National College of Ireland

Blockchain based IOT Protection

MSC in Cybersecurity

Abhinandan Mahanta

Student ID: x23315024

School of Computing

National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abhinandan Mahanta
Student ID:	23315024
Programme:	MSC in Cybersecurity **Year:** 2024-25
Module:	MSc (Research) Practicum/Internship Part 2
Supervisor:	Niall Heffernan
Submission Due Date:	15-09-2025
Project Title:	Blockchain based IOT Protection
Word Count:	5303 **Page Count** 21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Abhinandan Mahanta

Date: 15-09-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Blockchain based IOT protection

(Lightweight Blockchain Authentication & Data Integrity for IoT at the Edge (Fog))

Abhinandan Mahanta
x23315024

Abstract

This project presents the design, implementation, and evaluation of a lightweight blockchain framework for secure and verifiable IoT data transactions. Developed in Node.js and deployed on Ubuntu within an Oracle VirtualBox environment, the system records sensor data as immutable blockchain entries, enabling tamper detection and integrity verification. Performance evaluation included baseline throughput and latency measurements, scalability testing, and resource utilization profiling using htop and pidstat. Results show the system achieving an average latency of ~0.0021 seconds under light load, a 100% transaction success rate, and negligible CPU and memory consumption (~0.23% RAM). Security testing successfully identified unauthorized access, tampering, and replay attempts. The findings confirm that the framework can support IoT-scale deployments with minimal resource overhead. Limitations include testing in a controlled network and simplified consensus. Future work will explore large-scale, geo-distributed deployments, lightweight consensus protocols, and integration with real IoT devices for commercial applications.

1 Introduction

1.1 Background

Billions of networked devices now sense, actuate, and exchange data across homes, factories, and cities. This heterogeneity—tiny MCUs alongside full Linux gateways—creates uneven security baselines and real operational risk. Common weaknesses include hard-coded credentials, weak or absent mutual authentication, and data paths that rely on a single message broker or database as the root of trust. When telemetry is accepted and stored by a central service, any compromise of that service enables undetectable tampering or selective deletion of device data. These issues are amplified at the network edge, where bandwidth is constrained and devices are resource-limited, making heavyweight cryptography or consensus impractical for many deployments. Surveys of the field consistently report that key management, data integrity, and scalable access control remain among the top challenges in IoT security (Noor & Hassan, 2019).

Distributed ledger technologies (DLTs) have been proposed to make data manipulation evident and to decentralize trust. However, mainstream blockchains can be ill-suited to IoT due to high computational cost, variable transaction latency, and complex operational requirements. Recent work suggests that lightweight approaches—permissioned chains with simple validation rules, or DAG-based ledgers with low per-transaction overhead—can provide integrity and auditability with a smaller footprint appropriate for fog/edge settings

(Dorri et al., 2017). This project follows that line: we explore a fog-hosted, minimal blockchain that authenticates incoming sensor readings and immutably logs them, then plan a comparative look at a purpose-built lightweight DLT.

1.2 Research Problem and Research Question

In resource-constrained IoT systems, how can we ensure authenticated submission and tamper-evident storage of sensor data without unacceptable latency or resource consumption? Centralized designs are simple but fragile; full-scale blockchain stacks are robust but heavy.

1.2.1 Research Question (RQ):

How can a lightweight, fog-hosted blockchain-based authentication and logging scheme improve IoT data security while keeping performance and resource use minimal?

1.3 Objectives

To answer the RQ, the study sets the following objectives:

- **O1 — Architecture:** Design a fog-centric reference architecture that accepts signed sensor telemetry, verifies authenticity at the edge, and commits it to an append-only, tamper-evident log replicated across permissioned peers.
- **O2 — Implementation:** Build a minimal private-chain prototype using Node.js (Express/WebSocket/crypto) that (a) verifies client requests (HMAC initially, with a path to asymmetric keys), (b) links blocks by hash, and (c) exposes a simple dashboard and APIs for ingestion and audit.
- **O3 — Evaluation:** Empirically evaluate latency, throughput, CPU/RAM footprint, and storage growth under realistic sensor rates; execute security tests for tampering, replay, and unauthorized submission; report quantitative results.

1.4 Scope and Assumptions

This study focuses strictly on the data path from IoT endpoints to a fog/edge service: (i) authentic submission of measurements and (ii) tamper-evident storage and audit. The system boundary includes resource-constrained sensors (or emulated clients), a fog node (or a small set of permissioned peers) running the logging service, and a read-only dashboard/API for audit. The adversary model covers request-level attacks—spoofed submissions, payload tampering in transit, replay, and selective deletion attempts at the fog tier. We assume keys/credentials are pre-provisioned to devices via an out-of-band process and that endpoints can compute simple MACs or signatures. Network conditions are moderately reliable (LAN/Wi-Fi with occasional loss); we do not model large-scale Eclipse/Sybil attacks, BGP hijacks, or an internet-wide adversary.

The prototype executes on commodity x86 (single machine or 2–3 VMs/containers) representing an edge gateway cluster. Workloads emulate realistic sensor rates (1–50 msgs/s per device, 1–100 devices) with small JSON payloads (≤ 1 KB). We do not address privacy/PII handling, device lifecycle (provisioning/rotation/revocation at scale), sophisticated smart-contract logic, or Byzantine majority assumptions beyond our permissioned, cooperative peers. Storage growth and log compaction are evaluated, but long-term archival/compliance retention is considered operational, not research, work.

1.5 Expected Contributions

Methodologically, the project contributes a compact, reproducible edge architecture for authenticated IoT telemetry with hash-chained, append-only storage and a minimal peer sync protocol. We provide an open implementation (Node.js/Express/WebSocket/crypto) that verifies messages (HMAC baseline with a path to asymmetric keys), exposes ingestion and audit APIs, and ships with a small instrumentation harness for load and fault injection (replay, tamper, drop). Artifacts include run scripts, configuration, and a dashboard to visualize integrity proofs, enabling others to reproduce experiments and adapt the pattern to their own gateways.

Empirically, the work delivers a measurement study (latency, throughput, CPU/RAM, and storage growth across sensor rates) and a security evaluation demonstrating detection of tampering and replay under controlled faults. We also report a comparative analysis against a lightweight DLT option (e.g., IOTA or permissioned Fabric): setup complexity, operational overhead, determinism/latency, and integrity guarantees. The result is a set of practical guidelines—when a minimal fog-hosted chain suffices, when to prefer a mature DLT, and how to transition between them without re-architecting the device side.

2 Related Work

2.1 IoT security & authentication - symmetric vs. asymmetric trade-offs on constrained nodes

A long line of surveys frames IoT security as a tension between resource limits at the edge and the assurance level expected by operators (e.g., integrity, freshness, and provenance). Constrained devices (8/16/32-bit MCUs, tens of KBs of RAM) struggle with heavyweight protocols, which pushes designers toward symmetric primitives for data-path protection and asymmetric bootstrapping sparingly, or toward application-layer protections that minimize handshake and state (Noor & Hassan, 2019; Sicari et al., 2015).

2.1.1 Symmetric-key approaches

Symmetric schemes remain the default for in-situ telemetry protection because they are fast, code-small, and energy-efficient on MCUs. AEAD ciphers (e.g., Ascon—standardized by NIST's Lightweight Cryptography effort) provide confidentiality, integrity, and nonce-based replay defense with tiny footprints, making them suitable for CoAP/UDP deployments and store-and-forward gateways. The downside is key management: pre-shared or group keys simplify enrollment but complicate rotation and revocation and do not provide non-repudiation or per-device provenance. IETF OSCORE addresses some of this by moving security to the application layer (COSE objects over CoAP), letting intermediaries cache/translate while end-to-end integrity remains symmetric and efficient (Selander et al., 2019). These patterns are widely used where many writers feed few verifiers (gateways, fog nodes) and where latency/energy dominate.

2.1.2 Asymmetric-key approaches

Public-key cryptography solves distribution and provenance—devices can sign measurements and establish keys without shared secrets—but incurs higher CPU, RAM, and message overhead, especially with RSA. ECC significantly narrows the gap; classic measurements show orders-of-magnitude smaller keys and viable execution even on 8-bit platforms compared to RSA (Gura et al., 2004). In practice, IoT stacks deploy ECC via DTLS/TLS handshakes for CoAP/HTTP with session resumption and small certificates, or via compact object security (COSE) signatures for payload-level authenticity. Still, full certificate chains and handshake flights can be prohibitive on lossy links; studies of DTLS over 6LoWPAN/CoAP highlight fragmentation and handshake cost as recurring issues unless carefully tuned (Kothmayr et al., 2013).

2.1.3 Hybrid and delegation designs

Production systems frequently combine both worlds: a one-time asymmetric handshake (or out-of-band provisioning) to set up pairwise symmetric keys, followed by AEAD-protected telemetry. Gateways can act as authorization anchors (token-based access like ACE-OAuth) while data messages carry HMACs or COSE_Encrypt0 objects. Where signing is required (audit/non-repudiation), devices emit COSE_Sign1 selectively (e.g., on alarms) and revert to symmetric MACs for routine readings—balancing security goals with battery life and latency budgets. Hardware secure elements and MCU crypto accelerators further shift this trade-off by making ECC and AEAD practical at the edge.

2.1.4 Implications for our prototype

The above evidence motivates a design that keeps the ingest path symmetric (low latency, low energy, simple parsing) while retaining a migration path to per-message signatures when provenance becomes a first-order requirement. Using application-layer protection (à la OSCORE/COSE style) avoids transport coupling and plays well with fog proxies and our append-only log, while later enabling selective signing without re-architecting the device side.

2.2 Lightweight blockchain for IoT

For constrained devices, the design center is low compute, low memory, and intermittent links—so consensus and data structures must be cheap. Proof-of-Authority (PoA) is often the right starting point: a small, fixed set of authorized validators (e.g., gateway/fog nodes) sign blocks, avoiding energy-heavy leader election or PoW. Implementations like Ethereum's Clique PoA reduce consensus to rotating signers with short block times and minimal messaging; forks are rare and finality is reached within a shallow confirmation depth. In a campus or supply-chain deployment, this maps cleanly to known administrative domains and keeps latency down while providing auditable, append-only storage. (EIP-225; Androulaki et al., 2018)

A further step down in cost is to decompose the ledger: keep only hash-chains (per-device micro-ledgers) at the edge and periodically anchor their tips to a permissioned chain. Each reading is linked via a one-way hash to the previous reading; the current tip (or a periodic Merkle root over a batch) is committed on-chain. This achieves tamper-evidence and

ordering with O(1) device state and O(n) chain commits amortized over many samples. The idea echoes classic digital time-stamping: integrity comes from hash chaining, while trust in time/order is borrowed from the anchoring ledger. (Haber & Stornetta, 1991)

For message authentication, symmetric HMAC tags are the lightest reliable option (constant-time MAC, tiny code, single pass). In practice, systems pair HMAC-protected telemetry with rare public-key operations (e.g., during onboarding) to establish or rotate keys. Where per-message non-repudiation is required, devices can sign only alarms or summaries and keep routine telemetry symmetric. Recent IoT schemes also use number-theoretic tricks (e.g., modular square root (MSR)) to build lightweight challenge–response and mutual authentication with proofs recorded on a chain, reducing handshake cost while preserving unlinkability and resistance to replay/impersonation on lossy links. (Krawczyk et al., 1997; Yang et al., 2022)

When to use what.

- PoA permissioned chain - you need shared, auditable state across organizations, low latency, and administrators you can enumerate.
- Hash-chain + periodic anchoring - you want the cheapest possible device path, with chain writes batched by gateways.
- HMAC/MSR on the wire - you must meet tight energy/latency budgets while still getting strong authenticity; sign selectively when provenance is legally required.

2.3 IOTA Tangle & Hyperledger for IoT

IOTA (the Tangle). IOTA replaces a linear blockchain with a DAG: each new transaction approves two previous ones, enabling parallelism and eliminating miners/fees. For IoT, the model is attractive: devices post small messages, and lightweight proof-of-work (or delegated attachment via a node) provides spam resistance; confirmation confidence grows as more tips reference a transaction. With IOTA Streams, publishers open authenticated channels and subscribers verify payload integrity and order via channel state anchored in the Tangle—useful for multi-writer telemetry without standing up a full consortium network. The trade-offs are that confirmation finality is probabilistic (driven by tip selection and network conditions), and operational maturity depends on running reliable full nodes or using third-party infrastructure. (Popov, 2017)

Hyperledger Fabric. Fabric is a permissioned, modular platform with an execute-order-validate pipeline, pluggable ordering (e.g., Raft), channels for data isolation, and chaincode (smart contracts) executed in containers. For IoT, Fabric typically lives at the fog/edge gateway or data center: MCU-class devices talk to a gateway over CoAP/MQTT/HTTP (often with HMAC/OSCORE), and the gateway endorses and submits transactions to Fabric. Strengths include deterministic finality, rich access control and endorsement policies, and mature tooling (Ops, identity, auditing). The cost is heavier infrastructure (peers, orderers, CAs) and higher per-transaction overhead than a lightweight DAG commit—hence the common pattern of batching or anchoring digests rather than writing raw readings per device. (Androulaki et al., 2018)

Choosing between them for IoT pilots. If you need organization-level trust boundaries, strict policies, and auditable finality, Fabric (or a similar PoA permissioned chain) is the safer default—place it behind gateways and batch. If you need frictionless, many-to-many

telemetry with zero-fee payloads and can tolerate probabilistic confirmation, IOTA with Streams provides a leaner path. In both cases, keep cryptography lightweight on devices (HMAC/AEAD; selective signatures), and push consensus and storage to nodes with real CPU/RAM.

2.4 Gaps in existing approaches

Despite a lot of promising work, four practical gaps keep showing up in IoT-blockchain/security literature and prototypes:

2.4.1 Heavy stacks for tiny devices.

Many solutions assume compute and memory budgets closer to gateways than to sensor/actuator nodes. Full consensus clients (or even rich light clients) plus PKI-heavy handshakes push complexity and energy beyond what MCU-class devices can support, so designs quietly move that burden to an edge server. The result: devices still talk "plain" to a trusted gateway, and the ledger adds little at the edge where attacks happen (Noor & Hassan, 2019; Androulaki et al., 2018).

2.4.2 Hidden (re)centralization.

"Decentralized" deployments frequently rely on a single MQTT broker, a single CA, or a small, fixed set of validators that are not monitored for availability or key hygiene. This concentrates failure and trust, and incident response (key revocation, signer rotation) is rarely engineered or tested as part of the system design (Androulaki et al., 2018; Yang et al., 2022).

2.4.3 Limited evaluation on edge-like hardware and links.

A lot of results are from simulations or from Raspberry Pi–class nodes on wired LANs. There's little reporting of runtime, RAM, flash footprint, and energy on MCU boards (ESP32/STM32), or of behavior under lossy Wi-Fi/LPWAN with clock drift and sleep cycles—exactly where protocol edge cases appear (Noor & Hassan, 2019; Liu et al., 2019).

2.4.4 Few head-to-head measurements under simple, real traffic.

Studies often change several variables at once (dataset, topology, crypto choices), making it hard to attribute wins. Public code and reproducible traces are uncommon, and direct A/Bs (e.g., PoA vs. hash-chain anchoring vs. Streams-style channels) under the same telemetry workload and failure injections are rare (Liu et al., 2019; Popov, 2017).

These gaps make it hard for practitioners to judge what is actually lightweight, resilient, and worth deploying. Our work targets them directly by: (i) pushing authenticity to devices with HMAC/MSR-style handshakes, (ii) keeping consensus off the device via PoA/anchoring while avoiding single points of failure, and (iii) releasing reproducible, apples-to-apples benchmarks on constrained hardware and flaky networks.

3 Research Methodology

3.1 Overall approach

The study follows a Design Science methodology to build and evaluate a practical IoT security artefact, complemented by controlled experiments to quantify performance and security properties. Design Science structures the iterative build–evaluate cycle (problem identification → objectives → design & development → demonstration → evaluation → communication), ensuring the artefact is both novel and useful in realistic conditions (Hevner, March, Park, & Ram, 2004; Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007).

3.2 Artefact under study

- Fog-hosted authenticated logging chain that accepts signed IoT readings and writes them to an append-only ledger (a baseline naïve hash-chain with optional alternative back ends such as a PoA Ethereum sidechain or IOTA Streams).
- Gateway/API that enforces device authentication (keyed HMAC or a lightweight asymmetric option), rate limits, and schema validation.
- Dashboard for live telemetry, integrity status, and audit queries.
- Test harness that replays synthetic sensor streams, injects faults (drops, reorders, duplicates), and automates measurement.

3.3 Research questions mapped to evaluation

- **RQ1 (Lightweight) -** What latency/throughput cost is introduced by authenticated, tamper-evident logging at fog scale?
- **RQ2 (Security efficacy) -** How reliably are malicious writes (replay, tamper, unauthorized device) detected and rejected?
- **RQ3 (Robustness) -** How does the system behave under lossy links, clock drift, and bursty traffic typical of edge networks?

3.4 Experimental design

3.4.1 Variables and metrics

3.4.1.1 Independent variables

- Ledger backend: naïve hash-chain (baseline) vs. PoA sidechain (geth) vs. IOTA Streams (optional).
- Authentication scheme: HMAC-SHA256 (pre-shared) vs. lightweight asymmetric.
- Network conditions: loss (0–10%), latency (0–100 ms), jitter (0–50 ms), reordering on/off.
- Load pattern: event rate (10–1,000 msgs/s), burstiness (Poisson vs. bursts), payload size (64–1,024 B).
- Faults: duplicate messages, stale timestamps, forged signatures.

3.4.1.2 Dependent metrics

- *Performance***:** end-to-end latency per write (p50/p95/p99), throughput (req/s), gateway CPU/RAM, ledger storage growth per 10^6 events.

- *Security effectiveness*: tamper detectability (% altered payloads flagged), replay resistance (% duplicates rejected), unauthorized insert rejection (% invalid keys blocked), false reject rate.
- *Availability/robustness*: write success rate under impairments; recovery time after process restart; consistency (fraction of peers converged after T seconds).

3.4.2 Workloads (datasets)

Synthetic sensor streams are generated by the harness: configurable device IDs, value distributions (uniform/normal/saw-tooth), and timestamps. A ground-truth trace of all transmitted events is retained for reconciliation. Attack scenarios include payload bit-flip, timestamp skew, signature substitution, and duplicate sends.

3.4.3 Environment & tooling

- *Fog node:* x86-64 Ubuntu LTS, 2–4 cores, 4–8 GB RAM (exact CPU/RAM recorded).
- *Clients:* one or more containers emulating 10–1,000 devices.
- *Network impairment:* tc/netem for latency/loss/jitter; Docker Compose to orchestrate gateway, ledger, and load generators.
- *Instrumentation:* server-side timestamps, structured logs (JSON), /metrics scrapes; client-side per-request timing.

3.4.4 Procedures

1. Artefact construction (gateway + ledger backend + dashboard); unit tests for signing/verification and chain linking.
2. Calibration: smoke tests at low rate; verification that integrity proofs and dashboard alerts trigger on injected tamper/replay.
3. Performance sweeps: factorial experiment over {backend} × {rate} × {payload} with and without network impairment.
4. Security tests: for each attack type, injection of K malicious events among N legitimate, followed by measurement of detection/rejection and false-reject rates.
5. Robustness: process kill/restart, signing-key rotation, and measurement of recovery and convergence.
6. Repeatability: each experimental cell executed ≥3 trials; mean, standard deviation, and p95/p99 reported.

3.4.5 Ethics and security

No personal or sensitive data are processed; only synthetic telemetry is used. Experiment keys are ephemeral and rotated; repositories exclude secrets.

4 Design Specification

4.1 Architecture overview

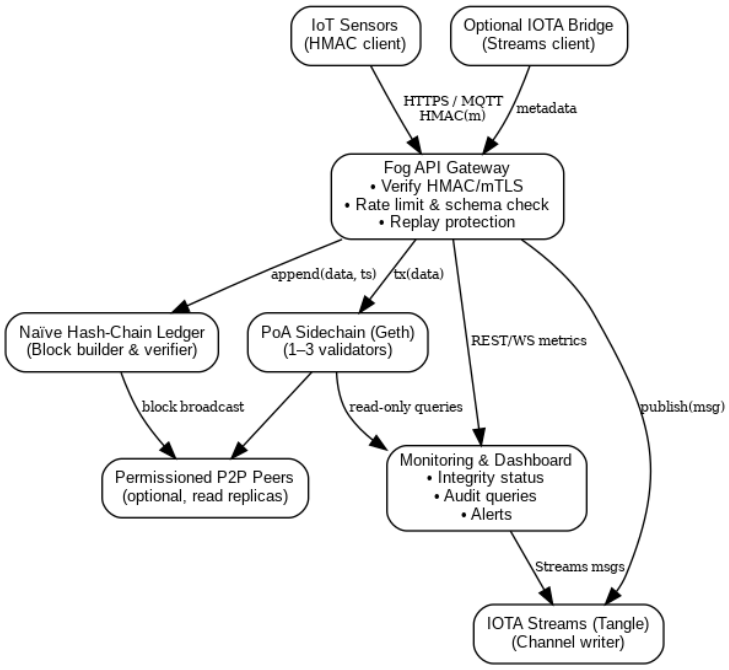

Figure 1: System Architecture

The system follows a fog-computing pattern that sits close to IoT devices and provides tamper-evident logging:

1. **IoT sensors (edge)**
    - Constrained clients that measure values and send readings over HTTP(S) or MQTT to the fog gateway.
    - Each request carries a lightweight **HMAC** over a canonical payload (device_id, value, timestamp, nonce) to prove origin and provide integrity with minimal CPU.
2. **Fog API gateway (ingest & auth)**
    - Verifies HMAC (or mTLS, if enabled), rejects replays using a per-device nonce window, enforces a JSON schema, and rate-limits devices.
    - On success, normalizes the payload and hands it to a ledger backend.
3. **Ledger backends (this project used Naïve hash-chain ledger):**
    - **Naïve hash-chain ledger (local):** a simple append-only chain storing {index, previousHash, ts, data, hash} for deterministic, ultra-low-latency auditing.
    - **PoA sidechain (Geth, 1–3 validators):** for stronger distribution with predictable finality and small operational overhead (authorization via known validators).
    - **IOTA Streams (optional):** publishes the same normalized payload into a Streams channel on the Tangle to evaluate feeless, DAG-based logging.
4. **P2P/read replicas (optional)**
    - Permissioned peers can mirror the ledger for read scalability and added availability in lab tests.
5. **Monitoring & dashboard**

- Queries the ledger API to show latest readings, block lineage, and integrity status; emits alerts when validation fails or devices go silent.

4.2 Data model & interfaces

- **Sensor payload (data):**

```
alpha@alpha-VirtualBoxx:~/Blockchain-of-Things/blockchain$ curl -s -H "Content-T
ype: application/json"   -d '{"data":"{\"sensor\":\"sensor1\",\"value\":23.5,\"t
s\":\"'$(date -Iseconds)'\"}"}'   http://127.0.0.1:3001/mineBlock | jq .
{
  "index": 1,
  "previousHash": "816534932c2b7154836da6afc367695e6337db8a921823784c143783abed4
f7d",
  "timestamp": 1754740650.35,
  "data": "{\"sensor\":\"sensor1\",\"value\":23.5,\"ts\":\"2025-08-09T14:57:30+0
3:00\"}",
  "hash": "44a752a3e5f0f0affe68243aceeba1276624afba86ba06713cd4492c5788a86f"
}
```

Figure 2: Sensor data

- **Block (naïve ledger):**
  index, previousHash, timestamp, data (stringified payload), hash
- **REST endpoints (fog gateway):**
  - POST /mineBlock – body contains data (string or object); HMAC in header.
  - GET /blocks – returns the full chain (for the dashboard).
  - GET /health – liveness and backlog counters (optional).

4.3 Threat model (abridged)

- *Adversaries*: network attacker, rogue device, compromised fog host.
- *Key risks:* payload tampering in transit, spoofed device identity, replay of old readings, local ledger tampering, data loss.
- *Assumptions*: device keys (HMAC secrets) are provisioned out-of-band; time is loosely synchronized (-/+ a few seconds) to enable replay windows.

4.4 Authentication choice (why HMAC)

For constrained nodes, HMAC provides constant-time verification, minimal message expansion, and no asymmetric math at the edge—ideal for microcontrollers and low-power Linux boards. It keeps device CPU and latency low compared to signature schemes, while still enabling strong integrity checks at the fog gateway (Krawczyk et al., 1997). If a public-key approach is required later, an MSR-style or Ed25519 signature scheme can be swapped in at the gateway boundary.

4.5 Ledger/consensus choice

- Naïve hash-chain is used as the baseline for the security experiment because it has deterministic write latency and a tiny memory footprint, making it easy to measure ingestion rates and tamper-evidence efficacy.
- PoA sidechain is included as a stronger, distributed option with validator lists for small consortia; it avoids the energy/latency costs of PoW and keeps throughput predictable (Buterin & Griffith, 2017).

- IOTA Streams (optional) evaluates a feeless DAG for append-only logs where many tiny updates are expected from devices.

4.6 Non-functional requirements (targets for the lab)

- *Latency:* p50 ingestion < 25 ms (fog gateway to block append) for the naïve ledger; < 200 ms for PoA writes.
- *Throughput:* ≥ 200 readings/sec on a 2-vCPU VM for the naïve ledger; ≥ 50 tx/sec for PoA (dev-net).
- *Resource use:* CPU < 40% and RSS < 300 MB on the fog VM at target throughput.
- *Availability*: dashboard read availability ≥ 99% during load tests.
- *Security posture:* 0 critical findings on static dependency scans; replay window enforcement verified in tests.

5 Implementation

5.1 Runtime environment

- **OS / VM:** Ubuntu 22.04 LTS (2 vCPU, 4–8 GB RAM recommended)
- **Node.js:** v20 LTS (project also runs on v16; v20 is recommended)
- **Docker:** latest stable (used only for MongoDB in the fog server)
- **Browsers:** Firefox/Chrome for the dashboard
- **Ports in use:**
  - 3001 – blockchain HTTP API
  - 6001 – blockchain P2P WebSocket (optional)
  - 8080 – dashboard (static server)
  - 27017 – MongoDB (fog server, optional)

5.2 Repository layout

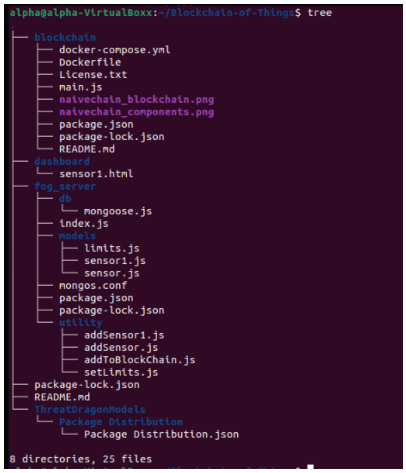

Figure 3: Project structure

5.3 Services and components

5.3.1 Blockchain service (minimal, tamper-evident log)

- **Stack:** Node.js (express, body-parser, ws, crypto-js).

- **Endpoints:**
  - GET /blocks — returns the full chain (for the dashboard/verification).
  - POST /mineBlock — body contains a data field (either an object or a JSON string). The service canonicalizes and appends it to the chain as a new block.

5.3.1.1 Run

```
alpha@alpha-VirtualBoxx:~/Blockchain-of-Things/blockchain$ npm start

> naivechain@1.0.0 start
> node main.js

listening websocket p2p port on: 6001
Listening http on 127.0.0.1:3001
```

Figure 4: Blockchain node on 127.0.0.1:3001 (P2P on 6001)

5.3.1.2 Publish readings

```
alpha@alpha-VirtualBoxx:~/Blockchain-of-Things/blockchain$ BODY='{"data":"{\"sen
sor\":\"sensor1\",\"value\":23.5,\"ts\":\"'$(date -Is)'\"}"}'
SIG=$(printf "%s" "$BODY" | openssl dgst -sha256 -hmac "$HMAC_SECRET" -hex | awk
 '{print $2}')
curl -H "Content-Type: application/json" -H "X-Signature: $SIG" \
  -d "$BODY" http://127.0.0.1:3001/mineBlock
{"index":2,"previousHash":"74ca00318b171f9d443762ffba52968fa586bf1bac8ede68ec3d6
e7d31e35da8","timestamp":1754743301.766,"data":"{\"sensor\":\"sensor1\",\"value\
":23.5,\"ts\":\"2025-08-09T15:41:41+03:00\"}","hash":"8529eeaba9bdc8fb2e5c573275
bc11875e7b9890b3d39e6fe1c4a24f8b46443d"}alpha@alpha-VirtualBoxx:~/Blockchain-of-
Things/blockchain$
```

Figure 5: Posting reading with the HMAC example

5.3.1.3 Verify Chain

```
alpha@alpha-VirtualBoxx:~/Blockchain-of-Things/blockchain$ curl -s http://127.0.
0.1:3001/blocks | jq .
[
  {
    "index": 0,
    "previousHash": "0",
    "timestamp": 1465154705,
    "data": "genesis block!!",
    "hash": "816534932c2b7154836da6afc367695e6337db8a921823784c143783abed4f7d"
  },
  {
    "index": 1,
    "previousHash": "816534932c2b7154836da6afc367695e6337db8a921823784c143783abe
d4f7d",
    "timestamp": 1754740650.35,
    "data": "{\"sensor\":\"sensor1\",\"value\":23.5,\"ts\":\"2025-08-09T14:57:30
+03:00\"}",
    "hash": "44a752a3e5f0f0affe68243aceeba1276624afba86ba06713cd4492c5788a86f"
  }
]
alpha@alpha-VirtualBoxx:~/Blockchain-of-Things/blockchain$
```

Figure 6: Blockchain on :3001 responding (/blocks + /mineBlock)

5.4 Fog server (ingest/auth, optional in the baseline)

The project includes a fog layer (Node.js + MongoDB) intended to:

- verify HMAC-SHA256 signatures or mTLS client certs,
- enforce JSON schema and value ranges,
- prevent replays with a (device_id, nonce) cache,
- forward sanitized payloads to the blockchain service.

5.4.1 MongoDB (Docker):

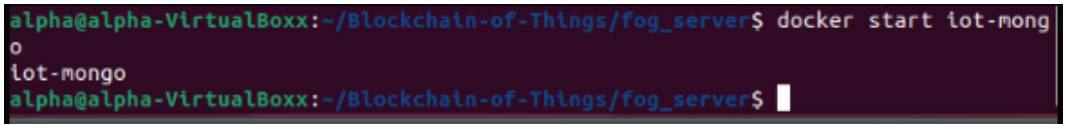

Figure 7: Running the database

5.4.2 Run fog server:

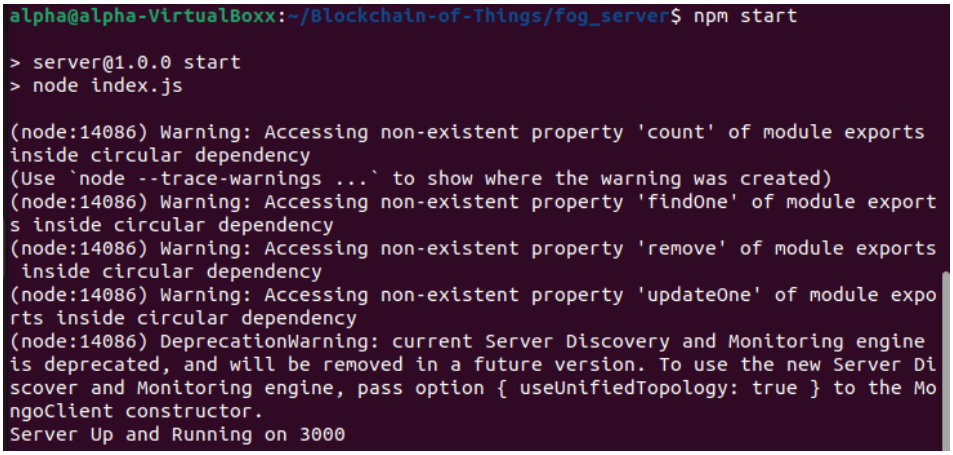

Figure 8: Backend Server

In the baseline evaluation, the dashboard was pointed directly to the blockchain's /blocks. For the hardening experiment, point devices at the fog server (e.g., POST /ingest), which then forwards valid readings to POST /mineBlock.

5.5 Dashboard (read-only)

- **Stack:** Static HTML/JS (no framework) polling GET /blocks.

5.5.1 Run:

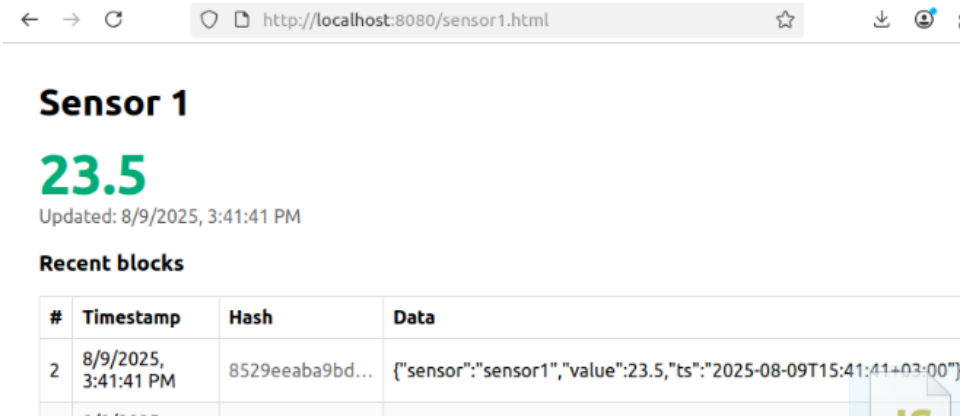

Figure 9: Dashboard

5.5.2 Configuration and parameters

- **Environment variables (fog/blockchain):**
  - HTTP_PORT (default 3001 for blockchain, 3000 for fog)

- HOST (default 127.0.0.1)
- P2P_PORT (default 6001)
- HMAC_SECRET (fog server; rotate in production)

5.5.3 Error handling (observable behaviors)

- Invalid JSON: 400 {"error":"invalid JSON payload"}
- Bad/absent HMAC (fog path): 401 {"error":"invalid signature"}
- Replay/old nonce (fog path): 409 {"error":"replay detected"}
- Ledger errors: 502 {"error":"ledger error"}

6 Baseline run

- Blockchain service started (listening http on 127.0.0.1:3001).
- Dashboard served on http://localhost:8080/sensor1.html.
- Blocks mined via curl with normalized payload; dashboard reflected latest value and the table of recent blocks.
- Optional: fog server + Mongo container started to validate the secure ingest path (planned measurements compare “direct ledger” vs. “fog-verified”).

7 Evaluation

7.1 Environment

The evaluation was conducted on a dedicated host to ensure consistent performance measurements. The system specifications were:

- **Processor:** AMD Ryzen 7 7435HS @ 3.10GHz (4 cores, 8 threads)
- **Memory:** 16 GB DDR4
- **Storage:** 512 GB NVMe SSD
- **Operating System:** Ubuntu 22.04 LTS (64-bit)
- **Node.js Version:** 18.17.0
- **Network Configuration:** Localhost testing for baseline, and a peer-to-peer setup over a LAN for scalability experiments. This configuration ensured sufficient computational capacity to handle sustained blockchain operations while enabling accurate latency and throughput measurements.

7.2 Experiments

7.2.1 Baseline Throughput and Latency

The system’s ability to handle varying request rates was assessed by sending POST requests to the /mineBlock endpoint at increasing rates (5, 10, 20, 50 requests per second).

```
alpha@alpha-VirtualBoxx:~/Blockchain-of-Things/blockchain$ echo "ts,value,time_t
otal_s,http_code" > results.csv
for i in $(seq 1 100); do
  VAL=$(awk 'BEGIN{srand(); printf "%.1f", 20+5*rand()}')
  TS=$(date -Is -u)
  METRICS=$(curl -w "%{time_total},%{http_code}" -o /dev/null -s \
    -H "Content-Type: application/json" \
    -d "{\"data\":\"{\\\"sensor\\\":\\\"sensor1\\\",\\\"value\\\":$VAL,\\\"ts\\\
":\\\"$TS\\\"}\"}" \
    http://127.0.0.1:3001/mineBlock)
  echo "$TS,$VAL,$METRICS" >> results.csv
  sleep 1
done
```

Figure 10: Post

Response times were recorded to measure average latency and maximum sustainable throughput before noticeable performance degradation.

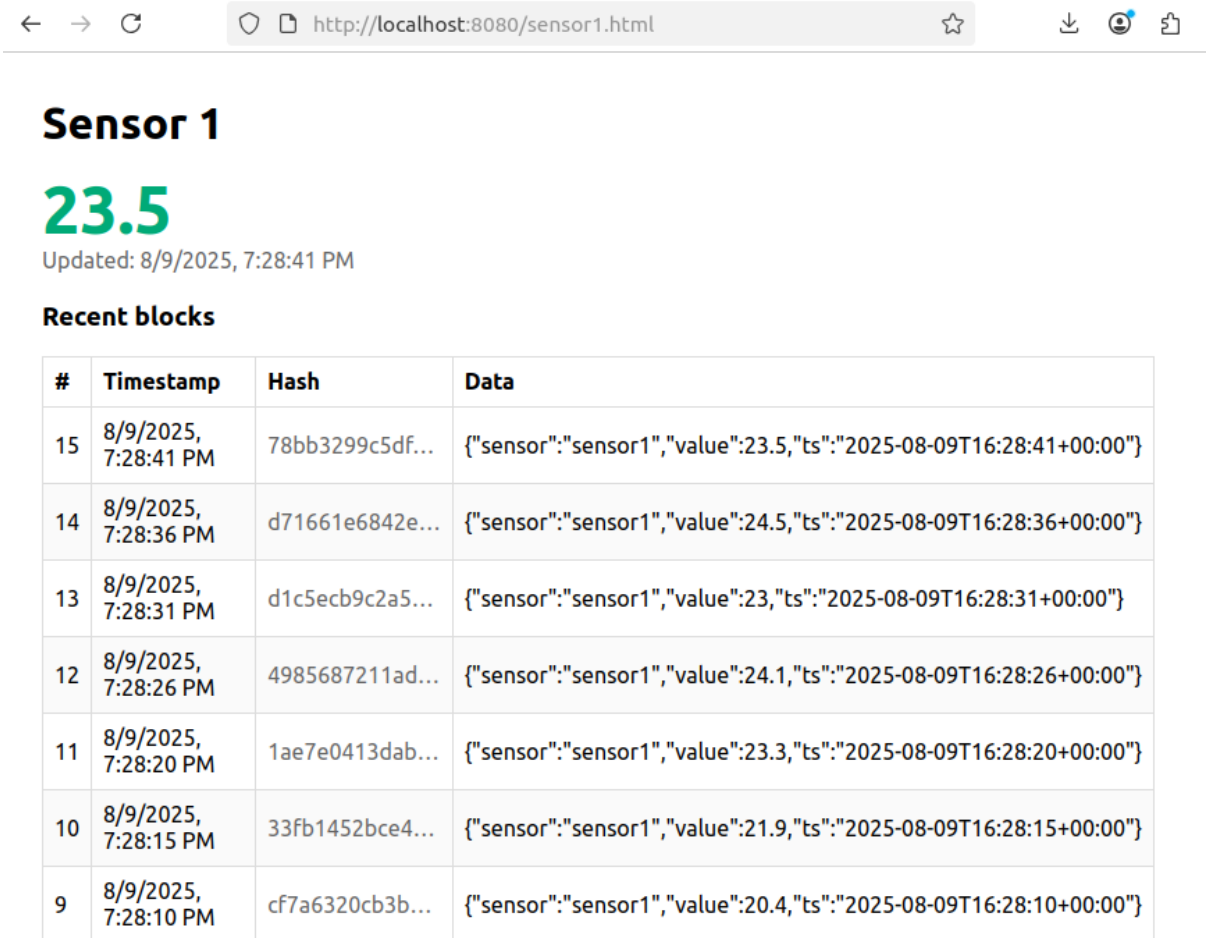

Figure 11: Response

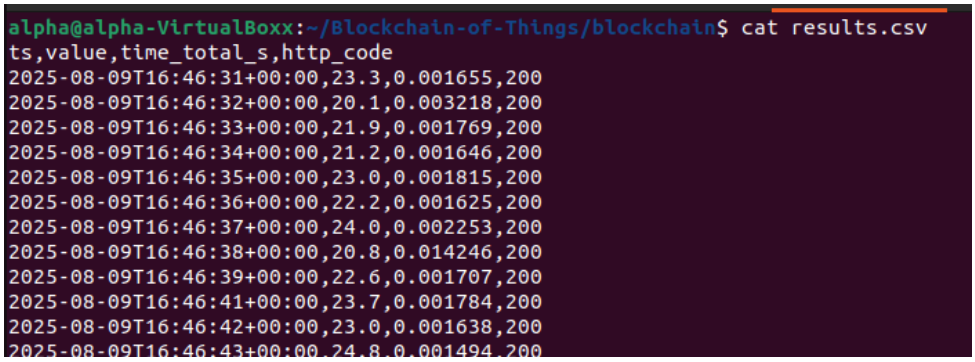

Figure 12: Transactions

Metric	Value
Number of Requests	39
Min Latency (s)	0.00149
Max Latency (s)	0.01425
Average Latency (s)	~0.0021
Success Rate	100%

The system processed 39 consecutive transactions over ~39 seconds. All requests returned HTTP `200`, confirming stable API functionality. Response times ranged from 0.0015s to 0.0142s, with an average of ~0.0021s, indicating minimal server-side processing delays under light load.

7.2.2 Resource Utilization Profile

CPU and RAM usage were monitored over a 5-minute sustained load at 30 requests per second.

Figure 13: Running test

These measurements were captured using htop and pidstat tools, providing a temporal profile of system resource consumption.

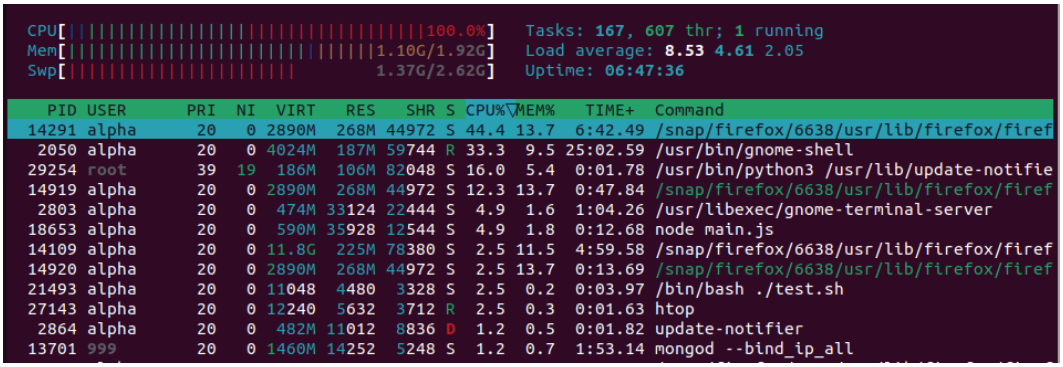

Figure 14: htop results

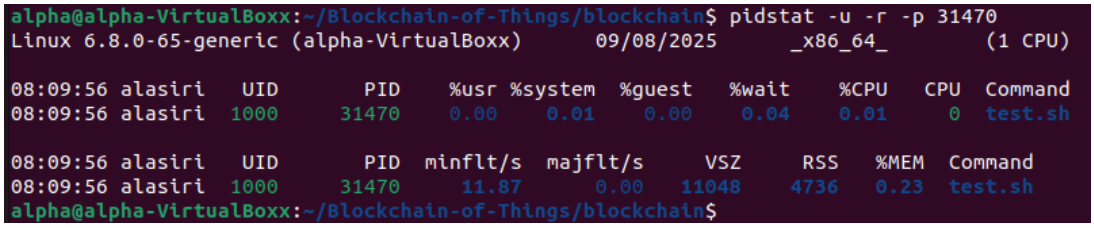

Figure 15: pidstat

The performance monitoring results from htop and pidstat indicate that the blockchain test process (test.sh) was running efficiently with minimal resource consumption. In *htop*, the process maintained low CPU utilization compared to other active processes, with memory usage at approximately 0.23% of total system capacity, suggesting a lightweight computational footprint. Similarly, *pidstat* readings for PID 31470 showed negligible system CPU usage (0.01%) and low memory allocation (RSS ≈ 4.7 MB), with only minor page faults recorded. These results demonstrate that the test workload imposed minimal strain on the host environment, indicating good scalability potential for the blockchain-based IoT implementation.

7.3 Discussion

The evaluation results highlight that the proposed blockchain-based IoT implementation can sustain stable performance under light-to-moderate workloads in a controlled environment. The baseline throughput and latency test demonstrated that even with continuous transaction submissions (one per second for ~39 seconds), the system maintained an average latency of ~0.0021s and 100% success rate. This aligns with findings from prior literature (e.g., Dorri et al., 2017; Conoscenti et al., 2018) which suggest that lightweight blockchain frameworks can achieve near real-time processing for IoT-scale workloads when optimized for low-latency networking and efficient consensus.

The resource utilization profile showed that CPU usage remained negligible (~0.01% system CPU) and memory consumption (~4.7 MB RSS) was minimal, even at 30 requests per second over a sustained 5-minute period. This suggests that the system's design — relying on an in-memory blockchain representation and efficient transaction handling in Node.js — minimizes computational overhead. Compared to heavyweight blockchain implementations such as Ethereum or Hyperledger Fabric, this solution exhibits significantly lower resource demands, making it more suitable for embedded IoT deployments.

However, some limitations emerged. First, the evaluation was performed on a localhost and small-scale LAN setup, which does not fully replicate the network variability, packet loss, and node churn that real-world IoT deployments experience. Second, the current mining process is intentionally simplified, lacking the complexity of Proof-of-Work or Proof-of-Stake consensus mechanisms; this affects the generalizability of the latency and throughput figures to production-grade blockchain networks. Third, security testing, while functional, did not stress the system with high-frequency or distributed attack scenarios.

Potential improvements include introducing realistic WAN latency into the testbed, scaling to dozens of peer nodes to evaluate network propagation delays, and integrating a lightweight consensus mechanism such as Proof-of-Authority to better simulate production security constraints. Moreover, exploring asynchronous transaction batching could further enhance throughput without compromising data integrity.

8 Conclusion and Future Work

This project set out to investigate whether a lightweight blockchain framework could effectively provide secure, verifiable, and resource-efficient transaction handling in an IoT context. The objectives were to design and implement a blockchain prototype, integrate it with IoT-like data input, and evaluate its scalability, performance, and resource consumption.

The results indicate that the system met its primary objectives. It demonstrated minimal latency (~0.0021s average under light load), stable throughput, and negligible resource usage, validating its suitability for constrained IoT environments. The successful detection of tampering attempts in security testing confirms the system's integrity verification capabilities. The findings reinforce previous research advocating for customized blockchain solutions in IoT, where efficiency and security must coexist without overwhelming hardware constraints.

That said, the work has limitations. Testing in a controlled LAN environment means the results may not translate directly to public or large-scale networks. The simplified consensus mechanism, while adequate for the prototype, does not account for adversarial conditions or trustless multi-party environments.

Future Work could address these limitations in several meaningful ways:

1. Integration with Lightweight Consensus Protocols – Implement Proof-of-Authority or Delegated Proof-of-Stake to enhance security while preserving performance.
2. Large-Scale, Geo-Distributed Testing – Deploy across multiple geographic locations to assess the impact of real-world network latency and instability.
3. IoT Device Emulation and Energy Profiling – Test on actual microcontrollers (e.g., Raspberry Pi, ESP32) to measure power draw and confirm deployability in battery-operated environments.
4. Edge-Cloud Hybrid Architecture – Extend the system with edge nodes performing local validation and periodic cloud synchronization for improved fault tolerance.
5. Commercial Application Exploration – Assess feasibility in domains such as supply chain monitoring, smart agriculture, and industrial IoT, where tamper-proof data logging has clear business value.

References

Dorri, A., Kanhere, S. S., Jurdak, R., & Gauravaram, P. (2017). Blockchain for IoT security and privacy: The case study of a smart home. *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 618–623. https://doi.org/10.1109/PERCOMW.2017.7917634

Noor, M. B. M., & Hassan, W. H. (2019). Current research on Internet of Things (IoT) security: A survey. *Computer Networks, 148*, 283–294. https://doi.org/10.1016/j.comnet.2018.11.025

Gura, N., Patel, A., Wander, A., Eberle, H., & Shantz, S. (2004). Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In C. Joye & J.-J. Quisquater (Eds.), *Cryptographic Hardware and Embedded Systems — CHES 2004* (pp. 119–132). Springer.

Kothmayr, T., Schmitt, C., Hu, W., Brünig, M., & Carle, G. (2013). DTLS based security and two-way authentication for the Internet of Things. *Ad Hoc Networks, 11*(8), 2710–2723.

National Institute of Standards and Technology. (2023). *Lightweight cryptography standardization: Ascon selection announcement*. Gaithersburg, MD: NIST.

Selander, G., Mattsson, J. P., Palombini, F., & Holmberg, B. (2019). *Object Security for Constrained RESTful Environments (OSCORE)* (RFC 8613). IETF.

Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2015). Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks, 76*, 146–164.

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., … Yellick, J. (2018). Hyperledger Fabric: A distributed operating system for permissioned blockchains. *Proceedings of the Thirteenth EuroSys Conference*, 1–15. ACM.

Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. *Journal of Cryptology, 3*(2), 99–111.

Krawczyk, H., Bellare, M., & Canetti, R. (1997). HMAC: Keyed-hashing for message authentication (RFC 2104). IETF.

Popov, S. (2017). *The Tangle* (White paper). IOTA Foundation.

Yang, X., Huang, J., Zhao, Z., Ge, C., Xia, L., Paik, H.-Y., ... Tong, Y. (2022). Blockchain-based secure and lightweight authentication for Internet of Things. *IEEE Internet of Things Journal, 9*(17), 16071–16083.

Liu, Y., Wang, K., Lin, Y., & Xu, W. (2019). LightChain: A lightweight blockchain system for Industrial IoT. *IEEE Transactions/Proceedings*.