sensor data on-chain is neither privacy-preserving nor efficient. Prior research consistently identifies latency, storage, consensus, and resource constraints as central challenges when blockchain is integrated with IoT [12], [21]-[24]. Consequently, blockchain should be applied selectively to security-critical evidence rather than treated as a universal database.

This paper presents a practical architecture that follows the approved project synopsis. The target implementation uses simulated industrial devices, a FastAPI backend, MongoDB, Solidity, Ganache, Web3.py, and a React dashboard. The design is intentionally different from a generic login prototype based on SQLite or a custom Python hash chain. It models a complete request path from IIoT telemetry generation to Zero Trust policy enforcement, operational storage, alert creation, blockchain transaction submission, and administrative verification.

The principal contributions are: 1) a strict request-level policy that combines user or service identity, device identity, token claims, role permission, request freshness, and context validation; 2) a five-layer full-stack architecture aligned with the synopsis technology stack; 3) a privacy-preserving on-chain/off-chain data split in which telemetry remains in MongoDB and only compact audit metadata is anchored by a Solidity contract; 4) explicit handling of both successful and failed requests, policy changes, and access revocations; 5) a threat-oriented evaluation plan that separates API decision latency from blockchain receipt latency; and 6) a candid treatment of the limits of a local Ganache deployment without presenting it as a production distributed ledger.

II. BACKGROUND AND RELATED WORK

A. Security Characteristics of IIoT and OT

IIoT differs from general consumer IoT because cyber events can affect physical processes, equipment, production continuity, and human safety. OT assets may operate for long lifecycles, use legacy protocols, and prioritize deterministic behavior over frequent software changes. Sadeghi et al. identify device heterogeneity, insecure legacy components, weak update mechanisms, and the convergence of industrial systems with Internet-connected services as important security and

privacy challenges [17]. Humayed et al. similarly show that cyber-physical systems must be analyzed across cyber, physical, and cyber-physical attack surfaces rather than through network security alone [18].

The proposed framework does not attempt to replace safety systems, process control logic, or network segmentation. Instead, it protects the application-layer path through which simulated devices submit telemetry and administrators manage identities and policies. NISTIR 8259A identifies device identification, configuration, data protection, interface access control, software update, and cybersecurity state awareness as core IoT capabilities [16]. These capabilities motivate the device registry, active/revoked status, constrained interfaces, audit events, and dashboard state used in the design.

B. Zero Trust for IoT and IIoT

NIST SP 800-207 defines logical components including a policy engine, policy administrator, and policy enforcement point, and emphasizes per-session decisions, least privilege, and continuous evaluation [9]. IEEE Std 3219-2023 applies blockchain-assisted Zero Trust concepts to IoT access control and provides an implementation framework for interactions among heterogeneous entities [3]. In the proposed prototype, FastAPI combines the logical policy decision and enforcement responsibilities at one gateway while maintaining distinct software modules for authentication, authorization, context evaluation, and action execution.

Son et al. propose a Zero Trust authentication scheme with access control for 6G-enabled IoT and demonstrate that identity verification and authorization must be considered together [2]. Aleisa integrates blockchain, Zero Trust, and privacy mechanisms in an IoT cybersecurity architecture [1]. Bicer et al. use blockchain-backed request history to extend Zero Trust at the edge [5]. These works support repeated verification and immutable evidence, but their system assumptions and cryptographic complexity do not directly prescribe the full-stack implementation required by the present synopsis.

C. Blockchain for IoT Security and Access Control

Blockchain-IoT research covers access control, trust management, secure gateways, data integrity, and distributed audit. Christidis and Devetsikiotis explain how smart contracts can mediate interactions among IoT entities [24]. Novo proposes a scalable blockchain-based access-management architecture [25], while Zhang et al. define smart-contract-based access control with registration, policy, and judgment logic [26]. These studies motivate the use of a contract as a security-event interface, although the proposed framework keeps the primary allow/deny decision in FastAPI so that industrial context rules can be changed without redeploying the contract.

Lightweight and gateway-oriented designs are particularly relevant. Maroufi et al. address blockchain overhead for 5G-enabled IoT [4]. Debe et al. analyze trustworthy blockchain gateways for resource-constrained clients [8]. Dorri et al. identify scalability and bandwidth constraints and later propose a tiered lightweight scalable blockchain design [14], [30]. Hasan et al. combine blockchain with external storage for trustworthy IoT streaming, illustrating why large data streams should not be placed entirely on-chain [6]. Javaid et al. use blockchain in a trust model for wireless sensor IoT [7]. Shala et al. investigate decentralized IoT service provision and end-user trust [11].

Broader reviews show both the promise and cost of this integration. Reyna et al. organize blockchain-IoT opportunities and challenges [12]; Dai et al., Makhdoom et al., and Panarello et al. discuss scalability, privacy, interoperability, resource limits, and consensus overhead [21]-[23]. Khan and Salah review blockchain solutions for IoT security [19], while Sengupta et al. focus on attacks and blockchain countermeasures for IoT and IIoT [20]. Ferrag et al. survey authentication protocols and blockchain-related privacy challenges [10], [13]. Collectively, the literature supports selective blockchain use, explicit device authentication, and careful overhead measurement rather than claims that blockchain automatically solves every IIoT security problem.

Table 1: Comparison of security approaches

Approach	Request verification	Audit evidence	Prototype fit
Perimeter login + DB	Initial identity; limited device/context re-check	Editable centralized records	Simple but weak after compromise
Zero Trust without blockchain	Repeated identity, device, and policy checks	Tamper evidence not inherent	Strong access control; mutable logs
Blockchain-only IoT access	Depends on contract and identity design	Strong append-only evidence	Can be heavy for context-rich checks
Proposed architecture	JWT + device + role + freshness + payload context	Solidity events linked to off-chain records	Full-stack, synopsis-aligned prototype

D. Research Gap

Existing studies establish the value of Zero Trust, blockchain auditability, and IoT access control, but a gap remains between conceptual frameworks and a reproducible student-scale IIoT implementation. The required system must show how simulated PLC, SCADA, gateway, and sensor requests are processed; how role and context policies are evaluated; how

authorized telemetry and denied requests are stored differently; how smart-contract events are linked to database records; and how administrators verify the complete security state from one dashboard. The proposed work addresses this engineering gap without claiming a new consensus protocol or replacing industrial safety controls.

III. REQUIREMENTS, SYSTEM MODEL, AND THREAT MODEL

A. Functional Requirements

The client layer shall provide independent Python scripts for temperature, pressure, humidity, gas, PLC, SCADA, camera, and industrial-gateway profiles. Each client shall submit a device identifier, UTC timestamp, structured payload, and bearer token. A test configuration shall allow deterministic activation of invalid tokens, unknown identifiers, stale timestamps, revoked-device requests, malformed payloads, and out-of-range measurements.

The backend shall expose REST endpoints for token issuance, device registration, telemetry ingestion, alert retrieval, role and policy management, device revocation, and audit-event queries. Every protected endpoint shall invoke the same policy pipeline before business logic. Authorized telemetry shall be written to MongoDB. Denied requests shall create an operational alert and a blockchain security event. Administrative changes to roles, device status, and policy versions shall also be auditable.

The dashboard shall display active and revoked devices, latest telemetry, alert severity, request outcomes, policy versions, blockchain transaction identifiers, and contract events. It shall provide controlled actions for role assignment, revocation, and alert resolution. Dashboard operations that alter security state shall be protected by administrator authorization and shall generate their own audit events.

B. Security and Privacy Requirements

The security objectives are default deny, least privilege, bounded token lifetime, explicit device state, replay resistance through freshness checks, endpoint-specific authorization, tamper-evident audit records, and traceable administrator actions. The privacy objective is data minimization: passwords, complete JWTs, raw sensor streams, personally identifying attributes, and facility-sensitive descriptions shall not be written to the blockchain. The contract shall store only the minimum metadata required to establish that a decision occurred and to verify the integrity of a related off-chain record.

JWT processing shall follow the format defined in RFC 7519 [27] and the deployment cautions in RFC 8725 [28]. The gateway must explicitly select allowed algorithms, validate issuer and audience where configured, enforce expiration,

reject missing mandatory claims, and avoid trusting unverified header values. Device secrets shall be stored as derived values rather than plaintext. In a production extension, device credentials should be replaced by certificates or hardware-backed identities.

C. Entities and Protected Assets

The entities are simulated IIoT devices, human users, administrators, the FastAPI gateway, MongoDB, the Ganache Ethereum network, a Solidity audit contract, and the React dashboard. Protected assets include telemetry integrity, authentication tokens, device credentials, role and policy definitions, device active/revoked state, security alerts, blockchain account keys, contract deployment information, and the audit trail. Each device record contains a pseudonymous identifier, type, owner or logical zone, role, active state, credential reference, expected metric schema, and acceptable operating range.

D. Adversary Model and Trust Assumptions

The adversary may submit incorrect passwords, stolen or modified JWTs, forged device identifiers, invalid device secrets, stale requests, unauthorized endpoint calls, malformed payloads, and values outside configured physical ranges. The adversary may repeat failures to probe policies or overwhelm alerting. A database-level adversary may edit or delete MongoDB telemetry or alerts. A privileged but incomplete insider may attempt to change device roles or revoke evidence. The prototype trusts the host operating system, FastAPI policy code, cryptographic libraries, contract bytecode, and the initial protection of blockchain private keys. Full compromise of the host, denial of service at network scale, malware inside trusted code, sensor hardware manipulation, and compromise of every blockchain copy are outside scope.

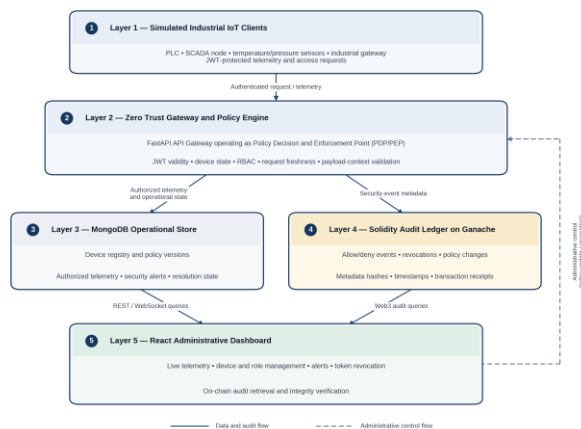


Figure 1: Five-layer blockchain-enabled Zero Trust architecture for Industrial IoT.

Table 2: Request fields and policy checks

Request field	Validation performed	Failure outcome
Authorization header	Signature, algorithm, expiry, issuer/audience, required claims	401; token-failure event
device_id	Registered, active, expected role/owner, credential match	403; device-alert event
timestamp	UTC format and absolute clock skew within policy window	403; replay/stale-request event
endpoint/action	RBAC matrix permits role to invoke operation	403; authorization event
payload	Schema, type, unit, range, and basic rate-of-change checks	422/403; context alert
request_id	Unique identifier for cross-store correlation	Reject duplicate or generate server ID

IV. PROPOSED FIVE-LAYER ARCHITECTURE

A. Layer 1: Simulated IIoT Devices

Independent Python clients emulate industrial assets without requiring physical hardware. A temperature client may send a scalar value and unit; a PLC or SCADA profile may send a compact JSON object containing process state, set point, and alarm flag. Each client obtains or is provisioned with a bounded token and includes its device identifier and source timestamp in every message. The simulator separates normal and adversarial modes so that each security test can be reproduced with known inputs.

Device simulation is not presented as equivalent to a real industrial testbed. It is a controlled means to validate the API, policy, storage, smart-contract, and dashboard integration. Physical-device extensions would need protocol adapters, reliable clock synchronization, certificate provisioning, safety interlocks, and tests that do not interfere with production processes.

B. Layer 2: FastAPI Zero Trust Gateway

The gateway is the sole application entry point for protected resources. Middleware extracts a correlation identifier, records receipt time, limits request size, and invokes authentication. The policy module then loads device state and endpoint policy, verifies the role, checks timestamp freshness, and evaluates payload rules. Business logic is reached only after all

mandatory checks pass. This structure maps the policy decision and enforcement concepts of ZTA to a practical prototype [9].

Centralizing the policy path avoids inconsistent security behavior across endpoints. However, the implementation should keep authentication, authorization, context validation, storage, and blockchain submission as separate modules with explicit interfaces. Separation improves testability and allows later replacement of JWT or RBAC without rewriting telemetry handling.

C. Layer 3: MongoDB Operational Storage

MongoDB stores data that must be queried, filtered, plotted, and updated efficiently. The devices collection contains registration and policy attributes. The telemetry collection stores authorized measurements and processing metadata. The system_alerts collection stores denied-request details, severity, reason code, resolution state, and the associated blockchain transaction identifier when available. Additional collections may hold token revocations and policy versions.

Indexes should support device_id with received_at for telemetry, active and role for device administration, unresolved and severity for alerts, and request_id for cross-store correlation. High-volume telemetry requires retention or archival policies. Security alerts should have longer retention than ordinary telemetry because they support forensic review.

D. Layer 4: Solidity Audit Contract on Ganache

The contract exposes a restricted recordSecurityEvent function. Its inputs represent a pseudonymous device identifier, event type, decision status, request identifier, policy version, and hash of selected off-chain metadata. The contract emits a LogRecorded event containing the device identifier, request identifier, block timestamp, and event type. Contract access should be limited to an authorized gateway account so that arbitrary clients cannot manufacture audit records.

Ganache provides deterministic local accounts and rapid transaction processing for development. It enables demonstration of contract deployment, transaction receipts, event retrieval, and resistance to ordinary database editing. It does not reproduce distributed consensus, independent validators, or Byzantine fault tolerance. This limitation must remain visible in both the paper and experimental interpretation.

E. Layer 5: React Administrative Dashboard

The dashboard consumes backend APIs rather than connecting directly to privileged databases. Telemetry charts show recent authorized values. A device panel displays identity, type, role, last-seen time, and active state. A policy panel allows authorized administrators to update role mappings and operating ranges. An alert panel exposes reason, severity,

source, and resolution. An audit panel reads contract events through the backend and compares the on-chain metadata hash with a recomputed hash of the corresponding MongoDB record.

V. ZERO TRUST DECISION MODEL AND WORKFLOW

A. Formal Request Rule

For request r , let $U(r)$ denote a valid user or service identity, $D(r)$ a registered and active device with valid credential binding, $J(r)$ a valid JWT, $P(r)$ permission for the requested endpoint and action, $T(r)$ timestamp freshness, $C(r)$ contextual payload validity, and $R(r)$ absence of an active revocation. The verification predicate is:

$$V(r) = U(r) \wedge D(r) \wedge J(r) \wedge P(r) \wedge T(r) \wedge C(r) \wedge R(r).$$

The access decision is Allow only when $V(r)=1$; otherwise it is Deny. The conjunction prevents a valid token from compensating for an unknown device, prevents a registered device from bypassing role restrictions, and prevents an authorized role from submitting stale or implausible telemetry. Each term has a distinct reason code so that administrators can distinguish authentication failure from policy or context failure.

B. JWT and Identity Verification

JWT validation includes cryptographic signature verification, algorithm allow-listing, expiration, not-before time where used, issuer, audience, subject, token identifier, and role or scope claims. The gateway must not accept an algorithm based solely on an attacker-controlled header. Short-lived access tokens reduce the period during which a stolen token remains useful. Revocation can be implemented through device state, user account state, or a token-identifier denylist, recognizing that stateless tokens require an external revocation decision when immediate invalidation is required [27], [28].

C. Device and Context Verification

Device verification checks registration, active state, logical owner or zone, credential binding, and compatibility between device type and endpoint. Context validation converts a syntactically valid request into an operationally meaningful request. A temperature device, for example, must provide the expected metric, numeric type, unit, plausible range, and acceptable timestamp. Basic rate-of-change checks can flag abrupt transitions. These rules are configuration-driven and are not a substitute for process-aware anomaly detection.

D. Decision and Audit Workflow

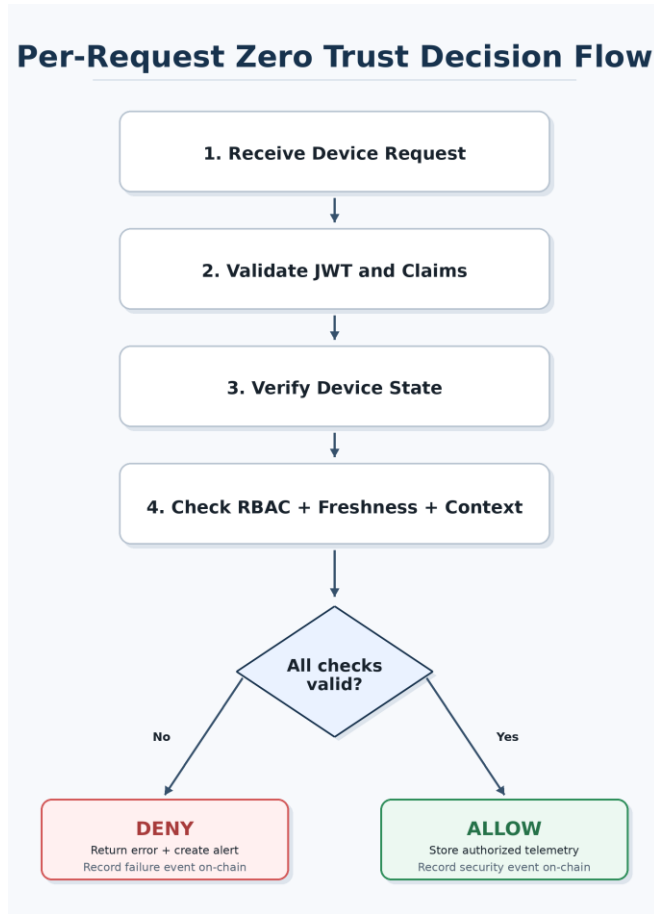


Figure 2: Request verification, storage, alert, and audit workflow.

A request first receives a correlation identifier. The gateway validates token and device state, then evaluates role, freshness, and payload context. An allowed request is written to telemetry storage and generates a compact success event if success auditing is enabled. A denied request is not written as normal telemetry; instead, an alert record is created and a failure event is submitted to the contract. The HTTP response can be returned before or after blockchain confirmation depending on the selected consistency mode. For a classroom prototype, synchronous confirmation simplifies traceability; for higher throughput, an authenticated queue can decouple the user response from transaction submission.

E. Verification Algorithm

Algorithm 1: Zero Trust request processing

Input: request r , endpoint a ; Output: HTTP decision, operational record, audit event

1. Create or validate request_id and record server receipt time.
2. Parse bearer token; verify signature, algorithm, expiry, issuer/audience, and claims.

3. Load user/service and device records; verify active state and credential binding.
4. Check RBAC permission for role, endpoint, method, and requested action.
5. Validate source timestamp against the configured freshness window.
6. Validate payload schema, metric type, unit, range, and optional rate of change.
7. If every check succeeds, store telemetry and set decision = ALLOW; otherwise create alert and set decision = DENY.
8. Canonicalize selected metadata and compute metadata_hash.
9. Submit recordSecurityEvent(...) through Web3.py and capture transaction receipt.
10. Update the operational record with transaction hash and return the response.

VI. BLOCKCHAIN AUDIT AND PRIVACY-PRESERVING DATA DESIGN

A. On-Chain/Off-Chain Separation

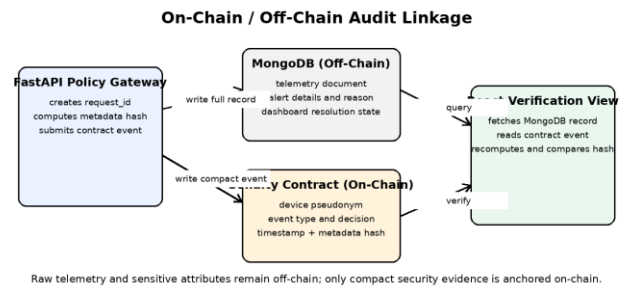


Figure 3: On-chain and off-chain audit linkage with data minimization.

Raw IIoT telemetry can reveal operating cycles, production conditions, alarm patterns, and infrastructure details. Writing it permanently to a blockchain would increase disclosure risk and create avoidable storage and transaction cost. The proposed design therefore stores raw measurements and detailed alerts in MongoDB, while the contract receives only compact event metadata. This follows the broader principle demonstrated in blockchain-assisted IoT streaming that large content should remain external while integrity evidence is anchored on-chain [6].

The metadata hash is computed from a canonical representation of selected fields such as request_id, pseudonymous device_id, event_type, decision, policy_version, server timestamp, and reason code. Canonicalization is essential: field order, string encoding, timestamp representation, and null handling must be stable. A dashboard verifier reconstructs the same representation from MongoDB, hashes it, and compares the result with the contract event.

B. Smart Contract Interface and Events

A minimal contract reduces attack surface and makes review easier. The gateway account is registered as an authorized writer. The `recordSecurityEvent` function validates non-empty identifiers, records or emits the event, and returns through a transaction receipt. Events are preferred for sequential audit retrieval because they can be filtered by indexed fields. The contract does not store passwords, complete tokens, payload values, personal information, or mutable alert-resolution notes. Policy updates and device revocations use distinct event types so that security-state changes can be reconstructed chronologically.

C. Integrity Verification

Verification has two levels. First, the dashboard checks that a transaction receipt succeeded and that the expected contract address emitted a matching event. Second, it recomputes the metadata hash from the operational record. Editing the MongoDB reason, decision, timestamp, or device identifier changes the recomputed digest and produces a mismatch. Deleting the operational row leaves an on-chain event without a corresponding detailed record, which is itself an observable inconsistency. Conversely, a blockchain event without an authorized gateway signature or from an unexpected contract address is rejected by the verifier.

D. Privacy Limitations

Pseudonymous identifiers do not guarantee anonymity. Repeated timestamps and event patterns may reveal device activity, a risk also observed in blockchain-IoT privacy studies [14], [30]. The prototype reduces this risk by avoiding raw payloads and personal attributes, but a production design may need rotating pseudonyms, access-controlled ledger views, batching, or a permissioned network. The ledger should not be described as privacy-preserving merely because it omits a human name; data-linkage risks must be assessed against the actual deployment.

VII. IMPLEMENTATION METHODOLOGY

A. Device Simulator

Each simulator is configured with `device_id`, device type, API base URL, token source, normal sampling interval, and expected metric profile. Normal operation produces reproducible values within defined ranges. Attack flags alter one property at a time so that failures can be attributed: `invalid_token`, `expired_token`, `unknown_device`, `revoked_device`, `stale_timestamp`, `wrong_role`, `malformed_payload`, `out_of_range`, `duplicate_request_id`, and `burst_mode`. Logging at the client records send time, response code, and server request identifier.

B. FastAPI Services

The backend is divided into authentication, device registry, policy engine, telemetry, alert, blockchain, and dashboard-query modules. Pydantic models enforce request schemas before business processing. Dependency injection applies authentication and role checks consistently. Policy rules are versioned so that each decision identifies the rule set used. Error responses avoid leaking whether a specific account or device exists, while server-side reason codes preserve diagnostic detail for authorized administrators.

C. MongoDB Schema and Consistency

Telemetry documents contain `request_id`, `device_id`, `source_timestamp`, `received_at`, `metric_type`, `value` or structured payload, `unit`, `policy_version`, `decision`, and blockchain transaction hash where applicable. Alert documents contain `reason_code`, `severity`, device and user references, `endpoint`, client metadata, `policy_version`, `metadata_hash`, transaction hash, and resolution information. The application writes the MongoDB record before contract submission, then updates it with the receipt. A failed transaction produces a pending-audit state and retry entry rather than silently losing evidence.

D. Web3.py, Solidity, and Ganache

The blockchain service loads the compiled application binary interface and deployed contract address from configuration rather than hard-coding them in request handlers. The gateway account private key is loaded from an environment secret for the laboratory prototype. Transaction construction specifies chain identifier, nonce, gas estimate with a bounded margin, and destination contract. Receipt handling captures status, block number, transaction hash, and emitted event. Tests cover unauthorized contract callers, malformed arguments, transaction failure, and duplicate application retries.

E. React Dashboard

The frontend separates monitoring from administration. Monitoring views use read-only APIs for telemetry, alerts, and audit events. Administrative views require an administrator token and provide confirmation before revocation or policy changes. Charts should not imply data continuity when values were denied; rejected samples appear in the alert timeline rather than the authorized telemetry series. Audit verification displays four states: verified match, transaction pending, off-chain record missing, and hash mismatch.

F. Deployment and Secret Management

The prototype can run as separate local processes or containers: MongoDB, Ganache, FastAPI, React, and device clients. Configuration includes token keys, database URI, contract address, gateway account, allowed clock skew, and device-

range policies. Secrets shall not be committed to Git. Development keys and Ganache mnemonics are unsuitable for public or production networks. TLS termination, certificate management, backup protection, rate limiting, and process isolation are deployment requirements beyond a local demonstration.

Table 3: Prototype modules and responsibilities

Module	Primary responsibility	Representative technology
Device simulator	Normal and adversarial IIoT requests	Python clients
Policy gateway	JWT, device, RBAC, freshness, and context checks	FastAPI / Pydantic
Operational store	Devices, telemetry, alerts, revocations, policies	MongoDB
Audit service	Contract calls, receipts, event queries, hash verification	Web3.py
Immutable ledger	Security-event recording and emission	Solidity on Ganache
Administrative UI	Monitoring, policy management, revocation, audit view	React.js / Tailwind CSS

VIII. SECURITY ANALYSIS

A. Credential and Token Attacks

Incorrect passwords fail identity verification and produce a failed-login event without storing plaintext credentials. A modified JWT fails signature validation; an expired token fails the time check; a token with the wrong audience or missing claim is rejected before protected business logic. Short token lifetime limits exposure but does not eliminate theft during validity. Immediate response therefore also depends on account, device, or token-identifier revocation. RFC 8725 guidance is relevant because common JWT vulnerabilities arise from algorithm confusion, weak validation rules, and inappropriate trust in token-supplied values [28].

B. Device Impersonation and Revocation

A valid user token is insufficient when the device identifier is unknown, inactive, or incompatible with the claimed role. Credential binding prevents an attacker from simply copying a registered identifier. Revocation changes the operational device state and emits a blockchain event. Subsequent requests fail R(r) or D(r). A stolen shared secret remains a limitation;

certificate-based mutual authentication and hardware-protected keys are stronger future controls.

C. Authorization, Replay, and Context Abuse

RBAC restricts a device or administrator to the endpoints and actions assigned to its role. A sensor cannot invoke policy administration, and a monitoring account cannot revoke devices. Timestamp freshness blocks requests captured outside the accepted window, although robust replay defense should also include nonces or sequence numbers. Payload validation rejects malformed types and physically implausible values. Because simple ranges can be evaded by gradual or coordinated attacks, context rules should be treated as a baseline rather than complete anomaly detection.

D. Centralized Log Tampering

Editing a MongoDB alert does not alter the blockchain event. Verification identifies changed fields through a hash mismatch. Deleting a record creates an unmatched on-chain event. Editing only a non-hashed resolution note does not invalidate the original event, which is intentional because incident handling state must remain updateable. The selection of hashed fields should therefore cover the immutable evidence while excluding legitimate mutable workflow fields.

E. Blockchain and Host Limitations

A local Ganache chain is tamper-resistant against ordinary application users but not against an administrator who controls the entire host, chain data, private keys, application, and backups. Wüst and Gervais emphasize that blockchain is appropriate only when its trust and writer assumptions justify the added complexity [29]. The current deployment is a functional audit prototype. Production-grade immutability requires independent validators or replicas, controlled governance, signed checkpoints, protected keys, and a recovery process.

IX. EVALUATION METHODOLOGY

A. Experimental Configurations

Evaluation begins only after the integrated prototype is executable. Configuration B0 is a baseline with JWT authentication and MongoDB logging but without repeated device/context checks or blockchain submission. Configuration ZT enables device, RBAC, freshness, revocation, and payload checks but does not submit contract transactions. Configuration ZT+BC enables the complete framework. Comparing B0, ZT, and ZT+BC separates the cost of policy enforcement from the cost of blockchain auditing.

Table 4: Threat-control mapping

Threat / failure	Control	Expected observable result
Unknown or revoked device	Registry, credential binding, active-state check	403; device alert and contract event
Modified or expired JWT	Signature and temporal claim validation	401; token-failure event
Role violation	Endpoint/action RBAC matrix	403 despite valid identity
Stale or replayed request	Timestamp window; future nonce extension	Request denied and replay reason logged
Implausible telemetry	Schema/range/rate validation	Telemetry not stored as authorized data
MongoDB record edited	On-chain metadata hash comparison	Verification mismatch
Operational record deleted	On-chain event reconciliation	Missing-record warning
Complete host compromise	Outside prototype trust boundary	Requires independent ledger nodes/backups

B. Security Test Cases

Deterministic tests include valid user and device, wrong password, unknown device, revoked device, expired token, modified token, wrong audience, missing role, insufficient role, stale timestamp, duplicate request identifier, malformed payload, out-of-range value, repeated failures, database-record modification, record deletion, unauthorized contract caller, and transaction failure. Each test specifies expected HTTP result, MongoDB state, contract event, dashboard alert, and verification state. Negative tests are repeated after policy updates to ensure that cached decisions do not bypass current state.

C. Performance Workloads

Workloads include steady normal traffic, burst traffic, mixed valid and malicious requests, and alert-heavy traffic. Device counts may be increased in controlled stages such as 10, 25, 50, and 100 simulated clients while request intervals remain fixed. Tests should report hardware, operating system, Python and Node versions, MongoDB version, Ganache version, contract compiler version, duration, warm-up, repetitions, and whether blockchain confirmation is synchronous. Device generation and server measurement clocks should be distinguished to avoid inaccurate latency calculations.

D. Metrics

Let N_m be the number of malicious requests and N_b the number correctly blocked. Attack-blocking rate is $ABR=(N_b/N_m) \times 100$. False acceptance rate is

$FAR=(\text{unauthorized requests accepted} / \text{unauthorized requests}) \times 100$. False rejection rate is $FRR=(\text{authorized requests rejected} / \text{authorized requests}) \times 100$. Authentication or request success rate is reported only for the valid workload. For N requests, average latency is the arithmetic mean of response time, accompanied by median and 95th-percentile values. Throughput is processed requests divided by experiment duration.

Blockchain overhead is measured as the difference between otherwise identical ZT+BC and ZT requests. Two values are required: submission overhead, ending when the transaction is accepted by the local node, and confirmation overhead, ending when a successful receipt is obtained. Additional measurements include CPU and memory utilization, MongoDB storage growth, alert count, blockchain transaction count, contract gas usage, retry count, and verification time for a batch of audit records.

E. Validity and Reporting Rules

Each scenario should be repeated sufficiently to expose variability, with raw measurements retained for review. Results must not be replaced by qualitative labels such as “high accuracy” or “low latency.” A zero FAR in deterministic tests means only that the implemented attack cases were blocked; it does not establish universal security. Likewise, Ganache gas and confirmation behavior cannot be generalized to a public or permissioned production chain. The paper should report failures, timeouts, and missing events rather than excluding them silently.

X. DISCUSSION

A. Why Blockchain Is Limited to Audit Metadata

The proposed design does not use blockchain because every IIoT application requires one. It uses a contract to create an independently verifiable sequence of security decisions and changes. MongoDB remains better suited to high-rate telemetry, filtering, updates, and chart queries. This division is consistent with the literature that identifies blockchain overhead and IoT resource constraints [21]-[23]. It also meets the privacy objective by preventing permanent publication of complete operational streams.

B. Policy Placement

Keeping policy evaluation in FastAPI allows rapid changes to device ranges, roles, and endpoint rules. A contract-only decision engine would make context-rich policies more expensive to update and could expose additional operational detail. The trade-off is that the gateway remains a trusted decision component. Blockchain detects inconsistency in retained evidence; it does not prove that every decision was

correct. Independent policy testing and signed policy versions are therefore important.

Table 5: Deterministic tests and expected outcomes

Scenario	Expected access result	Expected audit / dashboard result
Valid token, active device, permitted role, valid context	Allow	Telemetry stored; success event or configured audit
Unknown or revoked device	Deny	Device alert; deny event
Expired, modified, or wrong-audience JWT	Deny	Token alert; deny event
Insufficient role	Deny	Authorization alert; deny event
Stale timestamp or duplicate request	Deny	Replay alert; deny event
Malformed or implausible payload	Deny	Context alert; no authorized telemetry
MongoDB security record modified	Not applicable	Hash verification fails
Contract transaction failure	Policy decision retained	Pending-audit state and retry recorded

C. Alignment With Standards and Guidance

The architecture follows the ZTA separation of policy decision and enforcement concepts in NIST SP 800-207 [9], uses the blockchain-assisted IoT access-control framing of IEEE Std 3219-2023 [3], and recognizes the availability and safety constraints emphasized for OT in NIST SP 800-82 Rev. 3 [15]. Device registration, state awareness, restricted interfaces, and data protection are consistent with the NISTIR 8259A capability baseline [16]. These alignments provide design discipline, but the prototype should not claim formal compliance without a requirement-by-requirement assessment.

D. Practical Value

The resulting system is useful as an academic demonstrator because every important decision is visible: a simulator generates a request, the gateway exposes the reason for allow or deny, MongoDB holds operational detail, the contract holds compact evidence, and the dashboard reconciles both views. This makes it possible to teach Zero Trust, API security, blockchain integration, and incident traceability through one reproducible project rather than through disconnected demonstrations.

XI. LIMITATIONS AND FUTURE WORK

The present design uses simulated devices, a centralized gateway, rule-based context checks, JWTs, and a single-host Ganache network. It does not implement industrial protocols, distributed consensus, safety certification, process-aware anomaly detection, hardware roots of trust, or enterprise key management. Range checks may reject legitimate exceptional states or miss coordinated attacks. Synchronous blockchain confirmation can reduce throughput. MongoDB and FastAPI remain operational single points unless replicated. The dashboard itself introduces an administrative attack surface.

Future work should integrate physical sensors or a safe PLC/SCADA testbed, mutual TLS and certificate-based device identity, nonces or monotonic sequence numbers, automated key rotation, signed policy packages, behavioural trust scoring, process-aware anomaly detection, rate limiting, distributed permissioned validators, independent ledger checkpoints, secure queues for audit submission, and formal threat modeling. Privacy extensions may use controlled ledger membership, rotating pseudonyms, batching, or selective-disclosure techniques after the core system has been implemented and measured.

XII. CONCLUSION

This paper presented a full-length, synopsis-aligned blockchain-enabled Zero Trust architecture for privacy-preserving cybersecurity in Industrial IoT environments. The framework integrates simulated PLC, SCADA, gateway, and sensor clients with a FastAPI policy gateway, MongoDB operational storage, a Solidity smart contract on Ganache, Web3.py, and a React administrative dashboard. Every protected request is evaluated using token, identity, device, role, revocation, freshness, and context checks. Authorized telemetry remains off-chain for efficient querying, while compact security decisions, policy changes, and revocations are anchored on-chain for tamper-evident accountability. The paper defines implementable modules, formal rules, algorithms, threat controls, and a reproducible evaluation plan, while explicitly avoiding fabricated numerical results and overstating the security of a local blockchain. The architecture therefore provides a technically coherent basis for the proposed academic project and a clear path toward certificate-based, distributed, and behaviour-aware IIoT security.

XIII REFERENCE

- [1] M. A. Aleisa, "Blockchain-Enabled Zero Trust Architecture for Privacy-Preserving Cybersecurity in IoT Environments," IEEE Access, vol. 13, pp. 18660-18676, 2025, doi: 10.1109/ACCESS.2025.3529309.
- [2] S. Son, D. Kwon, S. Lee, H. Kwon, and Y. Park, "A Zero-Trust Authentication Scheme With Access Control for 6G-

- Enabled IoT Environments,” IEEE Access, vol. 12, pp. 154066-154079, 2024.
- [3] IEEE Computer Society, IEEE Standard for Blockchain-Based Zero-Trust Framework for the Internet of Things (IoT), IEEE Std 3219-2023, Apr. 2024.
- [4] M. Maroufi, R. Abdoole, B. Mozaffari Tazekand, and S. A. Mortazavi, “Lightweight Blockchain-Based Architecture for 5G Enabled IoT,” IEEE Access, vol. 11, pp. 60223-60239, 2023, doi: 10.1109/ACCESS.2023.3284471.
- [5] C. Bicer, I. Murturi, P. K. Donta, and S. Dustdar, “Blockchain-Based Zero Trust on the Edge,” in Proc. 2023 Int. Conf. Computational Science and Computational Intelligence (CSCI), 2023, pp. 1006-1013.
- [6] H. R. Hasan, K. Salah, I. Yaqoob, R. Jayaraman, S. Pesic, and M. Omar, “Trustworthy IoT Data Streaming Using Blockchain and IPFS,” IEEE Access, vol. 10, pp. 17707-17721, 2022, doi: 10.1109/ACCESS.2022.3149312.
- [7] N. Javaid et al., “A Secure and Efficient Trust Model for Wireless Sensor IoTs Using Blockchain,” IEEE Access, vol. 10, pp. 4568-4579, 2022, doi: 10.1109/ACCESS.2022.3140401.
- [8] M. Debe, K. Salah, R. Jayaraman, I. Yaqoob, and J. Arshad, “Trustworthy Blockchain Gateways for Resource-Constrained Clients and IoT Devices,” IEEE Access, vol. 9, pp. 132875-132887, 2021, doi: 10.1109/ACCESS.2021.3115150.
- [9] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, Zero Trust Architecture, NIST Special Publication 800-207, Aug. 2020, doi: 10.6028/NIST.SP.800-207.
- [10] M. A. Ferrag, L. Shu, X. Yang, A. Derhab, and M. Maglaras, “Security and Privacy for Green IoT-Based Agriculture: Review, Blockchain Solutions, and Challenges,” IEEE Access, vol. 8, pp. 32031-32053, 2020.
- [11] B. Shala, U. Trick, A. Lehmann, B. Ghita, and S. Shialeles, “Blockchain and Trust for Secure, End-User-Based and Decentralized IoT Service Provision,” IEEE Access, vol. 8, pp. 143900-143923, 2020.
- [12] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, “On Blockchain and Its Integration With IoT: Challenges and Opportunities,” Future Generation Computer Systems, vol. 88, pp. 173-190, 2018, doi: 10.1016/j.future.2018.05.046.
- [13] M. A. Ferrag, L. Maglaras, H. Janicke, J. Jiang, and T. Shu, “Authentication Protocols for Internet of Things: A Comprehensive Survey,” Security and Communication Networks, vol. 2017, Art. no. 6562953, 2017.
- [14] A. Dorri, S. S. Kanhere, and R. Jurdak, “Blockchain in Internet of Things: Challenges and Solutions,” arXiv:1608.05187, 2016, doi: 10.48550/arXiv.1608.05187.
- [15] K. Stouffer et al., Guide to Operational Technology (OT) Security, NIST Special Publication 800-82 Rev. 3, Sep. 2023, doi: 10.6028/NIST.SP.800-82r3.
- [16] M. Fagan et al., IoT Device Cybersecurity Capability Core Baseline, NISTIR 8259A, May 2020, doi: 10.6028/NIST.IR.8259A.
- [17] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, “Security and Privacy Challenges in Industrial Internet of Things,” in Proc. 52nd ACM/EDAC/IEEE Design Automation Conf. (DAC), 2015, pp. 1-6, doi: 10.1145/2744769.2747942.
- [18] A. Humayed, J. Lin, F. Li, and B. Luo, “Cyber-Physical Systems Security-A Survey,” IEEE Internet of Things Journal, vol. 4, no. 6, pp. 1802-1831, Dec. 2017, doi: 10.1109/JIOT.2017.2703172.
- [19] M. A. Khan and K. Salah, “IoT Security: Review, Blockchain Solutions, and Open Challenges,” Future Generation Computer Systems, vol. 82, pp. 395-411, 2018, doi: 10.1016/j.future.2017.11.022.
- [20] J. Sengupta, S. Ruj, and S. Das Bit, “A Comprehensive Survey on Attacks, Security Issues and Blockchain Solutions for IoT and IIoT,” Journal of Network and Computer Applications, vol. 149, Art. no. 102481, 2020.
- [21] H.-N. Dai, Z. Zheng, and Y. Zhang, “Blockchain for Internet of Things: A Survey,” IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8076-8094, Oct. 2019, doi: 10.1109/JIOT.2019.2920987.
- [22] I. Makhdoom, M. Abolhasan, H. Abbas, and W. Ni, “Blockchain’s Adoption in IoT: The Challenges, and a Way Forward,” Journal of Network and Computer Applications, vol. 125, pp. 251-279, 2019, doi: 10.1016/j.jnca.2018.10.019.
- [23] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, “Blockchain and IoT Integration: A Systematic Survey,” Sensors, vol. 18, no. 8, Art. no. 2575, 2018, doi: 10.3390/s18082575.
- [24] K. Christidis and M. Devetsikiotis, “Blockchains and Smart Contracts for the Internet of Things,” IEEE Access, vol. 4, pp. 2292-2303, 2016, doi: 10.1109/ACCESS.2016.2566339.
- [25] O. Novo, “Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT,” IEEE Internet of Things Journal, vol. 5, no. 2, pp. 1184-1195, Apr. 2018, doi: 10.1109/JIOT.2018.2812239.

- [26] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart Contract-Based Access Control for the Internet of Things," IEEE Internet of Things Journal, vol. 6, no. 2, pp. 1594-1605, Apr. 2019, doi: 10.1109/JIOT.2018.2847705.
- [27] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, May 2015, doi: 10.17487/RFC7519.
- [28] Y. Sheffer, D. Hardt, and M. Jones, "JSON Web Token Best Current Practices," BCP 225, RFC 8725, Feb. 2020, doi: 10.17487/RFC8725.
- [29] K. Wüst and A. Gervais, "Do You Need a Blockchain?" in Proc. 2018 Crypto Valley Conf. Blockchain Technology (CVCBT), 2018, pp. 45-54, doi: 10.1109/CVCBT.2018.00011.
- [30] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "LSB: A Lightweight Scalable Blockchain for IoT Security and Privacy," Journal of Parallel and Distributed Computing, vol. 134, pp. 180-197, Dec. 2019, doi: 10.1016/j.jpdc.2019.08.005.