UNIVERSITÀ DI PISA

Department of Computer Science

Master's Degree in Computer Science and Networking

Privacy-Preserving Digital Product Passports for Container Logistics: Design, Implementation and Experimental Evaluation of a oneM2M, EPCIS 2.0 and IOTA Platform with Zero-Knowledge Proofs

Supervisor:
Dr. Paolo Pagano

Candidate:
Samiullah Khairy

Academic year 2025/2026

Abstract

Maritime container shipping carries over 80% of global trade by volume, yet verifying cold-chain compliance in transit creates a fundamental conflict: carriers treat operational telemetry as commercially sensitive, while regulators, insurers, and port authorities require independently verifiable proof that goods remained within specification. Existing approaches force a binary choice: disclose raw sensor data (sacrificing confidentiality) or issue self-declared compliance flags (sacrificing verifiability). Neither is adequate under the EU Ecodesign for Sustainable Products Regulation (ESPR), which mandates machine-readable Digital Product Passports (DPPs) with verifiable provenance data, nor do current maritime IT systems provide standardized event models for cross-organizational DPP interoperability.

This thesis presents Ocean DPP, a blockchain-anchored DPP platform that resolves this deadlock by combining GS1 EPCIS 2.0 for interoperable event semantics, oneM2M as a standardized IoT service layer, the IOTA distributed ledger for immutable anchoring, and Groth16 zero-knowledge proofs (ZKPs) that let external stakeholders verify compliance predicates — such as "temperature remained below threshold" — without learning the underlying sensor values. Merkle-tree batching amortises anchoring cost by grouping events into configurable batches (evaluated at 10, 50, and 100 events per IOTA transaction) while preserving per-event verifiability through stored sibling proofs.

A campaign of 16 experiments quantifies the platform in five dimensions. *Latency:* 48 ms P95 without ZKP (within the 200 ms design target), rising to 520 ms P95 with proof generation. *Privacy cost:* proof generation averages 304 ms, verification 9.8 ms (31:1 ratio), making privacy cost-effective for multi-stakeholder verification. *Throughput:* 7–10 events/s; horizontal scaling (two replicas) reduces median latency by 37%. *Standards compliance:* 100% of events pass three independent EPCIS 2.0 conformance checks. *Resilience:* four failure-injection scenarios produce zero permanent message loss.

In general, this work delivers a quantitatively evaluated maritime DPP prototype integrating EPCIS 2.0, oneM2M, IOTA, and Groth16, a formal threat model defining the privacy boundary between external ZKP-protected verification and role-gated internal access, and a public verification portal performing Groth16 proof verification in-browser with a build-time pinned key that eliminates server trust.

Keywords: Digital Product Passport, Blockchain, IOTA, EPCIS, oneM2M, Maritime Logistics, Zero-Knowledge Proof, Privacy-Preserving Verification, Groth16, Supply Chain Transparency

Alhamdulillah, I am profoundly grateful to Allah (SWT), the Most Beneficent, for bestowing upon me the strength, wisdom, and perseverance to complete this academic journey. His infinite mercy and guidance have been my constant companions throughout this challenging yet rewarding process.

I would like to express my sincere gratitude to my supervisor, Dr. Paolo Pagano, for his guidance, patience, and trust throughout this work. His technical insight, high standards, and continuous encouragement were essential for shaping both the system and this thesis.

My thanks also go to the CNIT colleagues for their support, discussions, and for providing the environment and resources that made this research possible. Their expertise and availability greatly enriched the practical side of this project.

I am deeply indebted to my beloved family for their unconditional love, endless prayers, and unwavering encouragement. I owe everything I am to my parents; their lifelong sacrifices, the values they instilled in me, and their constant belief in my potential have been the compass guiding me since childhood. This achievement belongs to them as much as it does to me, and I am profoundly grateful for their silent strength and devotion

I would also like to thank Ana Rosa Rodríguez García, Paolo Conci, and Roberto Carlos Jokanovics, who welcomed me and supported me in Italy like family. Their kindness, help, and presence made adapting to a new country and life much easier, and I am sincerely grateful for everything they have done for me.

I extend my sincere gratitude to my friends, especially Ejaz Ulhaq Fazli, for their unwavering support both in Italy and abroad. Our conversations and shared moments provided much-needed balance and joy throughout this process; I am profoundly grateful for their presence in my journey.

Contents

List of Figures

List of Tables

List of Abbreviations

AE Application Entity (oneM2M)

API Application Programming Interface

CIN Content Instance (oneM2M)

CSE Common Service Entity (oneM2M)

DAG Directed Acyclic Graph

DPP Digital Product Passport

E2E End-to-End

EPCIS Electronic Product Code Information Services

ESPR Ecodesign for Sustainable Products Regulation (EU)

GPS Global Positioning System

GS1 Global Standards 1

IOTA IOTA Distributed Ledger

IoT Internet of Things

JSON JavaScript Object Notation

JSON-LD JavaScript Object Notation for Linked Data

KPI Key Performance Indicator

NFR Non-Functional Requirement

oneM2M One Machine-to-Machine

P95 95th Percentile

RBAC Role-Based Access Control

R1CS Rank-1 Constraint System

SNARK Succinct Non-Interactive Argument of Knowledge

TEU Twenty-foot Equivalent Unit

ZKP Zero-Knowledge Proof

Chapter 1

Introduction

1.1 Motivation and Context

Maritime transport is the backbone of the global economy. According to the United Nations Conference on Trade and Development (UNCTAD), global shipping carries over 80% of world merchandise trade by volume [1]. In 2024, seaborne trade reached 12.7 billion tonnes, with containerised cargo alone exceeding 800 million twenty-foot equivalent units (TEUs) [1]. The sheer scale of this system creates a correspondingly large exposure to temperature-related cargo damage: industry estimates place annual losses from cold-chain breakdowns at approximately $35 billion worldwide, encompassing spoiled food, compromised pharmaceuticals, and rejected perishable shipments [2, 3]. For the pharmaceutical sector specifically, up to 50% of vaccines are wasted due to cold-chain and logistics failures, including inadequate temperature control during storage and transit [2, 4].

Against this backdrop, the European Union is introducing a regulatory instrument that will fundamentally change how product lifecycle data is managed across supply chains. The Ecodesign for Sustainable Products Regulation (ESPR), adopted as Regulation (EU) 2024/1781 and in force since 18 July 2024 [5], establishes the legal framework for *Digital Product Passports* (DPPs) — machine-readable digital records that accompany a product through its entire lifecycle, from raw material extraction to end-of-life recycling, and that will become a prerequisite for placing many regulated products on the EU market. The first mandatory DPP implementation, the Battery Passport under Regulation (EU) 2023/1542, becomes compulsory on 18 February 2027 for all electric-vehicle and industrial batteries exceeding 2 kWh placed on the EU market [6]. The European Commission's ESPR Working Plan 2025–2030, adopted on 16 April 2025, identifies eleven priority product categories — including textiles, furniture, tyres, and electronics — for subsequent DPP mandates, with roughly thirty product categories expected to require passports by the end of the decade [7].

Maritime logistics — the physical layer that moves the products subject to these DPP mandates across continents — has no standardised DPP implementation. The existing DPP initiatives (CatenaX for automotive, the EU Battery Passport, the Electronics DPP Coalition) all target manufactured goods with a single dominant data owner. Maritime container shipping is structurally different: it is multi-stakeholder, event-driven, operates under severe bandwidth and latency constraints, and handles cargo whose compliance status must be continuously verified in transit rather than declared once at manufacture. This gap between the regulatory

demand for verifiable product data and the absence of a maritime-specific DPP architecture is the starting point for this thesis.

1.2 The Privacy Paradox in Maritime Supply Chains

Consider a refrigerated pharmaceutical container in transit from Hamburg to Singapore. EU customs regulations require the importer to demonstrate that the cargo was maintained below 5 °C throughout transit — a legally binding cold-chain compliance requirement. The carrier, however, regards the precise temperature curve as commercially sensitive: it reveals voyage routing, load factors, and historical compliance performance that competitors and customers could exploit in contract negotiations. Under a conventional data-sharing model, satisfying the customs requirement forces the carrier to disclose data they consider proprietary. Withholding the data satisfies the carrier but makes regulatory verification impossible. This is the *maritime privacy paradox*.

The paradox compounds across the 8–15 stakeholders typically involved in a single container shipment [1]: the customs authority requires proof of regulatory compliance before clearance; the port authority enforces entry conditions for temperature-sensitive cargo; the insurance underwriter needs a defensible audit trail for claims assessment; the freight forwarder needs status visibility for operational coordination; the consignee needs confirmation that goods arrived within specification; the inspection agency needs verifiable evidence for certification decisions; and third-party logistics partners need to contribute their own handling and clearance events into the container's digital record without gaining access to the carrier's operational telemetry. Each of these stakeholders has a legitimate but narrowly scoped information need — and none has a legitimate need for the carrier's full operational telemetry stream. Yet conventional IT systems offer only two options: disclose the raw sensor data (satisfying the verifier but violating the carrier's confidentiality) or issue a self-declared compliance flag (satisfying the carrier but providing no independent verifiability).

Neither option is adequate under the ESPR framework. The regulation explicitly requires that DPP data be "accurate, complete and up to date" (Article 12, ESPR) and that access be granted on a need-to-know basis, "balancing transparency with protection of business-sensitive data" [5]. A system that satisfies both requirements simultaneously — verifiable compliance *without* raw data disclosure — requires a fundamentally different architectural approach.

1.2.1 Three Root Causes

Three structural deficiencies in current maritime IT systems prevent the resolution of the privacy paradox.

1.2.1.1 Lack of Supply Chain Visibility

Despite carrying over 80% of global trade, maritime container logistics suffers from persistent visibility gaps. Tracking information is fragmented across dozens of independent systems operated by carriers, terminals, customs authorities, and freight forwarders, with no unified view of a container's status across organisational boundaries. The UNCTAD Review of Maritime Transport 2025 highlights that digital systems such as maritime single windows and port community systems are improving efficiency in some countries, but many economies still lack the infrastructure for end-to-end shipment visibility [1]. This fragmentation means that when a compliance dispute arises — for example, whether a temperature excursion occurred during a port transfer or during ocean transit — no single authoritative record exists to resolve it.

1.2.1.2 Data Fragmentation and Format Incompatibility

There is no common event vocabulary across maritime supply chain participants. Carriers use proprietary telemetry formats defined by their IoT device vendors; ports rely on legacy Electronic Data Interchange (EDI) messages; customs authorities process XML-based declarations; and freight forwarders maintain internal tracking databases with bespoke schemas. The GS1 EPCIS 2.0 standard [8] was designed precisely to address this fragmentation — it defines a universal event model for supply chain visibility with standardised business semantics, sensor data extensions, and a canonical hashing algorithm for tamper-evident event identification. However, EPCIS 2.0 has seen minimal adoption in the maritime IoT domain; existing deployments focus on retail and pharmaceutical warehouse operations rather than in-transit container telemetry.

1.2.1.3 The Privacy–Verifiability Deadlock

The two preceding deficiencies — visibility gaps and format incompatibility — are engineering problems with engineering solutions (better systems, better standards). The privacy paradox is different: it is a *structural conflict* between two legitimate requirements that cannot both be satisfied by any conventional data-sharing architecture. Sharing raw data provides verifiability but sacrifices confidentiality; withholding raw data preserves confidentiality but sacrifices verifiability. Zero-knowledge proofs (ZKPs) — cryptographic protocols that allow a prover to convince a verifier that a statement is true without revealing any information beyond the truth of the statement — are the theoretical resolution to this deadlock. A ZKP-based compliance check lets the carrier prove "the temperature remained below the threshold" without disclosing what the temperature actually was. The practical question is whether ZKP overhead is acceptable for real-time maritime IoT event processing. This thesis answers that question.

1.3 Thesis Statement and Research Questions

Thesis Statement:

> *This thesis demonstrates that blockchain-anchored Digital Product Passports with privacy-preserving zero-knowledge proof compliance verification are technically feasible and performance-acceptable for maritime container logistics.*

The argument proceeds in three acts:

1. Maritime supply chains have a privacy paradox — stakeholders need compliance proof, carriers will not share raw data.
2. Ocean DPP resolves it: EPCIS 2.0 events capture supply chain semantics; Groth16 ZKPs prove compliance without revealing sensor values; IOTA Merkle anchoring provides tamper-evident immutability.
3. Sixteen experiments prove Ocean DPP is fast enough, reliable enough, and correct enough for production maritime use.

The thesis statement is addressed through four focused research questions, each answered by a specific experiment group:

RQ1: Can EPCIS 2.0 supply chain events be captured, translated, and stored in an IoT platform with sub-200 ms end-to-end latency at production-relevant throughput?

Answered by experiments E1 (baseline latency), E2 (throughput saturation), S2/S3 (horizontal scaling), and E15 (EPCIS compliance).

RQ2: Can Groth16 zero-knowledge proofs be integrated into the event pipeline with overhead acceptable for maritime IoT intervals, while providing cryptographic privacy guarantees?

Answered by experiments E5 (proof generation distribution), E10 (E2E overhead), EV1 (verification latency), and EI4 (tamper detection via forged-proof rejection).

RQ3: Do IOTA Tangle anchoring and Merkle batching provide cost-effective, tamper-evident immutability for supply chain DPPs?

Answered by experiments EI1 (anchor latency vs. batch size), EI2 (anchoring throughput ceiling), EI3 (cost per event), and EI4 (tamper evidence).

RQ4: Does the hybrid messaging architecture (MQTT QoS 1 + RabbitMQ durable queues + Dead Letter Queue + backpressure) guarantee at-least-once event delivery under realistic failure scenarios?

Answered by reliability experiments R1 (MQTT resilience), R2 (queue durability), R3 (DLQ boundary), and R4 (backpressure).

1.4 Contributions

This thesis makes four principal contributions:

C1 — Ocean DPP Architecture: A multi-service, standards-compliant event pipeline integrating EPCIS 2.0 (supply chain events), oneM2M (IoT platform standard), and GS1 CBV 2.0 (canonical event hashing). This is the first demonstrated integration of EPCIS 2.0 with oneM2M for cold-chain Digital Product Passports in the maritime domain.

C2 — ZKP-EPCIS Integration: Groth16 proof generation embedded directly in the EPCIS enrichment pipeline, with verification key pinning and a browser-side WASM verification portal accessible to external stakeholders without authentication. This contribution demonstrates the first implemented and quantitatively evaluated integration of zero-knowledge proofs with the EPCIS 2.0 standard for maritime DPPs, with a formal threat model: 304 ms mean proof generation (P95: 461 ms), 9.8 ms constant-time browser verification (P95: 12 ms), 31:1 generation-to-verification asymmetry.

C3 — IOTA Merkle Batch Anchoring: An event batching algorithm that aggregates a configurable number of event hashes into a single Merkle tree, anchoring only the root to the IOTA blockchain. The batch size is a deployment parameter that governs a cost–latency tradeoff: larger batches reduce on-chain transaction count but increase the maximum delay before an event is anchored. Experiments evaluated batch sizes of 10, 50, and 100 events per transaction, demonstrating 90–99% reduction in on-chain transaction count while preserving per-event tamper evidence via Merkle inclusion proofs. This is the first systematic evaluation of Merkle batching for maritime DPP cost-latency trade-offs.

C4 — Comprehensive Quantitative Evaluation: Sixteen experiments covering performance (E1, E2, E10), ZKP characterisation (E5, EV1, EI4), scalability (S2, S3), IOTA anchoring (EI1, EI2, EI3), data quality (E15), and reliability (R1–R4), providing reproducible baseline metrics and experimental methodology for future DPP research. The controlled benchmark series is supplemented by an exploratory field test with a commercial container tracking device (CTD) on a live refrigerated shipment, confirming correct end-to-end operation under real maritime connectivity conditions.

1.5 Scope and Limitations

1.5.1 In Scope

- Multimodal maritime container cold-chain tracking

- EPCIS 2.0 event generation, translation, and schema validation
- IOTA blockchain anchoring with configurable Merkle batching
- oneM2M multi-AE architecture for IoT device and data management
- Groth16 zero-knowledge proof generation and verification (temperature compliance circuit)
- Public verification portal for unauthenticated stakeholder access
- External logistics partner event ingestion via scoped API
- Live deployment on an internet-reachable virtual machine with Docker Compose orchestration
- Exploratory field test with a commercial container tracking device (CTD) on a real refrigerated shipment
- Comprehensive experimental evaluation (16 controlled experiments)

1.5.2 Out of Scope

- Production deployment on IOTA mainnet (localnet sufficient for evaluation)
- Large-scale commercial pilot with multiple shipping companies (single CTD field test only)
- Complete EU ESPR regulatory compliance mapping (technical feasibility focus)
- Multiple ZKP circuit types beyond temperature compliance
- Machine learning integration (identified as future work)

1.5.3 Acknowledged Limitations

1. The controlled experimental campaign used a synthetic workload generated by an emulator. After completing the implementation, the platform was deployed on an internet-reachable virtual machine and exercised with a commercial container tracking device (CTD) on a refrigerated container during a round-trip between Italy and Germany, confirming correct ingestion, EPCIS event generation, ZKP proof creation, and IOTA anchoring under real connectivity conditions. This field test was exploratory and not part of the controlled benchmark series.

2. The internet-reachable VM deployment confirms operational viability over a wide-area network, but end-to-end latency and throughput under realistic public-network conditions were not systematically characterised.

3. Limited scale testing (100 containers); production deployment would require further characterisation at 10,000+ TEU scale

4. Localnet IOTA — mainnet latencies and gas pricing may differ

5. Single ZKP circuit type (`tempBound`) — extensibility to humidity, route, or multi-condition predicates is architecturally supported but not empirically validated

6. Groth16 trusted setup used the public Hermez Powers-of-Tau Phase 1 ceremony; this is acceptable for academic work but a production deployment should run a domain-specific ceremony or adopt a universal-setup system

1.6 Thesis Outline

The remainder of this thesis is organised as follows.

Chapter 2 establishes the technical vocabulary and surveys prior work. The first part introduces the five foundational technologies — Digital Product Passports, GS1 EPCIS 2.0, oneM2M, IOTA distributed ledger, and zero-knowledge proofs — at the depth required to understand the design decisions in subsequent chapters. The second part surveys existing DPP platforms, blockchain-based supply chain traceability systems, and privacy-preserving approaches, constructing a comparative gap analysis that motivates the present work.

Chapter 3 derives functional and non-functional requirements from a stakeholder analysis, defines a formal threat model with four adversary types, and presents the complete system architecture including the ZKP integration, Merkle batch anchoring, and eight-section DPP data model.

Chapter 4 describes how the design became software: eight Docker containers (nine with the optional Nginx load balancer), the Translator's EPCIS 2.0 translation pipeline, the DPP Core enrichment engine, the ZKP proof generation module with its Circom circuit and trusted setup, the IOTA anchoring service with its gas coin pool and WASM client lifecycle, and the graceful degradation strategy.

Chapter 5 presents all 16 experiments in a uniform format (objective, method, results, finding), covering baseline latency, throughput saturation, ZKP overhead, horizontal scaling, IOTA anchoring, EPCIS 2.0 compliance, and four reliability failure-injection scenarios.

Chapter 6 synthesises findings across all four research questions, analyses ZKP cost–benefit, identifies the Mobius4 HTTP bottleneck and its horizontal scaling resolution, discusses threats to validity, compares with published systems, states limitations, proposes future work, and concludes.

Chapter 2

Background and Related Work

This chapter is organised in two parts. Sections 2.1–2.5 introduce the five foundational technologies that underpin the Ocean DPP platform: Digital Product Passports, GS1 EPCIS 2.0, oneM2M, the IOTA distributed ledger and zero-knowledge proofs. Sections 2.6–2.10 survey existing systems, construct a comparative analysis, and identify the research gaps that this thesis addresses.

2.1 Digital Product Passports and the EU ESPR Mandate

2.1.1 Concept and Motivation

A Digital Product Passport (DPP) is a structured, machine-readable digital record that accompanies a product through its entire lifecycle from raw material extraction, through manufacturing and distribution, to use, repair, and end-of-life recycling. The concept is anchored in the European Union's Ecodesign for Sustainable Products Regulation (ESPR), adopted as Regulation (EU) 2024/1781 and in force since 18 July 2024 [5]. The ESPR grants the European Commission authority to mandate DPPs for nearly all physical goods placed on the EU market, with requirements specified through product-group-specific delegated acts [5]. Early frameworks for DPP requirements and circular-economy integration have been proposed by Adisorn et al. [9] and Jansen et al. [10].

The policy goals of the DPP framework, as defined by the European Commission, are threefold [5]. First, *sustainability*: make the environmental performance of products measurable and comparable between manufacturers. Second, *circularity*: improving reuse, repair, refurbishment, and recycling by providing structured access to material composition, disassembly instructions, and component-level data. Third, *legal compliance*: simplifying market surveillance and regulatory enforcement through standardized digital documentation that can be verified programmatically.

The first mandatory implementation of DPP is the Battery Passport under Regulation (EU) 2023/1542 [6]. From 18 February 2027, all electric-vehicles and industrial batteries exceeding 2 kWh placed on the EU market must carry a QR-code-linked digital passport containing data on its carbon footprint, recycled content, performance, durability, and material composition. The European Commission's ESPR Working Plan 2025–2030, adopted on 16 April 2025, identifies eleven additional priority product categories — including textiles, furniture, tyres, iron and steel, and consumer electronics — for subsequent DPP mandates [7]. Roughly 30

product categories are expected to require passports by the end of the decade, with each delegated act triggering an 18-month compliance period after adoption.

2.1.2 DPP Data Model

The DPP data model, as specified in Article 12 of the ESPR, comprises both static and dynamic data elements [5]. Static attributes describe the product at time of manufacture: dimensions, weight, material composition, certifications, and manufacturer identity. Dynamic lifecycle events capture what happens to the product after manufacture: shipment, inspection, repair, ownership transfer, and end-of-life processing. The regulation requires that DPP data be "accurate, complete, and up to date" and that access be granted on a need-to-know basis, with different data elements visible to different stakeholder roles (regulators, recyclers, consumers, manufacturers). This access-control requirement creates an inherent tension: the richer the data in a DPP, the more useful it becomes for circular-economy objectives, but the more exposure it creates for data owners — a tension that is particularly acute in multi-stakeholder supply chains where telemetry data is commercially sensitive.

Technically, the ESPR mandates a decentralised data architecture: product data remain distributed across trusted actors and systems rather than stored in a single EU-wide database. Each DPP must be accessible via a data carrier (QR code, RFID tag, or NFC chip) that links to a unique digital identifier, and the underlying infrastructure must ensure interoperability through standardised data formats and APIs. The EU DPP registry, scheduled for establishment by 19 July 2026, will serve as a centralised index for discovering DPP endpoints without centralising the data itself [5].

2.1.3 The Maritime DPP Gap

Existing DPP frameworks target manufactured goods with a single dominant data owner. The Catena-X automotive network [11] defines DPPs for automotive components; recent blockchain-based DPP prototypes have been demonstrated for textiles [12] and generic products [13, 14]; the EU Battery Passport covers batteries; textile DPPs are in preparation under the ESPR Working Plan. These frameworks share a common assumption: a single manufacturer creates the product, attaches the DPP at the point of manufacture, and updates it at discrete lifecycle milestones (sale, repair, recycling).

Maritime container logistics breaks this single-owner assumption. A single refrigerated container shipment involves 8–15 independent stakeholders (carrier, port authorities, customs, insurers, freight forwarders, consignees) generating continuous telemetry across weeks or months of transit. The DPP must capture dynamic, high-frequency sensor events (temperature, humidity, GPS position, seal status) rather than static manufacturing attributes, and must support real-time compliance verification during transit rather than point-in-time declarations. Furthermore, maritime IoT operates under bandwidth and latency constraints (satellite uplinks,

intermittent connectivity) that manufactured-goods DPP architectures do not address. To the best of our knowledge, no existing DPP framework has been designed for or evaluated in this multi-stakeholder, event-driven maritime context — a gap that this thesis directly addresses.

2.2 GS1 EPCIS 2.0 Standard

2.2.1 Overview

The Electronic Product Code Information Services (EPCIS) standard, maintained by GS1 and published as ISO/IEC 19987, is the flagship data-sharing standard for supply chain visibility [8]. EPCIS enables the capture and sharing of event data using a common language across, between, and within enterprises. Each EPCIS event answers five dimensions: *what* (which objects are involved, identified by GS1 identifiers), *when* (event timestamp and timezone), *where* (read point and business location), *why* (business step, disposition, and transaction context), and *how* (sensor data such as temperature, humidity, or pressure readings).

EPCIS 2.0, released in June 2022, represents a major update over the previous EPCIS 1.2 standard [8]. The key enhancements relevant to this work are: (1) JSON and JSON-LD serialisation alongside the legacy XML format, making EPCIS natively compatible with modern web APIs and linked-data ecosystems; (2) a REST API for event capture and query, replacing the SOAP-based interfaces of earlier versions; (3) GS1 Digital Link URI syntax for expressing product and location identifiers as standard web URLs; and (4) sensor data extensions that allow timestamped series of business-relevant measurements (temperature, humidity, pressure, light levels) to be embedded directly within EPCIS events.

EPCIS 2.0 defines five event types [8]. *ObjectEvent* records an observation, action, or state change on one or more identified objects. *AggregationEvent* records the packing or unpacking of objects into containers. *TransactionEvent* associates objects with business transactions (purchase orders, invoices). *TransformationEvent* records manufacturing or processing steps where inputs are consumed and outputs are produced. *AssociationEvent* records relationships between objects (e.g., a sensor attached to a container). This work uses ObjectEvent exclusively, as maritime cold-chain telemetry is naturally modelled as a series of observed sensor readings on identified containers.

2.2.2 Core Business Vocabulary and Canonical Event Hashing

The GS1 Core Business Vocabulary (CBV) 2.0, published alongside EPCIS 2.0 as ISO/IEC 19988, provides the standardised terminology used within EPCIS events [15]. It defines approximately 41 business step codes (e.g., `shipping`, `receiving`, `inspecting`), disposition codes (e.g., `in_transit`, `sellable_not_accessible`), and controlled vocabularies for source/destination types, sensor measurement types, and business transaction

types.

Section 8.9.2 of the CBV 2.0 specification defines a canonical event hashing algorithm that produces a deterministic, interoperable SHA-256 digest of an EPCIS event [15]. The hash is computed from semantically significant fields only — event type, timestamps, identifiers, business step, disposition, and sensor data — rather than from the full JSON or XML serialisation. This design ensures that two independently implemented systems processing the same logical event will produce identical hash digests, regardless of field ordering, whitespace, or serialisation format. This property is essential for tamper-evident blockchain anchoring: any system that implements the CBV 2.0 hashing algorithm can independently verify the integrity of an anchored event without access to the originating platform's source code.

2.2.3 Relevance to This Work

EPCIS 2.0 was selected for Ocean DPP over alternatives (legacy EDI, proprietary JSON telemetry formats) for four reasons. First, the standardised vocabulary enables interoperability: any EPCIS-compliant system can consume Ocean DPP events without platform-specific adapters. Second, JSON-LD serialisation enables linked-data queries and is natively compatible with modern web infrastructure. Third, the sensor data extensions map naturally to maritime IoT telemetry (temperature, humidity, GPS, seal status) without requiring custom schema extensions. Fourth, the canonical hashing algorithm (CBV 2.0 §8.9.2) enables tamper-evident anchoring to a distributed ledger using a standardised, reproducible digest.

2.3 oneM2M IoT Platform Standard

oneM2M is a global partnership project established in 2012 by leading regional standards development organisations — ETSI (Europe), TIA and ATIS (North America), CCSA (China), TTA (Korea), and ARIB and TTC (Japan) — to develop globally applicable technical specifications for a common IoT service layer [16]. The objective is to provide a horizontal middleware layer between IoT devices, communication networks, and applications, standardising data management, device management, security, and access control across all industry verticals.

The oneM2M functional architecture defines three principal entities [16]. The *Application Entity* (AE) is an entity in the application layer that implements IoT application logic — for example, a fleet tracking application, a temperature monitoring service, or a sensor data publisher. The *Common Services Entity* (CSE) provides a set of common service functions (data management, device management, security, subscription and notification, discovery, and group management) that AEs consume through a standardised RESTful interface. The *Network Services Entity* (NSE) provides underlying connectivity services.

All data in a oneM2M system is organised as a hierarchical resource tree rooted at

the `CSEBase` resource [16]. Key resource types include: `<AE>` (a registered application), `<container>` (a logical grouping for data, analogous to a folder), `<contentInstance>` (CIN, an individual data record within a container), `<subscription>` (a notification policy that triggers callbacks when a resource changes), and `<accessControlPolicy>` (ACP, which governs which AEs can access which resources). Communication between AEs and CSEs uses the *Mca* reference point; communication between CSEs uses the *Mcc* reference point. Both support CRUD+N (Create, Retrieve, Update, Delete, Notify) operations over HTTP, CoAP, or MQTT bindings.

In the Ocean DPP platform, the Mobius4 open-source Node.js implementation serves as the CSE [17]. Each shipping container is represented as a set of oneM2M containers (one per DPP section), and each enriched EPCIS event is persisted as a ContentInstance within the appropriate container. This architecture provides standards-compliant interoperability: any oneM2M client can retrieve container event history without platform-specific adapters, and access control is enforced at the container or section level through ACP resources.

2.4 Blockchain and IOTA Distributed Ledger

2.4.1 Distributed Ledger Technology for Supply Chains

A distributed ledger is a shared, replicated database maintained across multiple nodes without a central authority. For supply chain applications, the key property is *tamper evidence*: once data is committed to the ledger, it cannot be modified or deleted without detection. This provides a neutral, independently verifiable audit trail that no single stakeholder controls — a critical requirement when multiple parties (carriers, customs, insurers) need to trust the same compliance record.

Two broad categories of distributed ledger are relevant. *Permissioned ledgers* (e.g., Hyperledger Fabric) restrict participation to authorised nodes, providing higher throughput and privacy at the cost of centralised governance. *Permissionless ledgers* (e.g., Ethereum, IOTA) allow any node to participate, providing stronger decentralisation and censorship resistance at the cost of higher operational complexity. This work uses IOTA, a permissionless ledger specifically designed for IoT and industrial applications.

2.4.2 IOTA Rebased Architecture

IOTA is a distributed ledger that differs architecturally from traditional blockchains. Where Bitcoin and Ethereum organise transactions into sequential blocks forming a linear chain, IOTA uses a Directed Acyclic Graph (DAG) structure — known as the Tangle [18] — that enables parallel transaction processing [19]. Silvano and Marcelino [20] evaluated IOTA's suitability for IoT data communication, demonstrating acceptable latency and throughput for

sensor-driven workloads.

In 2025, IOTA underwent its most significant protocol upgrade to date: IOTA Rebased [19]. Accepted by a governance vote of IOTA token holders in December 2024 and launched on mainnet on 5 May 2025, the Rebased upgrade replaced the legacy UTXO ledger with an object-based architecture powered by the Move Virtual Machine (MoveVM). Key characteristics of the Rebased protocol include: a fully decentralised Delegated Proof-of-Stake (dPoS) consensus via the Mysticeti protocol, removing the centralised Coordinator that governed earlier IOTA versions; native Layer 1 smart contract support through MoveVM; throughput exceeding 50,000 transactions per second with sub-second finality (≈400 ms average); and minimal transaction fees (on average 0.005 IOTA per transaction).

For supply chain anchoring, the critical primitive is the *Locked Notarization*: an on-chain object that, once created, cannot be modified or deleted [19]. IOTA Notarization, released in July 2025, provides a lightweight and verifiable mechanism for anchoring documents, credentials, or events to the IOTA ledger, creating immutable audit trails with minimal overhead. The platform uses this primitive to anchor Merkle roots of event hash batches, providing tamper-evident immutability for DPP events without writing raw data to the ledger.

2.4.3 Merkle Tree Batching for Cost Reduction

A Merkle tree is a binary hash tree in which each leaf node contains the hash of an individual data element and each internal node contains the hash of its two children [21]. The root hash cryptographically commits to the entire set of leaves: if any single leaf is modified, the root changes. This property enables two operations relevant to DPP anchoring. First, *batch anchoring*: a configurable number of event hashes (e.g., 10, 50, or 100) can be grouped into a single Merkle tree, and only the root hash is submitted to the blockchain as one transaction. This reduces on-chain transaction count proportionally to the batch size. Second, *per-event verifiability*: any individual leaf can be independently verified using a Merkle inclusion proof (the leaf hash plus its sibling hashes on the path to the root), without knowledge of any other events in the batch. The OpenZeppelin SimpleMerkleTree library [21] provides the implementation used in this work.

2.5 Zero-Knowledge Proofs

2.5.1 Cryptographic Foundations

Zero-knowledge proofs (ZKPs), introduced by Goldwasser, Micali, and Rackoff [22], are cryptographic protocols in which a *prover* convinces a *verifier* that a statement is true without revealing any information beyond the truth of the statement. The canonical example is a prover demonstrating knowledge of a password without transmitting the password itself.

Three properties define a valid ZKP system. *Completeness* requires that an honest prover can always convince an honest verifier of a true statement. *Soundness* requires that a dishonest prover cannot convince a verifier of a false statement (except with negligible probability). *Zero-knowledge* requires that the verifier learns nothing beyond the binary truth of the statement — the underlying witness data is information-theoretically hidden.

2.5.2 zk-SNARKs and the Groth16 Proving System

Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) are ZKPs that are (a) succinct — proofs are short, typically hundreds of bytes — and (b) non-interactive — no back-and-forth between prover and verifier is required. The first large-scale deployment of zk-SNARKs was the Zerocash protocol [23], which enabled fully anonymous blockchain transactions. Groth16 [24] is the most compact zk-SNARK: it produces proofs consisting of three elliptic-curve points, totalling approximately 850 bytes in JSON serialisation.

Groth16 requires a *structured reference string* (SRS) produced by a trusted setup ceremony specific to each circuit. The trust assumption is that at least one participant in the ceremony destroyed their random contribution — if every participant colluded, false proofs could be fabricated. Multi-party ceremonies with hundreds of independent contributors, such as the Hermez Network Phase 1 ceremony used in this work [25], reduce this risk to a practical negligible level for academic deployments.

2.5.3 Rank-1 Constraint Systems and Circom

ZKP circuits are expressed as Rank-1 Constraint Systems (R1CS): systems of equations over a finite field. The Circom language [26] provides a high-level syntax for writing circuits that compile to R1CS. The `circomlib` library provides reusable circuit components including comparators, hash functions, and multiplexers. The snarkjs library [27] provides the JavaScript and WebAssembly runtime for proof generation and verification, supporting Groth16, PLONK, and FFLONK proving systems.

2.5.4 ZKP for Supply Chain Compliance

Supply chain compliance verification is a natural application domain for ZKPs. The fundamental structure of a compliance check is a binary predicate: “did the temperature remain below the threshold?” The verifier (a customs officer, insurer, or port authority) needs the answer to this predicate but has no legitimate need for the underlying witness (the actual temperature readings). Traditional approaches force a binary choice: either the carrier discloses the raw sensor data (providing verifiability but sacrificing confidentiality) or the carrier issues a self-declared compliance flag (preserving confidentiality but providing no independent verifiability).

ZKPs resolve this deadlock. A Groth16 proof allows the carrier to prove that the temperature readings satisfy the compliance predicate — without revealing what those readings were. The verifier obtains a cryptographically sound guarantee that the statement is true, backed by the knowledge soundness property of Groth16 (under the q-SDH assumption over BN254), while learning nothing about the private input. The proof is non-interactive (no back-and-forth required), succinct (approximately 850 bytes), and constant-time to verify (approximately 10 ms in browser WebAssembly on commodity hardware), making it suitable for multi-stakeholder verification scenarios where the same event may be checked by customs, insurers, and port authorities independently.

2.6 DPP Systems in Supply Chain

This section surveys the principal DPP initiatives and positions each relative to the maritime cold-chain requirements identified in Section 2.1.

Catena-X (automotive). The Catena-X Automotive Network [11] is an open data ecosystem for the European automotive value chain, with approximately 190 member organisations as of 2025. It uses the Eclipse Tractus-X Dataspace Connector (EDC) to enable sovereign, standardised data exchange between manufacturers, suppliers, and service providers. Catena-X provides a Digital Product Passport application for battery passports and transmission passports, with BMW establishing over 100 digital connections to automotive suppliers via the platform by end of 2024. However, Catena-X does not include any zero-knowledge capability; data sharing is governed by policy contracts and access-control policies negotiated bilaterally through the EDC. The platform uses proprietary aspect models rather than GS1 EPCIS 2.0 for event semantics, and no empirical performance evaluation of the DPP pipeline has been published.

EU Battery Passport. The EU Battery Regulation (2023/1542) [6] mandates a digital battery passport from 18 February 2027. The technical specification defines static data requirements (carbon footprint, recycled content, performance metrics) and dynamic lifecycle data (state-of-health tracking). The battery passport follows a decentralised data model where information remains distributed across trusted actors. While the battery passport identifies the need for role-based access control (regulators see different data than consumers), it does not specify a cryptographic privacy mechanism for multi-stakeholder verification. No zero-knowledge proof layer is defined or evaluated.

Reflow (textile). The Reflow project was an EU Horizon 2020 initiative for circular textile supply chains [28]. It proposed DPPs for tracking textile products through reuse and recycling cycles, relying on a distributed ledger concept. However, Reflow used no standardised event vocabulary, provided no privacy-preserving mechanism, and produced no open-source implementation or empirical evaluation. The project concluded in 2022.

CIRPASS. The CIRPASS project (2022–2024), funded by the European Commission,

developed recommendations for DPP infrastructure including data models, identifier systems, and interoperability standards. CIRPASS produced technical reports that informed the ESPR delegated act process but did not implement a working DPP platform. It identified GS1 Digital Link and EPCIS as candidate standards for DPP data exchange but did not specify blockchain anchoring or ZKP privacy mechanisms.

2.7 Blockchain-Based Supply Chain Traceability

Blockchain technology has attracted significant research attention for supply chain management. Kshetri [29] analysed blockchain's roles in meeting key SCM objectives including cost reduction, quality assurance, and risk mitigation. Saberi et al. [30] examined adoption barriers and enablers for blockchain in sustainable supply chains. Helo and Hao [31] provided a reference implementation model for blockchain-based operations and supply chains. Despite this broad literature, none of the systems below combines EPCIS 2.0 event semantics, zero-knowledge proofs, and quantitative performance evaluation in the maritime domain.

IBM Food Trust (Walmart). The IBM Food Trust, deployed on Hyperledger Fabric [32], is the most widely cited production-scale blockchain supply chain system. Walmart adopted it for leafy greens traceability in 2018, requiring its suppliers to upload food provenance data to the permissioned ledger. The system uses a proprietary data format (not EPCIS 2.0), provides no ZKP or privacy-preserving mechanism, and does not implement DPP semantics. The only published evaluation metric is Walmart's press-release claim that product tracing time was reduced from seven days to 2.2 seconds.

TradeLens (Maersk/IBM). TradeLens was a Hyperledger Fabric deployment for maritime container documentation, launched by Maersk and IBM in 2018 [33]. It aimed to digitise shipping documents (bills of lading, customs declarations) and provide supply chain visibility. TradeLens was shut down in November 2022, with Maersk citing insufficient industry adoption — competing carriers were unwilling to share data on a platform controlled by their largest competitor. TradeLens used a proprietary data format, did not integrate EPCIS 2.0, provided no ZKP or privacy mechanism, and published no systematic performance evaluation.

VeChain. VeChain is a public blockchain platform targeted at luxury goods and pharmaceutical supply chains [34]. It provides product authentication and traceability using a proprietary event format and NFC/RFID-based product identification. VeChain does not implement ZKP, does not use EPCIS 2.0 or any standardised event vocabulary, and does not address cold-chain DPP requirements. Published evaluations cover blockchain throughput but not privacy or DPP compliance.

2.8 Privacy-Preserving Approaches in Supply Chain Systems

The intersection of zero-knowledge proofs and supply chain systems remains sparsely populated in the literature, particularly for implemented systems with quantitative evaluations. Prasad et al. [35] provide a comprehensive analysis of ZKP applications in blockchain-enabled supply chain management but report no implemented maritime system.

Silveirinha et al. [36] conducted a systematic review of ZKP applications in maritime blockchain systems, analysing 40 articles published between 2020 and 2024 from Web of Science and Scopus. The review identified growing research interest in blockchain for maritime supply chains (28% of relevant publications in 2024, up from 14% in 2020), but found that ZKP-specific research constituted only 10% of the keyword distribution. The central finding is unambiguous: no implemented, quantitatively evaluated ZKP system for maritime supply chains exists in the surveyed literature. The review identifies scalability limitations of ZKP-enhanced blockchains for high-frequency processes such as container monitoring, and notes that interoperability across different platforms (port authorities, customs, shippers) remains a major unsolved challenge.

Several conceptual proposals exist in adjacent domains. Steffen et al. [37] surveyed blockchain privacy techniques for supply chain management, identifying ZKP as a candidate mechanism but reporting no implementation for physical-goods DPPs. Udokwu and Craß (2026) proposed extending DPPs with selective disclosure using zkSNARKs but provided no implementation or empirical evaluation. Babel et al. (2025) advocated human-centric DPPs with attribute-level privacy at HICSS but presented a conceptual design only, with no circuit implementation or performance data.

The common thread across all these works is the gap between conceptual proposals and implemented, evaluated systems. This thesis fills that gap by providing the first implemented and quantitatively evaluated ZKP-EPCIS integration for maritime DPPs.

2.9 Comparative Analysis

Table 2.1 compares the most relevant prior systems across six properties that define Ocean DPP's novel claims. In the "Quant. Eval." column, "Limited" denotes systems that report only vendor-published or anecdotal figures (e.g., a press-release metric) without reproducible experimental methodology.

2.10 Identified Research Gaps

Analysis of Table 2.1 and the narrative above reveals six compounding gaps:

Table 2.1: Comparison of Digital Product Passport and Supply Chain Traceability Systems

System	Domain	Event Std.	Blockchain	ZKP	Cold-chain	Quant. Eval.
CatenaX [11]	Automotive	Proprietary	—[a]	No	No	No
Battery Passport [6]	Batteries	—	—	No	No	No
Reflow [28]	Textile	None	Conceptual	No	No	No
IBM Food Trust [32]	Food	Proprietary	Hyperledger	No	No	Limited
TradeLens [33]	Shipping	Proprietary	Hyperledger	No	No	No
VeChain [34]	Luxury/Pharma	Proprietary	VeChainThor	No	No	Limited
Silveirinha et al. [36]	Maritime	—	—	Survey only	—	—
Ocean DPP	**Maritime**	**EPCIS 2.0**	**IOTA**	**Yes**	**Yes**	**Yes (16 exp.)**

[a]Catena-X uses Eclipse Dataspace Connectors for data exchange; the underlying infrastructure can be deployed on various cloud platforms but does not use a blockchain for DPP anchoring.

1. **Maritime DPP**: No academic paper has implemented a Digital Product Passport specifically for maritime container logistics.

2. **ZKP + Maritime Supply Chain**: Silveirinha et al. [36] confirm that no implemented, quantitatively evaluated ZKP system for maritime supply chains exists as of 2025.

3. **ZKP + EPCIS 2.0**: No prior work integrates zero-knowledge proofs with the GS1 EPCIS 2.0 standard.

4. **Cold-Chain Privacy**: No existing DPP addresses the privacy paradox (verifiable compliance without raw data disclosure) for temperature-controlled cargo.

5. **IOTA for DPP**: IOTA's low-cost, IoT-targeted architecture has not been evaluated for supply chain DPP anchoring with Merkle batching.

6. **Quantitative Evaluation**: Existing DPP papers are conceptual or provide only anecdotal evidence. No paper provides a systematic performance evaluation with statistical rigour.

Gap statement: *No prior work combines EPCIS 2.0 event semantics, privacy-preserving ZKP compliance verification, IOTA blockchain anchoring, and a rigorous quantitative evaluation in the context of maritime container cold-chain logistics. Ocean DPP fills all six gaps simultaneously.*

The requirements derived in Chapter 3 are grounded directly in these gaps.

Chapter 3

System Design and Architecture

3.1 Stakeholder Analysis

3.1.1 Identified Stakeholders and Requirements

The Ocean DPP platform serves five main stakeholder categories, each with distinct data needs and competing privacy interests. Identifying these conflicts precisely motivates the architectural choice of zero-knowledge proofs as the privacy-preserving verification mechanism.

3.1.1.1 Container Operators (Carriers)

The carriers own the shipping containers and the IoT sensors attached to them. They require full access to every raw sensor observation — GPS position, temperature curve, speed, and seal status — for operational decisions. The temperature profile is commercially sensitive: it reveals voyage routes, loading patterns, and historical compliance performance. Disclosing exact readings to third parties gives competitors negotiating leverage and exposes the carrier to liability in contested claims. Consequently, the platform gives carriers unrestricted access to raw event data through the authenticated DPP Passport viewer.

3.1.1.2 External Verifiers: Customs, Insurers, and Port Authorities

Three stakeholder categories share a common data requirement: each needs independently verifiable proof that the cold chain was maintained, but none has a lawful or operational need for the underlying sensor stream. Their mandates differ only in purpose — customs officers verify regulatory compliance before clearance, insurers require a legally defensible audit trail for claims assessment, and port authorities enforce entry conditions for temperature-sensitive cargo — yet all three are satisfied by the same cryptographic artefact: a Groth16 proof attesting that the temperature remained below the declared threshold, anchored to an immutable IOTA record, without revealing the temperature itself. This convergence is a key design driver: a single ZKP layer serves all three verification roles without per-stakeholder customisation.

3.1.1.3 Logistics Partners (External Application Entities)

Third-party logistics providers contribute EPCIS events for handling, customs clearance, and port operations. The platform exposes a dedicated External AE write path (`POST /api/v1/external/events`) with container-scoped access control.

3.1.2 Privacy Conflict Matrix

Table 3.1 formalises the conflict. A conventional system can satisfy either requirement — full disclosure or opaque binary flags — but not both simultaneously, because any verifiable binary flag must either be self-asserted (unverifiable) or backed by disclosed data (privacy violation). Zero-knowledge proofs resolve this deadlock: they produce a cryptographic statement that is both independently verifiable and perfectly zero-knowledge (revealing nothing about the private input beyond the binary verdict).

External stakeholders not listed in the table (e.g., consignees, cargo insurers, third-party auditors) access compliance proofs via the unauthenticated public portal, which provides ZKP verification without requiring platform credentials or revealing raw sensor data.

Table 3.1: Stakeholder Data Access Requirements

Stakeholder	Role	Compliance Proof	Raw Sensor Data
Carrier	Data owner and operator	✓	✓
Customs	Regulatory verifier	✓	×
Insurer	Claims auditor	✓	×
Port Authority	Entry/exit clearance	✓	×
Logistics Partner	External event contributor	✓	Scoped only

3.2 Functional Requirements

3.2.1 FR1–FR3: Data Acquisition and Translation

- **FR1**: The system shall accept raw IoT sensor payloads from containerised MQTT publishers and translate them into GS1 EPCIS 2.0 ObjectEvents.
- **FR2**: Each translated event shall be assigned a globally unique, deterministic event identifier conforming to the NIST Named Information URI scheme (`ni:///sha-256;{hash}?ver=CBV2.0`).
- **FR3**: The system shall persist every translated event as a oneM2M ContentInstance (CIN) in the Mobius4 CSE resource tree.

3.2.2 FR4–FR6: Enrichment, Anchoring, and Passport Assembly

- **FR4**: The system shall enrich each event with a GS1 CBV 2.0 cryptographic hash and geolocation metadata.
- **FR5**: The system shall anchor event integrity evidence to the IOTA distributed ledger using a Merkle batch strategy (configurable batch size; one IOTA transaction per batch; per-event Merkle proofs stored in Mobius4).
- **FR6**: The system shall serve an eight-section Digital Product Passport per container via a REST API, with section visibility filtered by the requesting user's JWT role.

3.2.3 FR7: Privacy-Preserving Verification

- **FR7.1**: The system shall generate a Groth16 zero-knowledge proof for every EPCIS event containing a temperature observation, proving that the sensor reading satisfies the compliance predicate without revealing the reading. The compliance threshold shall be configurable per deployment.
- **FR7.2**: The system shall support off-chain proof verification via a REST endpoint (`POST /api/v1/zkp/verify`).
- **FR7.3**: When a ZKP proof is successfully generated, the proof's SHA-256 digest (`proofHash`) shall replace the raw event hash as the Merkle leaf submitted to IOTA.
- **FR7.4**: The proof, public signals, and verification key required for independent Groth16 verification shall be retrievable via public API endpoints.
- **FR7.5**: A public-facing verification portal shall be accessible without authentication, enabling external stakeholders to verify container compliance proofs using only a container identifier.

3.3 Non-Functional Requirements

The latency and ZKP performance thresholds below are grounded in two considerations rather than arbitrary choices. First, interactive back-end systems commonly adopt 100–200 ms server-side latency and 200–500 ms end-to-end API response time as indicative of "good" performance for non–real-time applications. Second, maritime IoT telemetry operates at relatively low frequencies (seconds between successive readings per container), so end-to-end processing times on the order of a few hundred milliseconds remain operationally acceptable. Within this envelope, this work sets NFR4 to 200 ms median pipeline latency and NFR9.1/NFR9.2 to 2 s proof generation and 20 ms verification as explicit performance targets for the prototype.

3.3.1 NFR1–NFR4: Reliability and Availability

- **NFR1**: The event ingestion pipeline shall tolerate the transient unavailability of DPP Core for up to 30 minutes without data loss.
- **NFR2**: The system shall recover automatically from AMQP connection failures using exponential back-off reconnection.
- **NFR3**: The IOTA anchoring service shall degrade gracefully when the IOTA node is unreachable.
- **NFR4**: End-to-end pipeline latency from MQTT publish to Mobius4 persistence (without ZKP) shall not exceed 200 ms at median load.

3.3.2 NFR5–NFR8: Security and Standards Compliance

- **NFR5**: All service-to-service communication inside the Docker network shall be restricted to explicitly declared routes (mitigates SSRF).
- **NFR6**: All user passwords shall be stored as bcrypt hashes with a cost factor of 12.
- **NFR7**: Event hashes submitted to IOTA shall conform to GS1 CBV 2.0 §8.9.2.
- **NFR8**: Role-based access control shall be enforced at the API Gateway boundary using JWT claims.

3.3.3 NFR9: Zero-Knowledge Proof Performance

- **NFR9.1**: Proof generation shall not exceed 2 seconds per event under normal load.
- **NFR9.2**: Proof verification shall complete within 20 ms per proof.
- **NFR9.3**: The serialised proof payload shall not exceed 1,500 bytes.
- **NFR9.4**: The ZKP pipeline shall sustain at least 300 proof-generation operations per minute as a throughput floor. *Note:* at 304 ms mean per proof, single-threaded throughput is ≈197 proofs/minute; meeting this target requires parallel proof generation across multiple worker threads, which is architecturally supported but not evaluated in this work.

3.4 High-Level Architecture

The platform is structured as containerised microservices with all message routing implemented via AMQP (RabbitMQ). The internal event enrichment pipeline (Translator, RabbitMQ, DPP Core, Mobius publisher) uses durable queues with per-message persistence and

acknowledgements. The Translator publishes translated events to two durable RabbitMQ queues: `translated_events` (consumed exclusively by DPP Core for enrichment and anchoring) and `ws_events` (consumed exclusively by the API Gateway WebSocket server for real-time dashboard broadcast). The complete data path from IoT sensor to blockchain anchor passes through seven loosely coupled components: the IoT Emulator, RabbitMQ, the Translator, Mobius4, DPP Core (with its inline ZKP and IOTA anchoring modules), the API Gateway (backed by Ocean-ADM for identity), and the Dashboard with its public verification portal.

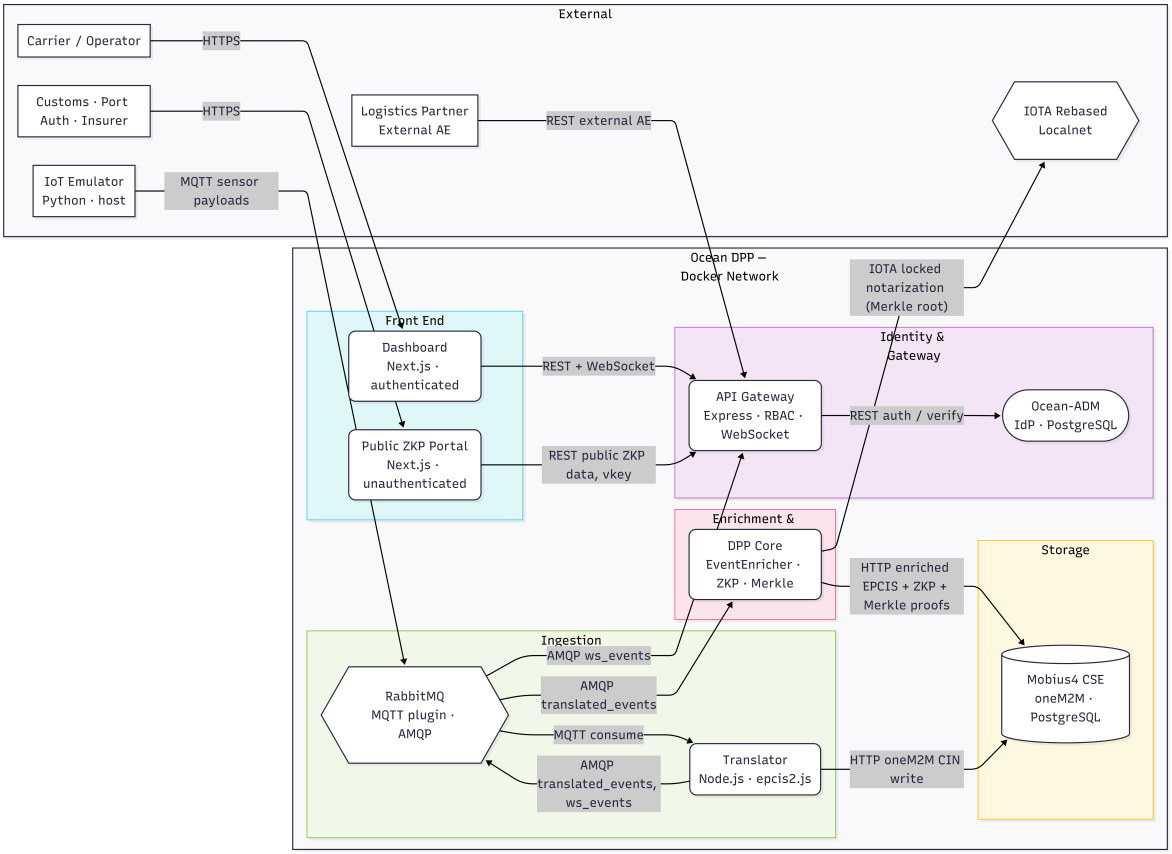

Figure 3.1: Ocean DPP platform architecture with privacy-preserving verification.

3.4.1 Component Descriptions

3.4.1.1 IoT Emulator

The IoT emulator is a Python-based tool running directly on the host (outside the Docker deployment), orchestrating synthetic voyages and emitting temperature readings, e-seal status changes (lock/unlock), and alarm events. Its primary design objective is reproducibility: controlled generation of compliant/non-compliant telemetry and exception paths enables systematic measurement of proof overhead and graceful degradation. MQTT messages are published to RabbitMQ's MQTT plugin, where they are consumed by the Translator service

and converted into EPCIS 2.0 ObjectEvents.

3.4.1.2 Mobius4 CSE (oneM2M Resource Server)

Mobius4 provides the platform's authoritative data store as a standards-compliant oneM2M Common Services Entity. The decision to persist EPCIS events as oneM2M ContentInstances (CINs) rather than rows in a relational database serves two objectives. First, it enables standard-compliant interoperability: any oneM2M client can retrieve container history without platform-specific adapters. Second, it provides a hierarchical resource tree (`/OCEAN-DPP/{containerId}/events/{section}`) that mirrors the eight-section DPP structure defined in Section 3.7, simplifying access control (ACP policies attached at the container or section level). Mobius4 manages its own PostgreSQL persistence layer internally; platform services interact exclusively via the oneM2M HTTP API, eliminating direct database coupling and enabling Mobius4 to be swapped for alternative CSE implementations (e.g., ACME, OM2M) without service rewrites. The sole exception is DPP Core's direct PostgreSQL connection for three auxiliary tables that have no oneM2M equivalent, described under PostgreSQL Databases below; these tables are private to DPP Core and do not affect CSE interoperability. The oneM2M HTTP API introduces higher per-write latency (≈80–120 ms per CIN creation) compared to direct database writes; this is the cost of standards-compliant interoperability. The evaluation (Chapter 5) demonstrates that this cost is acceptable at maritime event rates and that horizontal Mobius4 scaling resolves the bottleneck when higher throughput is required.

3.4.1.3 RabbitMQ Message Broker

RabbitMQ is the central event pipeline infrastructure, implementing AMQP 0.9.1 with a dual-protocol configuration: native AMQP on port 5672 for service-to-service communication and an integrated MQTT plugin on port 1883 for IoT device connectivity. The Translator consumes MQTT messages directly from the plugin using a persistent session so the broker buffers messages while the service is offline. The Translator publishes to two durable queues: `translated_events` (with an associated dead-letter queue `translated_events.dlq`) consumed exclusively by DPP Core for enrichment, and `ws_events` consumed exclusively by the API Gateway's WebSocket server for live dashboard broadcast. This dual-queue design prevents slow WebSocket clients from blocking event acknowledgement on the critical enrichment path. Messages are persisted to disk and acknowledged explicitly; if a consumer crashes mid-processing, RabbitMQ requeues the message. This satisfies NFR1 (30-minute fault tolerance): even if a downstream service is inaccessible, messages remain buffered until it recovers. Queue state survives broker restarts thanks to durable queues and per-message persistence.

3.4.1.4 Translator

The Translator bridges raw MQTT sensor payloads to GS1 EPCIS 2.0 ObjectEvents. Its critical design feature is a 24-hour LRU idempotency cache (keyed by `eventID`): if the IoT gateway or Mobius notification channel replays a message, the Translator recognises the duplicate and suppresses it. Each message is processed in two sequential steps: (1) persist to Mobius4 via HTTP with retries; (2) publish the EPCIS event to both durable RabbitMQ queues. These steps are not a distributed transaction; if the RabbitMQ publish fails after Mobius persistence, the event is sent to the DLQ or left unacked so the persistent MQTT session will redeliver it. This provides at-least-once delivery end-to-end, meeting NFR1.

3.4.1.5 DPP Core: EventEnricher and ZKP Module

DPP Core orchestrates event enrichment (GS1 CBV hash, geolocation metadata, and ZKP proof generation) synchronously, then schedules blockchain anchoring as a non-blocking background operation. The enrichment pipeline completes immediately without waiting for the IOTA Merkle batch transaction; anchor results (Merkle proofs, notarization identifiers, transaction digests) are posted to separate `iota-anchors` containers in Mobius4 after batch confirmation, typically within 30 seconds of the enriched event being persisted. The ZKP module runs inline in the enrichment pipeline rather than as a separate microservice, trading isolation for lower latency. The pipeline implements graceful degradation: if ZKP proof generation fails (e.g., missing temperature field or circuit unavailability), the enricher logs the failure, falls back to the raw GS1 event hash, and continues anchoring. ZKP availability is best-effort; traceability proceeds even when proofs are skipped.

3.4.1.6 PostgreSQL Databases

Two independent PostgreSQL instances support the platform. (1) **Mobius4 Database**: Mobius4 maintains its oneM2M resource tree (container hierarchies, CIN metadata, ACPs) in a dedicated PostgreSQL container. The schema is managed entirely by Mobius4; all EPCIS event access is through the oneM2M HTTP API, ensuring data consistency and enabling future CSE swaps. DPP Core additionally writes three auxiliary tables (`dpp_snapshots`, `dpp_subscriptions`, `dpp_audit_logs`) directly via `pg.Pool`; these are internal to DPP Core and have no oneM2M counterpart. (2) **Ocean-ADM Database**: Ocean-ADM persists user credentials (usernames, bcrypt password hashes), JWT refresh tokens, and role assignments in a separate PostgreSQL instance. The schema is private to Ocean-ADM; the API Gateway and other services authenticate against Ocean-ADM's `POST /auth/verify` endpoint rather than querying the database directly. Both databases are confined to the Docker internal bridge network and are not exposed to external clients.

3.4.1.7 Ocean-ADM (Identity Provider)

Ocean-ADM provides JWT-based authentication. The API Gateway translates Ocean-ADM roles to a five-tier gateway role hierarchy: `admin` (full write access, user management), `operator` (read/write events, view raw sensor data), `external_ae` (scoped write access for logistics partners), `service-user` (internal service-to-service communication), and `viewer` (read-only DPP sections). The public ZKP verification portal operates without authentication and requires no role. Passwords are stored as bcrypt hashes with per-user salts; legacy SHA-256 hashes are supported for backward compatibility but trigger a warning during login.

3.4.1.8 API Gateway (Reverse Proxy with RBAC Enforcement)

The API Gateway (Express.js with RBAC middleware) acts as the sole external entry point, enforcing JWT role checks and proxying requests to internal services. The gateway uses an explicit allow-list of proxy routes rather than wildcard-based proxying. This design mitigates Server-Side Request Forgery (SSRF): an attacker cannot manipulate the URL to access internal endpoints (e.g., Mobius4 admin panel, RabbitMQ management UI) because the gateway rejects any path not in the allow-list. Role-based access is enforced before proxying. The gateway also operates a secure WebSocket server (`/ws`) that consumes the dedicated `ws_events` RabbitMQ queue and broadcasts arriving EPCIS events to authenticated dashboard clients in real-time, providing immediate updates without polling.

3.4.1.9 Dashboard and Public ZKP Portal

The Ocean DPP Dashboard (Next.js) serves two distinct verification paths. The authenticated dashboard (`/dashboard`) displays the full DPP passport (eight sections, defined in Section 3.7), including raw `sensorElementList` data for users with `operator` role. The public portal (`/portal`) implements a zero-trust verification model: the Groth16 verification key is compiled into the client bundle at build time, and when a stakeholder enters a container identifier the portal verifies the proof entirely in-browser using the pinned key. The server cannot influence the verification outcome because the key is hardcoded; the portal optionally fetches the server's key fingerprint and displays a security warning if it diverges from the pinned value, providing tamper detection without loss of cryptographic integrity. The concrete implementation of both paths — including the build-time key export pipeline, deferred `snarkjs` WASM loading, and API endpoint structure — is described in Section 4.7.

3.5 Threat Model and Security Analysis

With the platform's components and data flows established in Section 3.4, it is now possible to reason precisely about what the system protects, against whom, and where its limits lie. This

section defines the adversary model, the security properties Ocean DPP claims to provide, and the properties it explicitly does not claim.

3.5.1 Adversary Model

Four adversary types are in scope:

Passive network observer An adversary who can intercept network traffic between platform components. *Mitigation:* All internal service communication is restricted to the Docker bridge network (NFR5); raw sensor values traverse the internal pipeline (MQTT → Mobius4 → Translator → AMQP → DPP Core) but are never exposed on any external interface. TLS is applied on all external-facing endpoints.

Malicious verifier A legitimate stakeholder (e.g., a customs officer or insurer) who attempts to extract the raw sensor value from the ZKP proof or the public portal response. *Mitigation:* Groth16 provides *perfect* zero-knowledge: the verifier sees only the proof object (three elliptic-curve points), the public signals (`compliant`, `max_temp`), and the verification key. The actual temperature (`actual_temp`) is a private input to the circuit and is never serialised or transmitted by the verification API. Because Groth16's zero-knowledge property is perfect (i.e., the simulator's output is identically distributed to a real proof), no computationally unbounded adversary can extract the private input from a valid proof. To prevent offline brute-force enumeration of the bounded temperature range (≈1,250 values at 0.1 °C resolution), the platform employs a keyed commitment scheme: a platform-wide secret K_{anchor} (injected via the `ANCHOR_SECRET_KEY` environment variable) is used to derive a unique per-container key $K_c = \text{HMAC-SHA256}(K_{anchor}, containerId)$, which serves as the private salt input to the ZKP circuit. An adversary who obtains a database dump — including container identifiers and anchored event hashes — cannot derive K_c without K_{anchor}, rendering offline enumeration infeasible. If K_{anchor} itself is compromised (e.g., through server-environment access), the privacy guarantees revert to those of a standard hash; however, the *soundness* property of Groth16 is unaffected: a false compliance claim still cannot be proven.

Compromised Mobius4 instance An adversary who obtains read access to all ContentInstance resources stored in Mobius4. *Acknowledged limitation:* Raw sensor values are retained in the stored EPCIS events (DPP Core and Translator both persist full GS1 EPCIS JSON, including `sensorElementList`, as CINs in Mobius4). A party with authenticated read access can recover raw temperature readings from these records. The ZKP system does not provide confidentiality against an adversary who can read Mobius4 directly — ZKP privacy is scoped to the *external verification path*: parties

interacting exclusively via the public portal or `POST /api/v1/zkp/verify` endpoint learn only the binary compliance verdict. Restricting raw-data access to the `operator` and above JWT roles (NFR8) and confining Mobius4 to the Docker bridge network (NFR5) constitute the primary mitigations for this threat.

Proof forgery An adversary who fabricates a Groth16 proof claiming compliance for a container that was non-compliant. *Mitigation:* Groth16 knowledge soundness guarantees that a valid proof can only be produced by a prover who possesses a valid witness satisfying the circuit constraints. Forgery is computationally infeasible under the q-SDH (Strong Diffie-Hellman) assumption over BN254. Additionally, the IOTA Merkle anchor ties each proof hash to a specific blockchain transaction; a forged proof submitted post-hoc would fail Merkle inclusion verification against the on-chain root (tested in Experiment EI4: 100% tamper detection rate).

3.5.2 Security Properties Provided

Compliance verifiability ✓ Groth16 knowledge soundness guarantees that a valid proof can only be produced for a true compliance statement (under the q-SDH assumption). Any third party can verify a proof independently using the pinned verification key and the `snarkjs` WASM verifier.

Data confidentiality (external verification path) ✓ The ZK property of Groth16 ensures that a party verifying a proof via the public portal or REST endpoint learns nothing about the private input (`actual_temp`) beyond the binary compliance verdict. As established in the malicious verifier analysis above, this guarantee is *perfect* (information-theoretic).

Tamper evidence ✓ Every proof hash is included in an IOTA Merkle anchor. Any modification to a stored proof invalidates its Merkle inclusion proof against the immutable on-chain root.

Replay prevention ✓ The IOTA notarization identifier, event timestamp, and HMAC-derived container key together uniquely bind each proof to a specific event. Replaying a proof from a compliant event for a non-compliant event is detectable via the circuit's public signals.

3.5.3 Properties Not Claimed

Network anonymity Not a design goal. Stakeholders are authenticated via JWT; their access patterns are logged.

Metadata privacy Event frequency, timing, and container identifiers are observable to any party with network access. A traffic analysis adversary could infer container activity patterns from the event rate alone.

Quantum resistance Groth16 relies on elliptic-curve pairings over BN254; it is not post-quantum secure. This is acknowledged as a limitation and identified as future work (Section 6.8).

Threshold confidentiality The compliance threshold (`max_temp` = 19 °C in the evaluation configuration) is a *public* input to the circuit and is visible to any verifier. Commodity-specific threshold confidentiality would require a hiding commitment scheme.

Raw data confidentiality Authenticated platform users with the `operator` role or above can retrieve the full EPCIS event (including `sensorElementList` with the raw temperature float) from the DPP Passport or Mobius4 browser endpoints. The ZKP architecture does not encrypt stored events; raw-data access is controlled solely by JWT role enforcement at the API Gateway.

Verification key auditability While the verification key is pinned in the client-side portal, the `vkeyHash` fingerprint is not currently persisted in Mobius4 ContentInstances alongside each proof. Future circuit upgrades would require out-of-band communication of which key corresponds to which archived proof batch.

3.6 Key Architectural Decisions

3.6.1 Hybrid Messaging Architecture: RabbitMQ over oneM2M Notifications

The internal event pipeline uses RabbitMQ AMQP rather than oneM2M HTTP subscriptions for service-to-service communication. oneM2M notifications are best-effort HTTP POSTs: if the subscriber is temporarily unavailable, the notification is lost. RabbitMQ provides AMQP acknowledgements, durable queues, per-message persistence, and a Dead Letter Queue. Together these satisfy NFR1 (30-minute outage tolerance).

3.6.2 IOTA Locked Notarization with Merkle Batching

The Merkle tree data structure, originally proposed by Merkle [38], enables efficient and tamper-evident verification of data membership through logarithmic-depth hash chains. Raw event data is never written to any public ledger. Only a SHA-256 digest — or a ZKP proof hash — is anchored. Merkle batching reduces the IOTA transaction count proportionally to batch size (e.g., a batch of 100 events requires one transaction instead of 100). Individual

event integrity remains independently verifiable using the sibling proofs stored in Mobius4 and the OpenZeppelin `SimpleMerkleTree.verify()` function with no knowledge of other batched events.

3.6.3 Event-Level Merkle Anchor Verification

The batching design described above anchors only the Merkle *root* to IOTA — individual event hashes are leaves inside the batch, not on-chain. This creates a verification gap: a third party holding an event CIN from Mobius4 cannot independently confirm it was included in the anchored batch without a dedicated verification path.

The platform addresses this gap through a designed five-step cryptographic verification path that uses data already stored in Mobius4 by the anchor pipeline, requiring no additional on-chain writes. Figure 3.2 illustrates the verification sequence.

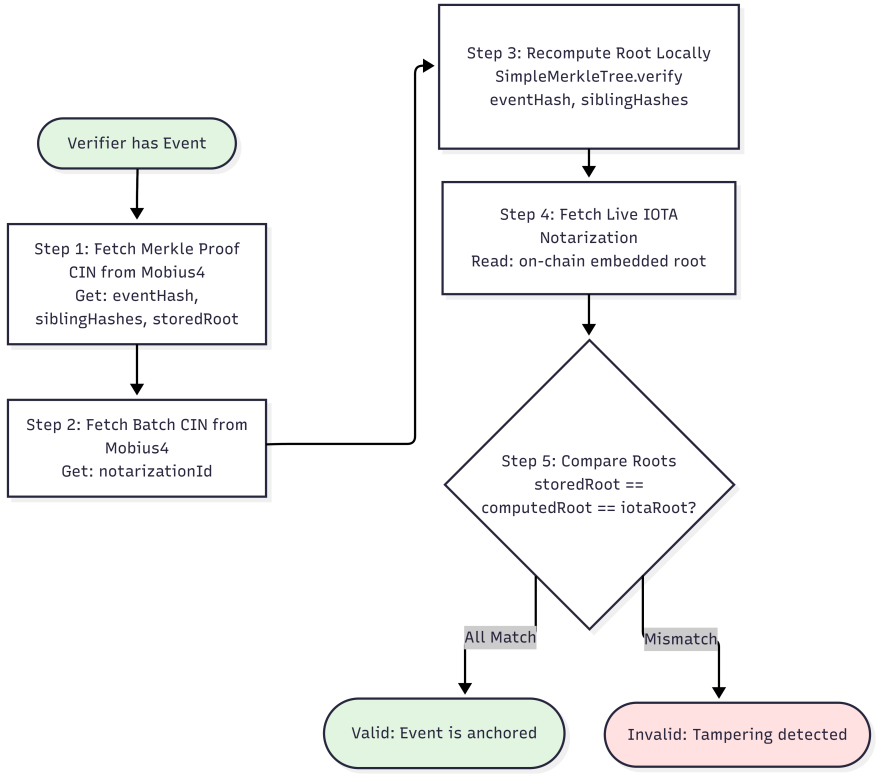

Figure 3.2: Five-step cryptographic verification path. A verifier fetches per-event and batch CINs from Mobius4, recomputes the Merkle root locally using `SimpleMerkleTree.verify()`, retrieves the live IOTA notarization, and compares all three roots. The result is a binary `valid`/`invalid` judgement that detects both off-chain CIN tampering and on-chain root substitution.

The verification process executes as follows: (1) fetch the per-event Merkle proof CIN from Mobius4 to retrieve the event hash, sibling hashes, and stored Merkle root; (2) fetch the batch CIN to retrieve the IOTA notarization identifier; (3) recompute the root locally using `SimpleMerkleTree.verify()`; (4) fetch the live IOTA notarization and read

its embedded root; (5) compare the two roots — both must match. This binary judgement detects both Merkle proof tampering (modified off-chain CINs) and root substitution (on-chain payload mismatch, which a locked notarization prevents). A production endpoint that automates steps 1–5 is identified as future work.

This verification path is equivalent to a standard Merkle inclusion proof: the verifier recomputes the Merkle root from the leaf hash and its sibling hashes and checks equality with the notarized root stored on IOTA. The design assumes that Mobius4 provides an authentic copy of the per-event and batch CINs; verifiers that maintain independent CIN copies can detect any Mobius4 tampering, while any mismatch between recomputed and on-chain roots signals on-chain or anchoring-level manipulation.

3.6.4 GS1 CBV 2.0 Canonical Event Hashing

An early design hashed a JSON-canonicalised serialisation of the complete event object. This was not interoperable: field ordering, whitespace, and implementation-specific extensions caused the same logical event to produce different digests on different systems. The adopted design derives the hash exclusively from the semantically significant fields defined in GS1 CBV 2.0 §8.9.2, in a deterministic order, making the hash reproducible by any GS1-compliant implementation without access to Ocean DPP source code.

3.6.5 Verification Key Pinning for the Public Portal

The Groth16 verification key is bundled into the client-side portal at build time, rather than being fetched from the server at verification time. This is a stronger trust model than server-assisted verification: even if the Ocean DPP backend is compromised, an attacker cannot substitute a different verification key to accept forged proofs. The portal fetches the server's key fingerprint and warns if it diverges from the pinned value, providing tamper detection without loss of the zero-trust verification guarantee. The implementation mechanics of key pinning and the build-time export script are described in Section 4.4.

3.6.6 Groth16 over Alternative ZKP Systems

Table 3.2 summarizes the trade-off space.

Table 3.2: Zero-Knowledge Proof System Comparison.[1]

System	Proof Size	Verify Time	Trusted Setup	Quantum-Safe
Groth16 (chosen)	∼850 B	∼10 ms	Yes (per-circuit)	No
PLONK	∼2 KB	∼10 ms	Universal	No
STARKs	∼50 KB	∼100 ms	None	Yes
Bulletproofs	∼1.5 KB	∼500 ms	None	No

Groth16 was selected primarily because its proof size (∼850 bytes) is two orders of magnitude smaller than an equivalent STARK (∼50 KB). This matters for maritime satellite uplinks, where per-byte transmission costs are significant [36] and NFR9.3 imposes a 1,500-byte ceiling. Verification time (∼10 ms) satisfies NFR9.2 with an order-of-magnitude margin. The per-circuit trusted setup is the principal trade-off; this is mitigated by using the Hermez Network Phase 1 Powers-of-Tau ceremony, which is the standard approach for academic Groth16 deployments. The trust assumption is that at least one ceremony participant acted honestly and destroyed their random contribution — a reasonable assumption given the public, multi-party nature of the ceremony and the 2^{12} constraint capacity of the `pot12` file, which is far in excess of the `tempBound` circuit's 17 non-linear constraints.

3.7 Data Models: Input Types, DPP Structure, and ZKP Extensions

The architectural decisions in Section 3.6 — GS1 canonical hashing, Groth16 proof generation, and Merkle anchoring — all produce structured data that must be persisted and exchanged between components. This section defines the three input data models accepted by the platform, the eight-section DPP passport structure, the role-based access control matrix, and the ZKP extension appended to each enriched EPCIS event.

3.7.1 Three Input Data Types

The platform accepts three distinct input types from IoT devices. Each type is published to the same MQTT topic (`/onem2m/notifications/mobius2_data`) but carries a different payload structure. The Translator differentiates them by checking fields in priority order: if the payload contains an `e-seals` array, it is routed to the e-seal translator; if it contains an `alarms` array, it is routed to the alarm translator; otherwise, it is processed as a telemetry event. Table 3.3 summarises the three types.

Telemetry events are the primary data source. Each event carries the complete sensor suite: temperature, humidity, GPS coordinates, speed, heading, battery level, signal strength, accelerometer, and e-seal detection status. The emulator generates telemetry at a configurable interval (default 60 s per container) [42, 43]. Every telemetry event containing a temperature observation triggers Groth16 ZKP proof generation.

Alarm events are emitted asynchronously when a sensor reading crosses a configured threshold (onset) or returns within range (recovery). The alarm payload includes the triggering sensor, the alarm type (high/low/recovery), and the current sensor readings at alarm time.

[1]Proof sizes and verification times are representative values from the original publications: Groth16 [24], PLONK [39], STARKs [40], Bulletproofs [41]. Values depend on circuit complexity and hardware; they are not measured by this work.

Table 3.3: Three Input Data Types: Trigger, EPCIS Mapping, and ZKP Applicability

Input Type	Trigger	EPCIS bizStep / disposition	Key Payload Fields	ZKP?
Telemetry	Periodic (default 60 s)	`transporting` / `in_transit`	temp, humidity, GPS, speed, heading, battery, signal	✓
Alarm	Threshold transition (onset or recovery)	`transporting` / `dpp:alarm_condition`	temp, humidity, pressure, alarm sensor & type	✓
E-Seal	Seal state change (CLOSED → TAMPERED → OPEN)	`inspecting` / `container_open` or `container_closed`	seal status, chip ID, seal ID, correlation ID	×

Since alarm events always include a temperature field, they also trigger ZKP proof generation. The Translator maps alarm events to the custom disposition `dpp:alarm_condition` and annotates the offending sensor report with an `exception` field (`dpp:SensorWarning` or `ALARM_CONDITION`).

E-seal events are emitted on physical seal state transitions. The emulator models a three-state seal machine: CLOSED → TAMPERED (after sustained motion ≥2 s) → OPEN (after continued motion ≥8 s), and OPEN/TAMPERED → CLOSED (after motion stops for ≥5 s). E-seal events carry no sensor readings and therefore do *not* trigger ZKP proof generation; the HMAC-SHA256 event hash is used directly as the Merkle leaf. The Translator maps seal events to `BizStep-inspecting` with the appropriate `container_open` or `container_closed` disposition and records a `persistentDisposition` for tamper and unauthorised-open states.

All three input types follow the same downstream pipeline: EPCIS 2.0 translation, GS1 canonical hashing, enrichment (with or without ZKP), Merkle batching, and IOTA anchoring.

3.7.2 Eight-Section DPP Structure

The Ocean DPP Passport is structured as eight oneM2M containers created per shipping container, following the resource tree shown in Figure 3.3. Sections 1–3 and 5–8 are *static*: populated once from a catalog template when the first event for a container arrives (container seeding). Section 4 (EPCIS-Events) is *dynamic*: each enriched event is stored as a separate ContentInstance (CIN), accumulating over the container's lifetime.

Table 3.4 summarises each section's purpose, field count, data source, and population strategy.

In addition to these eight user-facing sections, the platform creates an `iota-anchors`

```
Mobius/                          CSE Base
  DPP-Core/                    AE (Application Entity)
       {containerId}/          cnt (per ISO 6346)
            identification/    Section 1 (static)
            dimensions/        Section 2 (static)
            manufacturer/      Section 3 (static)
            epcis-events/      Section 4 (dynamic, 1 CIN/event)
            maintenance/       Section 5 (static)
            sustainability/    Section 6 (static)
            supply-chain/      Section 7 (static)
            documentation/     Section 8 (static)
            iota-anchors/      Internal (merkle-batch + merkle-proof CINs)
            DPP-Complete/      Snapshot container
```

Figure 3.3: oneM2M resource tree per shipping container. Resource names are slugified from the section display names. The `iota-anchors` container is internal and not part of the DPP passport API response.

Table 3.4: Eight-Section DPP Passport: Summary

#	Section	Purpose	Fields	Static/Dyn.	Source
1	Identification	Container identity, owner, type, e-seal status	10	Static	Catalog
2	Dimensions	External/internal dimensions, weight, material	12	Static	Catalog
3	Manufacturer	Factory details, materials, certifications	16	Static	Catalog
4	EPCIS-Events	Enriched events with ZKP + IOTA anchoring	17/event	Dynamic	Pipeline
5	Maintenance	Service history, inspection records	11	Static	Catalog
6	Sustainability	Carbon footprint, recyclability, end-of-life	10	Static	Catalog
7	Supply-Chain	Partners, origin/destination, traceability flags	9	Static	Catalog
8	Documentation	Manuals, safety data sheets, certificates	10	Static	Catalog

container for each shipping container to store blockchain anchoring artefacts. This container holds two CIN types: `merkle-batch` records (one per batch, containing the Merkle root, leaf count, notarization ID, and transaction digest) and `merkle-proof` records (one per event, containing the event hash, sibling path, leaf index, and a reference to the parent batch). The `iota-anchors` container is *not* part of the DPP passport API response; it is accessed by the verification subsystem to validate event integrity against the IOTA ledger.

3.7.3 Role-Based DPP Section Access

The DPP passport builder filters sections based on the requesting user's JWT role. Table 3.5 shows the role-to-section mapping.

Table 3.5: Role-Based Section Access Matrix

Role	§1	§2	§3	§4	§5	§6	§7	§8
Manufacturer	✓	✓	✓	×	×	×	×	✓
Carrier	✓	×	×	✓	×	×	✓	×
Customs	✓	×	×	✓	×	✓	✓	×
Auditor	✓	✓	✓	✓	✓	✓	✓	✓
Admin/Operator	✓	✓	✓	✓	✓	✓	✓	✓
External AE	*No DPP access; uses dedicated `/external/timeline` endpoint*							

The `external_ae` role receives zero DPP sections. Logistics partners access only a stripped event timeline via the dedicated `/api/v1/external/containers/{id}/timeline` endpoint, which returns event metadata and anchor status without raw telemetry or ZKP proof objects.

3.7.4 ZKP Data Model

When ZKP proof generation succeeds, an additional `zkp` object is appended to the enriched EPCIS event before persistence. The object contains the following fields:

- **circuitName** — identifies which circuit was used (e.g., `tempBound`)
- **proof** — Groth16 proof object: three elliptic-curve point arrays (`pi_a`, `pi_b`, `pi_c`), protocol identifier (`groth16`), and curve (`bn128`)
- **publicSignals** — ordered public outputs from the circuit: `publicSignals[0]` is the compliance flag (`"1"` = compliant, `"0"` = violated), `publicSignals[1]` is the threshold scaled by $\times 10$ (e.g., `"190"` = 19.0 °C). The actual temperature remains a private witness and never appears in the public signals
- **proofHash** — SHA-256 digest of the proof and public signals, used for Merkle anchoring

- **compliant** — convenience boolean decoded from `publicSignals[0]`
- **meta** — auxiliary fields: threshold value, circuit name, scale factor, proof generation time (`proofGenMs`), salt (container ID), and generation timestamp

When ZKP is not applicable (e.g., e-seal events with no temperature sensor data), the `zkp` field is set to `null` and the HMAC-SHA256 `eventHash` is used as the Merkle leaf instead.

Not Stored: The verification key hash (`vkeyHash`) is not persisted in the CIN. The verification key is pinned in the client-side portal at build time and served via `GET /api/zkp/vkey`. A future enhancement (see Section 6.8) would include `vkeyHash` in the `zkp` object to enable auditability of circuit upgrades.

Chapter 4

Implementation

4.1 Overview and Technology Stack

The Ocean DPP platform comprises five Node.js microservices (Translator, DPP Core, API Gateway, Ocean-ADM, and Mobius4), two PostgreSQL databases, a RabbitMQ AMQP broker, an IOTA Rebased localnet node, and a Next.js front-end serving both authenticated and public routes, all orchestrated via Docker Compose. The architectural rationale for each component is presented in Section 3.4; this chapter focuses on the concrete implementation. Table 4.1 enumerates the key technology choices.

Table 4.1: Technology Stack

Layer	Component	Technology
IoT Platform	Device and data management	Mobius4 [17] (oneM2M)
Translation	IoT → EPCIS	Node.js 20, epcis2.js 2.7 [44]
Message Bus	Async communication	RabbitMQ 3 [45] (AMQP 0-9-1)
ZKP	Circuit / proof / verify	Circom 2 [26], snarkjs 0.7 [27], circomlib 2
Blockchain	Immutable anchoring	IOTA Rebased [19], @iota/notarization 0.1
Blockchain	Merkle tree	@openzeppelin/merkle-tree 1.0 [21]
Identity	Auth and JWT issuance	Ocean-ADM (Node.js 20)
API	REST + WebSocket gateway	Express.js 4 [46]
Dashboard	Authenticated UI & Public Portal	Next.js 14 [47]
Deployment	Orchestration	Docker Compose [48]

4.1.1 Deployment Topology

The platform is deployed as up to nine Docker containers on a single bridge network (`ocean-dpp-net`), orchestrated by Docker Compose. Table 4.2 lists each container and its role.

Startup ordering is enforced via health-check dependencies: both PostgreSQL instances must pass `pg_isready` before Mobius4 and Ocean-ADM start; RabbitMQ must report healthy before the Translator and DPP Core connect; and Mobius4 must be serving before any service that writes CINs. All containers use `restart: unless-stopped` for automatic recovery from transient failures. DPP Core's `container_name` is intentionally left unset to allow horizontal scaling via `docker compose up -scale dpp-core=N`; a separate override file (`docker-compose.scale.yml`) routes Mobius4 traffic through

Table 4.2: Docker Compose Deployment Topology

Container	Role
`ocean-dpp-postgres-db`	Mobius4 persistence (PostGIS 15)
`ocean-dpp-postgres-auth`	Ocean-ADM credentials and roles (PostgreSQL 14)
`ocean-dpp-rabbit`	AMQP broker + MQTT plugin (RabbitMQ 3)
`ocean-mobius4`	oneM2M CSE (Mobius4)
`ocean-nginx-mobius`	Nginx reverse proxy for Mobius4 scaling
`ocean-dpp-ocean-adm`	Identity provider and JWT issuance
`ocean-dpp-translator`	MQTT → EPCIS translation
`ocean-dpp-api-gateway`	REST + WebSocket entry point with RBAC
`ocean-dpp-core`	Enrichment, ZKP, and IOTA anchoring

the Nginx reverse proxy when multiple Mobius4 replicas are active. Nginx's role as a load-balancing reverse proxy is not limited to Mobius4: the same pattern can be applied to any stateless service in the stack—such as the API Gateway or Translator—by adding a corresponding upstream block and scaling the target service with the `-scale` flag, making Nginx the general horizontal-scaling entry point for the entire platform.

During the controlled experimental campaign (Chapter 5), the stack ran on a single-host laptop environment. After the experiments were completed, the identical Docker Compose stack was deployed on a cloud-hosted virtual machine reachable over the public internet, with TLS termination at the API Gateway. This internet-reachable deployment was used to conduct an exploratory field test with the commercial container tracking device (CTD) described in [43], mounted on a refrigerated container during a round-trip between Italy and Germany. The CTD transmitted live MQTT telemetry throughout the voyage; the platform successfully ingested the messages, translated them into EPCIS 2.0 events, generated Groth16 proofs for temperature compliance, and anchored event hashes to the IOTA Tangle. This field test was not instrumented as a controlled performance experiment; it is reported as qualitative evidence that the architecture operates correctly with real maritime IoT hardware under wide-area network conditions.

4.2 Translator Service

The Translator uses the `epcis2.js` reference library to construct standards-compliant EPCIS 2.0 ObjectEvents from raw MQTT payloads. It accepts the three input types defined in Section 3.7.1 — telemetry, alarm, and e-seal — and routes each to the appropriate translation function based on payload field inspection:

1. If the payload contains an `e-seals` array → `translateESealEvent()`: maps seal state transitions (CLOSED, TAMPERED, OPEN) to `BizStep-inspecting`

with `container_open` or `container_closed` disposition and a `persistentDisposition` for tamper states.

2. Else if the payload contains an `alarms` array → `translateAlarmEvent()`: maps threshold breaches to `BizStep-transporting` with `dpp:alarm_condition` disposition and annotates the offending sensor report with an `exception` field.

3. Else → `translateTelemetryEvent()`: maps the full sensor suite (temperature, humidity, GPS, speed, heading, battery, signal strength) to `BizStep-transporting` with `Disp-in_transit`.

All three translation paths produce a GS1-compliant `ObjectEvent` with JSON-LD context, sensor reports using GS1 and `dpp:` namespace URIs, a geo-URI read point (when GPS is available), and a deterministic event identifier using the NIST Named Information URI scheme (satisfying FR2). Platform-specific extensions (cellular metadata, accelerometer readings, GNSS quality metrics) are placed in a `dpp:extension` block to maintain EPCIS 2.0 schema compliance while preserving domain-specific data.

Two non-trivial mechanisms prevent data corruption under failure. First, a 24-hour LRU idempotency cache (keyed by event identifier) suppresses duplicate deliveries from the MQTT broker or Mobius4 notification channel without redundant downstream writes. Second, the MQTT consumer uses a stable client identifier with a persistent session (`clean=fa lse`), ensuring that messages buffered during a Translator restart are delivered to the same session queue rather than lost to an orphaned queue.

A third mechanism addresses cascading failure under burst load. The Translator enforces a configurable concurrency ceiling on in-flight Mobius4 HTTP requests. When the Mobius4 response latency exceeds a threshold, a circuit breaker opens and reduces the effective concurrency limit, throttling inbound processing until the downstream service recovers; when latency returns to normal, the limit is restored. Messages that cannot be processed within the reduced limit are left unacknowledged on the MQTT session queue and redelivered after recovery. This adaptive concurrency pattern prevents a slow Mobius4 instance from causing unbounded memory growth in the Translator process.

4.3 DPP Core and Enrichment Engine

Figure 4.1 illustrates the `EventEnricher` pipeline. Each dequeued AMQP message passes through seven sequential steps: extract and normalise the container identifier from `epcList[0]`; seed the eight-section oneM2M container hierarchy on first encounter; compute the GS1 CBV 2.0 event hash; generate a Groth16 ZKP proof if a temperature observation is present; select the anchor hash (`proofHash` if ZKP succeeded, otherwise the plain event hash,

satisfying FR7.3); schedule a non-blocking IOTA Merkle anchor; and persist the enriched event to Mobius4, then ACK to RabbitMQ.

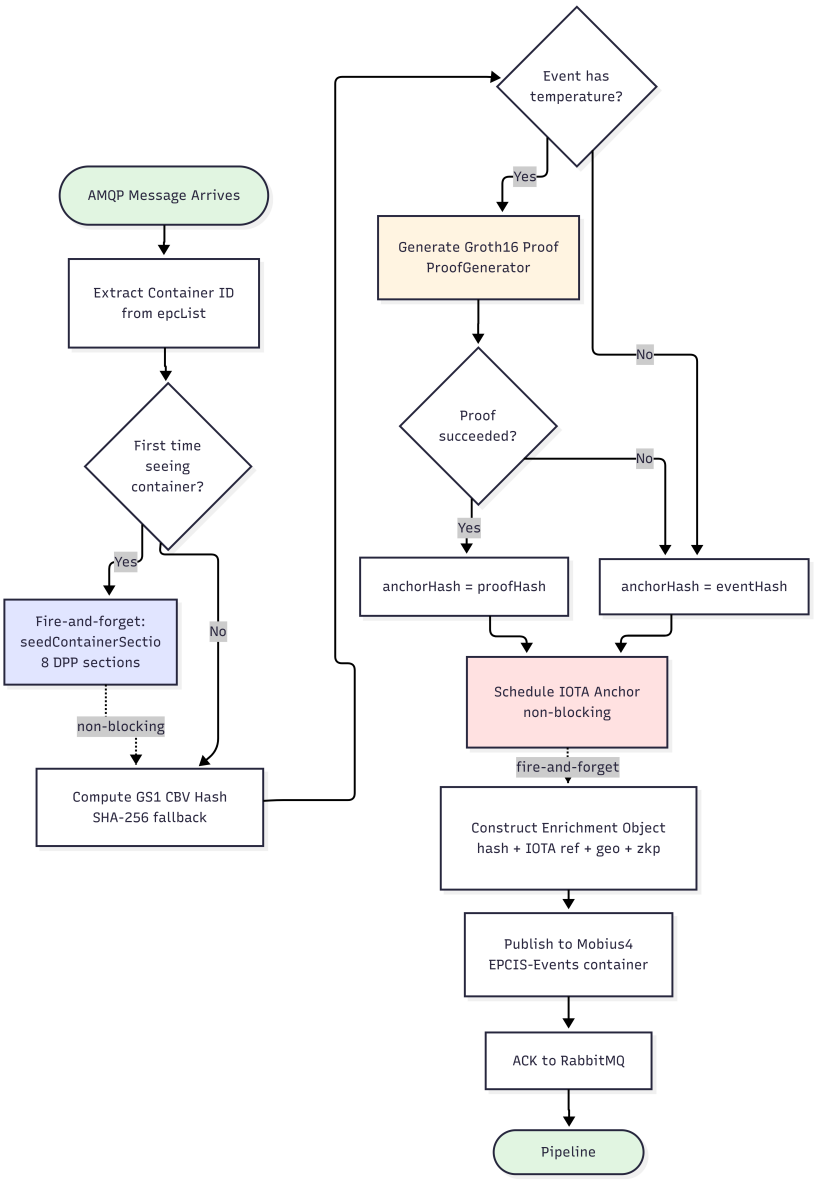

Figure 4.1: EventEnricher pipeline. The seeding step (dashed) is fire-and-forget; IOTA anchoring is scheduled non-blocking. The pipeline ACKs to RabbitMQ only after successful Mobius4 persistence, implementing at-least-once delivery (NFR1).

Two concurrency decisions visible in the diagram are worth explaining. The seeding step is fire-and-forget rather than awaited because it issues multiple sequential oneM2M HTTP calls; blocking would stall the AMQP consumer under burst load. A deduplication guard in the seeder ensures exactly-once container creation despite the non-blocking dispatch. The IOTA anchor is similarly non-blocking: the enrichment pipeline completes and ACKs immediately, while the Merkle batch accumulates and flushes asynchronously (Section 4.5).

DPP Core additionally maintains direct PostgreSQL auxiliary tables for DPP snapshots, webhook subscriptions, and access audit logs, supplementing the primary Mobius4 CIN

persistence path.

4.4 ZKP Module

4.4.1 Circuit Design: `tempBound.circom`

The circuit's purpose, signal interface, and privacy guarantees are established in FR7.1 and the threat model (Section 3.5); the full annotated source is in Appendix B. The implementation uses the `SignedLessThan(16)` component from circomlib, with temperatures scaled by a factor of 10 to integer arithmetic. This 16-bit signed representation covers $\pm 3{,}276°$C, far exceeding the operational range, while supporting the negative values required for frozen-cargo scenarios.

The circuit includes a container-specific secret key as a private input. This key is derived via $K_c = \text{HMAC-SHA256}(K_{anchor}, containerId)$, where K_{anchor} is a platform-wide secret injected through the `ANCHOR_SECRET_KEY` environment variable and never persisted to the database. Because K_c is a 256-bit value that cannot be derived without K_{anchor}, offline brute-force enumeration of the bounded temperature range is infeasible even if the database is compromised. The circuit compiles to 17 non-linear R1CS constraints.

4.4.2 Trusted Setup

The Phase 1 Powers-of-Tau ceremony uses the Hermez Network `pot12` contribution, supporting up to 2^{12} constraints — the rationale for this choice is in Section 3.6. Phase 2 (circuit-specific key generation) is fully automated by the DPP Core container's entrypoint script (`docker-entrypoint.sh`), which implements an idempotent setup sequence: on first launch, it downloads the `pot12` file from public mirrors (with fallback to local generation if all mirrors are unreachable), compiles the circuit, generates the proving key, contributes circuit-specific randomness, and exports the verification key. All artifacts are persisted to a bind-mounted host volume (`/data/zkp`), so subsequent container restarts detect the existing proving key and skip the 2–3 minute ceremony entirely. Symlinks from the application's expected path (`/app/circuits/`) into the persistent volume decouple the trusted setup from application code. The resulting artifacts are a 78 KB WASM witness generator, a 5.2 MB proving key, and a 2.1 KB verification key.

4.4.3 Proof Generation Pipeline

Figure 4.2 shows the `ProofGenerator` decision tree. The module first checks whether ZKP is enabled and whether the event carries a temperature observation; either absence produces a typed skip result so the enricher can fall back to the plain event hash without error. For events that pass both checks, the module validates the reading is within the circuit's

supported range (-40°C to $+85$°C), scales both the measured and threshold temperatures by a factor of 10 to integer arithmetic, derives the container-specific key K_c (derived via HMAC from the platform secret), and calls `snarkjs.groth16.fullProve()` to produce the proof and public signals. It then computes `proofHash` as the SHA-256 of the serialised proof and signals — this hash becomes the Merkle leaf submitted to IOTA (FR7.3) — logs proof generation latency for evaluation (Chapter 5), and returns a structured result containing the proof, public signals, compliance flag, `proofHash`, and metadata. Any error from the WASM prover returns a typed failure object; the enricher falls back to the plain event hash so ZKP errors never cause data loss.

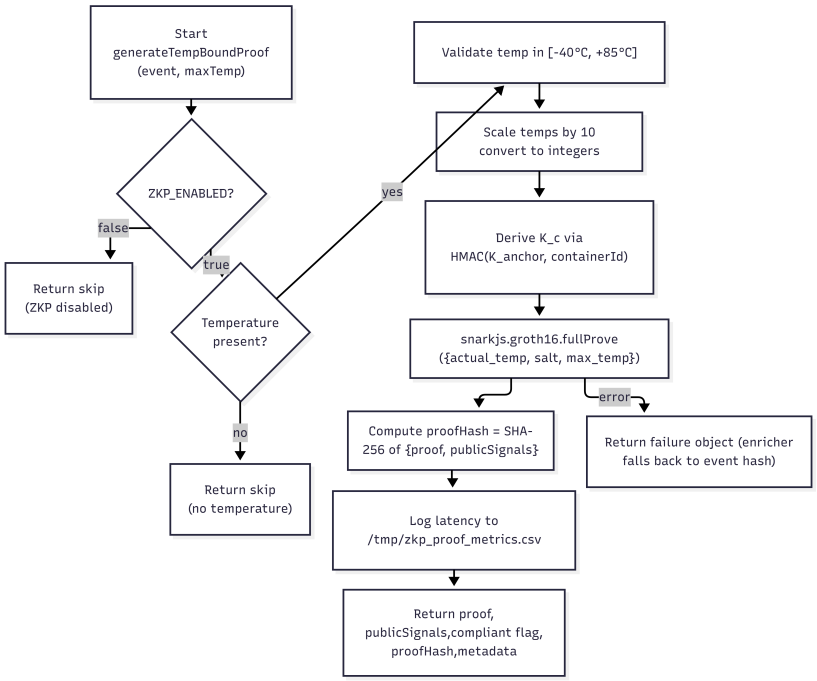

Figure 4.2: Groth16 proof generation pipeline (`ProofGenerator`). Two skip paths (ZKP disabled; no temperature observation) allow the enrichment pipeline to proceed without a proof. Any prover error returns a typed failure object; the enricher falls back to the plain GS1 event hash, satisfying FR7.3 in both success and fallback modes.

Secure Event Hashing. A dedicated utility module (`secureEventHash.js`) manages the HMAC-based key derivation scheme. It reads the platform secret `ANCHOR_SECRET_KEY` from the environment at startup and exposes two functions: `deriveContainerKey`, which computes $K_c = \text{HMAC-SHA256}(K_{anchor}, containerId)$, and `computeSecureEventHash`, which computes $H_{event} = \text{SHA-256}(K_c \parallel \text{Payload})$. The derived K_c is passed transiently to the proof generator as the private circuit witness and is never persisted to the database, returned via any API, or logged in plaintext. The resulting H_{event} serves as the Merkle tree leaf for IOTA anchoring.

4.4.4 Verification Key Pinning

The zero-trust rationale for key pinning is established in Section 3.6. The implementation compiles the verification key into the client bundle at build time via an export script that reads the key from the DPP Core ZKP volume, computes its SHA-256 fingerprint, and generates a static ES module (`lib/trustedVkeys.js`). A pre-deployment validation script detects key drift by comparing the pinned fingerprint against the current key on disk. At runtime, the portal verifies proofs entirely client-side using the pinned key; as an additional integrity check, it fetches the server's key fingerprint and displays a warning banner if it diverges, without blocking verification.

4.5 IOTA Anchoring Service

The design rationale for Locked Notarization and Merkle batching is established in Section 3.6. This section covers the three non-trivial implementation aspects: gas coin management, WASM client lifecycle, and the batch flush sequence.

4.5.1 Gas Coin Pool

IOTA Rebased uses a UTXO-based gas model where each transaction consumes one coin. Naive anchoring would serialise all batch transactions through a single coin, limiting throughput. DPP Core addresses this at startup by splitting available funds into a pool of concurrent transaction lanes, providing sufficient parallelism for the batch anchoring workload.

4.5.2 Client Lifecycle and WASM Cache

A fresh `NotarizationClient` is instantiated per batch transaction rather than reused as a singleton. This design decision warrants explanation because per-use instantiation is typically an anti-pattern for SDK clients. The IOTA Rebased WASM SDK maintains an internal cache of gas coin object references; once a transaction consumes a coin, the cached reference becomes stale. Reusing the same client for a subsequent transaction triggers an unrecoverable `ObjectNotFound` error ("object not available for consumption") from the Rust WASM runtime, which propagates as an uncatchable panic that terminates the Node.js process. No public API exists to invalidate the internal cache without destroying the client. The per-batch instantiation overhead ($\sim$200 ms) is negligible relative to the 30-second batch interval and the 3–4 second IOTA confirmation latency. As an additional safeguard, if a stale coin error is nonetheless detected (e.g., due to a race condition during concurrent batch flushes), the service triggers a controlled process restart to clear the WASM memory entirely.

4.5.3 Merkle Batch Anchoring

The `merkleBatch.js` module accumulates event hashes into an in-memory buffer and flushes when the buffer reaches a configurable batch size (default 100 events, set via `MERKLE_BATCH_SIZE`) or a configurable timeout elapses (default 30 seconds, set via `MERKLE_BATCH_INTERVAL_MS`), whichever comes first. Figure 4.3 illustrates the flush sequence.

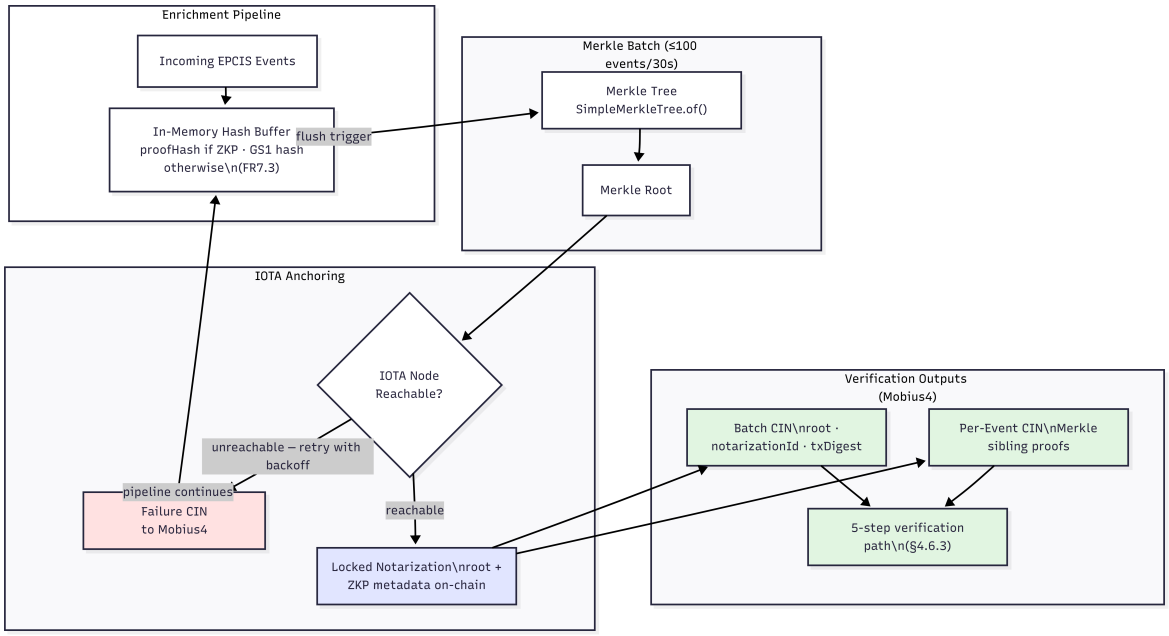

Figure 4.3: Merkle batch anchoring and per-event verifiability pipeline. Each batch anchors a configurable number of event hashes as a single IOTA Locked Notarization; per-event sibling proofs stored in Mobius4 enable independent verification without additional on-chain writes (five-step path, Section 3.6).

Three implementation details are worth noting beyond what the diagram shows. First, a pre-flight HTTP health check runs before the WASM SDK is invoked: a network timeout inside the `@iota/iota-sdk` WASM runtime produces an uncatchable Rust panic that terminates the Node.js process, making the check non-optional rather than a standard defensive measure. Second, the on-chain notarization payload embeds the Merkle root alongside ZKP circuit metadata (`vkeyHash`, `circuitName`, `zkpItemsCount`) when the batch contains ZKP-enabled events, and a `version` field distinguishes ZKP payloads (`zkp-v1`) from plain hash batches (`hash-v1`); this allows future circuit upgrades to be audited against the on-chain record. Third, the `iota-anchors` Mobius4 container that holds batch and per-event CINs is a technical container separate from the eight-section DPP structure (Section 3.7), created on demand and accessed exclusively by the verification subsystem.

At startup, DPP Core performs an IOTA health check and disables anchoring gracefully if the node is unreachable (NFR3), setting a flag checked per event so the enrichment pipeline continues uninterrupted.

4.5.4 IOTA Explorer Integration

The dashboard's `/iota-verify` page lists all anchor CINs from Mobius4 with filtering and search, and provides direct links to the self-hosted IOTA Explorer via transaction digest, allowing operators and auditors to verify on-chain payloads independently of any Ocean DPP API. The public portal similarly provides explorer links for anchored events.
Figure 4.4 shows a verified Merkle batch with its anchor chain for container LMCU2231201.

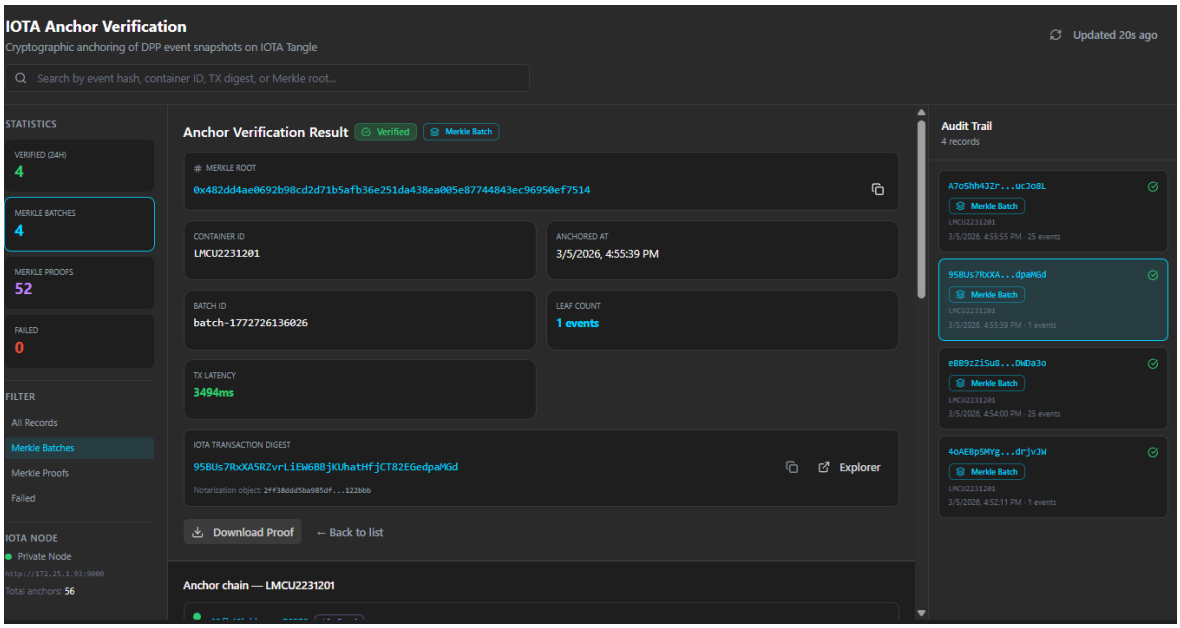

Figure 4.4: IOTA Anchor Verification dashboard showing a verified Merkle batch for container LMCU2231201: 25 events anchored as a single notarization (TX latency 3,328 ms), with per-event sibling proof links visible in the anchor chain below.

4.6 Graceful Degradation Strategy

The platform implements a layered degradation model: each non-critical subsystem can fail independently without halting the event pipeline. Table 4.3 summarises the failure modes and their handling.

The key design principle is that no downstream failure prevents an event from being acknowledged: the AMQP `ack` is sent only after successful Mobius4 persistence, and all other steps (ZKP, IOTA, seeding) are either non-blocking or produce typed failure objects that the enricher handles without rethrowing. This ensures at-least-once delivery end-to-end (NFR1) even when multiple subsystems are degraded simultaneously.

Table 4.3: Graceful Degradation Behaviour per Subsystem

Failing Subsystem	Symptom	Platform Response
ZKP prover	WASM circuit error or missing temperature field	Fall back to plain GS1 event hash; persist event without proof (FR7.3)
IOTA node	Unreachable at startup or during batch flush	Set `global.iotaAvailable = false`; continue enrichment and persistence (NFR3)
Mobius4 CSE	HTTP timeout on CIN write	Throttle translator concurrency; leave AMQP message unacked for redelivery (NFR1)
RabbitMQ	Connection drop mid-processing	Reconnect with exponential back-off; rely on durable queues for buffered messages (NFR2)
Container seeding	oneM2M section creation fails for new container	Fire-and-forget dispatch; deduplication guard retries on next event for same container

4.7 Supporting Services: Messaging, Gateway, and Dashboard

The remaining services complete the platform but their implementation follows directly from the architectural decisions in Section 3.4; the non-trivial points are noted below.

Messaging. Both `translated_ events` and `ws_events` are declared as durable classic queues with a single exclusive consumer each: `dpp-core-enrichment-consumer` and `api-gateway-websocket-consumer` respectively. A dedicated `translated_events.dlq` dead-letter queue receives messages that exhaust their requeue limit, providing a recoverable failure lane separate from the live enrichment path. DPP Core's AMQP consumer uses a prefetch count of 10 with explicit acknowledgement — the `ack` is sent only after successful Mobius4 persistence — ensuring no event is lost if the enricher crashes mid-processing, and bounding the number of in-flight unacknowledged messages under burst load. This directly implements the at-least-once delivery guarantee required by NFR1.

API Gateway and RBAC. The allow-list routes four path groups to internal services: DPP passport endpoints and ZKP verification to DPP Core, external AE event ingestion (requiring `external_ae` role) to DPP Core, and authentication to Ocean-ADM. Two unauthenticated public endpoints expose ZKP proof data and the server-side verification key without JWT validation, supporting the public portal (FR7.4, FR7.5). Rate limiting is applied to all HTTP endpoints, and the WebSocket server broadcasts live events to authenticated dashboard clients

from the dedicated `ws_events` queue.

Dashboard and Public Portal. The authenticated dashboard is a Next.js 14 application with a collapsible sidebar navigation, dark-themed UI (Tailwind CSS), and a global container selector shared across all pages via React context. The dashboard exposes six primary views:

1. **DPP Passport Viewer** (`/dpp`): The central DPP interface. A top bar provides a container dropdown, three view-mode tabs (Sections, Timeline, JSON), and a refresh button. Below, a KPI summary bar displays total events, active alarms, ZKP compliance rate, and IOTA anchor status. An analytics panel renders a temperature line chart (with the compliance threshold marked) and an event-type donut chart (telemetry vs. alarms vs. e-seal). The Sections view renders all eight DPP sections as tabbed cards with structured field display; the Timeline view presents a chronological event list with anchor-status and ZKP-compliance badges; the JSON view provides the raw API response.

2. **GPS Live Map** (`/map`): Full-width interactive map (Leaflet) showing container positions as colour-coded markers: green (active), red (alarm), grey (stale). A right sidebar lists all containers with quick stats. Clicking a marker opens a popup card with the container's last telemetry, linking to the DPP viewer.

3. **Real-Time Event Stream** (`/event-stream`): Live scrolling feed connected via WebSocket to the API Gateway. Events appear as they flow through the pipeline, each card showing container ID, event type, timestamp, sensor readings, and disposition.

4. **IOTA Anchor Audit Trail** (`/iota-verify`): Calls the IOTA verification endpoint for a selected container, fetches all `merkle-proof` CINs, verifies each Merkle path against the on-chain root, and displays per-event verification results with notarization ID, transaction digest, and an IOTA explorer link.

5. **User Administration** (`/admin/users`): Admin-only. Provides a user management table with create, edit, role assignment, password reset, and delete operations, proxied through the API Gateway to Ocean-ADM.

6. **Logistics Partner Portal** (`/logistics-partner`): Available only to the `external_ae` role. Provides an EPCIS 2.0 event submission form and a container-scoped timeline showing the partner's submitted events with anchor and ZKP status.

Role-filtered content is enforced at the API Gateway boundary before data reaches the front end. Figures 4.5–4.7 show the authenticated dashboard; Figures 4.8 and 4.9 show the public ZKP verification portal.

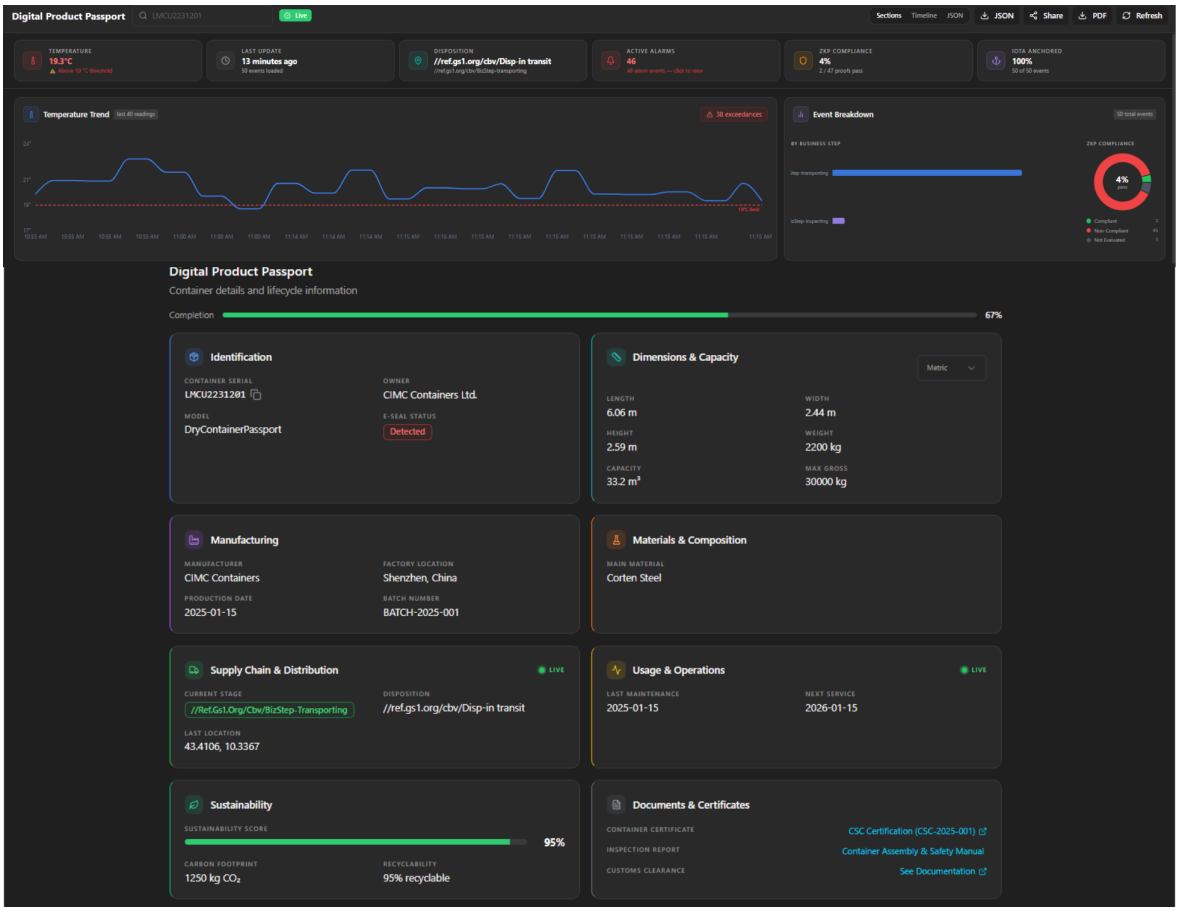

Figure 4.5: DPP Passport Viewer showing the eight-section passport structure for container LMCU2231201. The KPI summary bar (top) displays total events, active alarms, ZKP compliance rate, and IOTA anchor status. The temperature chart shows the compliance threshold at 19 °C. Below, the Sections tab renders each DPP section as a structured card.

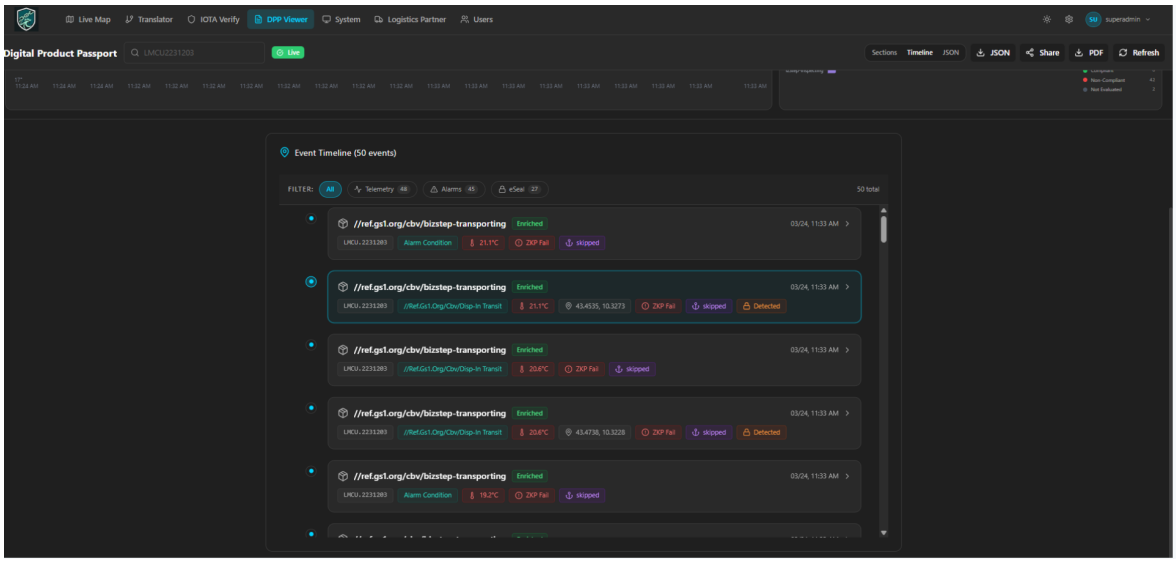

Figure 4.6: DPP Passport Viewer in Timeline mode. Each event row shows the event time, business step, disposition, IOTA anchor status badge (pending/anchored), and ZKP compliance badge (compliant/violated/not evaluated). Filter buttons allow isolation of telemetry, alarm, and e-seal events.

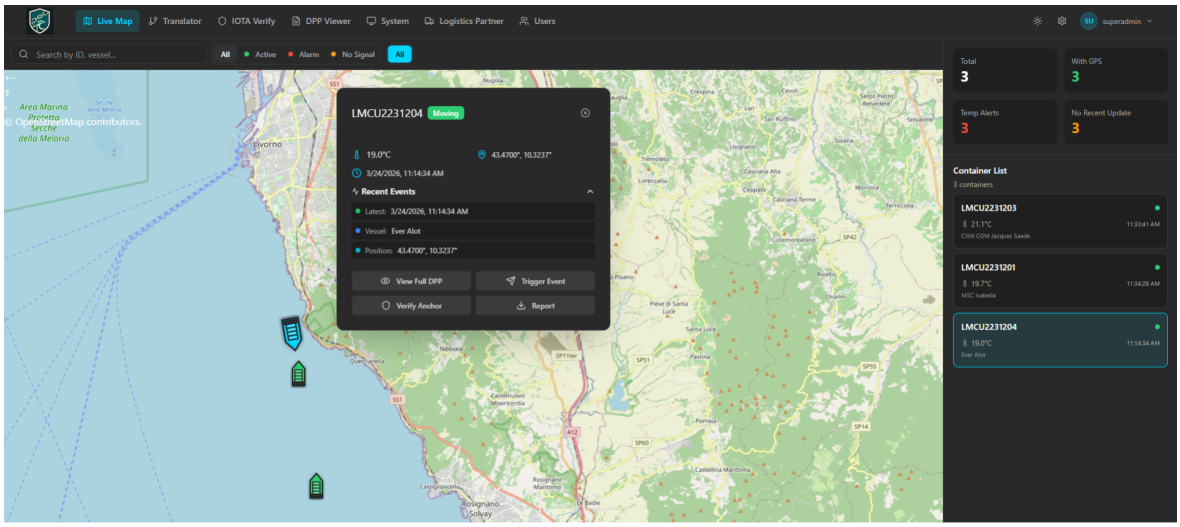

Figure 4.7: GPS Live Map showing container positions as colour-coded markers. Green markers indicate active containers with recent telemetry; red markers indicate containers in alarm state (temperature exceeding the compliance threshold). The right sidebar lists all tracked containers with quick-glance status indicators.

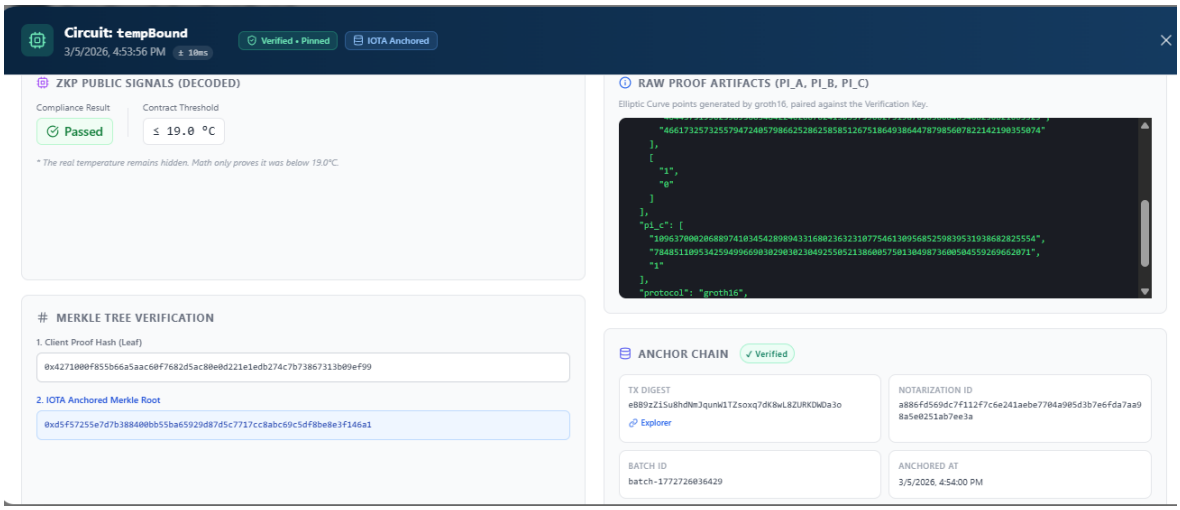

Figure 4.8: Compliant event: the Groth16 proof verifies client-side, the IOTA immutability check passes, and the temperature reading satisfies the contract threshold — all without revealing the raw sensor value.

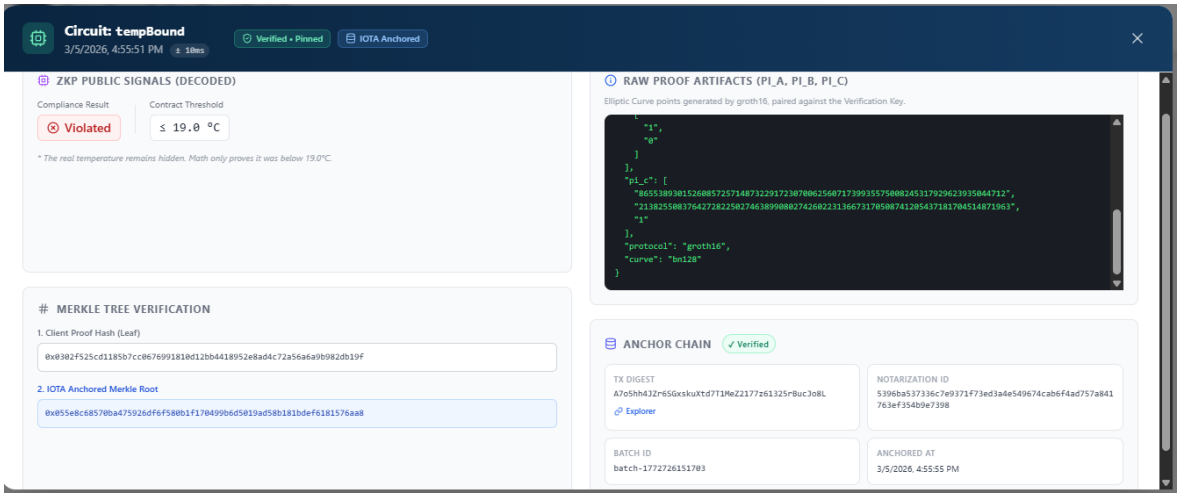

Figure 4.9: Non-compliant event: the Groth16 proof and IOTA immutability checks succeed, but the temperature exceeds the contract threshold, yielding a failed compliance verdict while the actual reading remains hidden.

4.8 IoT Emulator

The evaluation workloads (Chapter 5) are generated by a Python-based IoT emulator that simulates multiple container voyages concurrently. The emulator's voyage model, telemetry structure, and container tracking scenario are derived from the 5G Global Tracking System maritime platform described by Noto et al. [42], which demonstrated end-to-end NB-IoT container tracking using a oneM2M platform with commercial container tracking devices [43]. The emulator is configured via YAML profiles with JSON Schema validation and supports four transport protocols (MQTT, HTTP, CoAP, and UDP). For all evaluation experiments, the MQTT transport is used with QoS-1 delivery to match the production ingestion path.

Each simulated container follows a deterministic voyage model seeded by the container identifier: temperature oscillates sinusoidally around a configurable baseline (-18°C for frozen cargo, $+4$°C for chilled), with Gaussian noise ($\sigma = 0.3$°C) added per reading to simulate sensor jitter. Humidity, pressure, GPS coordinates, and vessel speed are generated from similar parametric models. A configurable `container_count` parameter (default 1, up to 1,000) launches concurrent coroutines, each emitting telemetry at a configurable interval (default 1 s), enabling the burst and scaling experiments without modifying the emulator's core data model.

The emulator generates all three input types defined in Section 3.7.1. *Telemetry events* are emitted periodically on every tick of the configurable interval. *Alarm events* are generated asynchronously by an alarm engine that monitors each sensor against configurable thresholds (e.g., temperature: $-20/+50$ °C, humidity: 0/100%, accelerometer: 5,000 mg); an alarm fires on threshold crossing (onset) and clears when the sensor returns within range (recovery). *E-seal events* are generated by a three-state seal machine (CLOSED $\rightarrow$ TAMPERED $\rightarrow$ OPEN) driven by the gyroscope sensor: sustained motion beyond a configurable duration triggers state transitions, and motion cessation restores the CLOSED state. All three message types are published to the same MQTT topic and carry the container's ISO 6346 identifier, enabling the Translator to route them to the appropriate EPCIS translation function.

4.9 Testing and Code Quality

The DPP Core test suite uses Jest 29 with `-runInBand` (serial execution to avoid port conflicts on shared Mobius4 and RabbitMQ instances). The suite comprises 12 test files organised into five phases with approximately 98 individual test cases:

1. **Phase 1 — Pipeline:** GS1 CBV 2.0 event hashing, NI URI generation, and enrichment pipeline flow (7 cases).

2. **Phase 2 — Storage:** Mobius4 CIN persistence via the oneM2M HTTP API, including container creation, CIN retrieval, and error responses (17 cases).

3. **Phase 3 — API:** REST endpoint routing, response formatting, and DPP passport builder output structure (30 cases).

4. **Phase 4 — Catalog:** Container catalog seeding, eight-section structure creation, and idempotent re-seeding (24 cases).

5. **Phase 5 — ZKP Integration:** End-to-end proof generation (compliant, non-compliant, and missing-temperature cases), proof verification (valid, invalid, and tampered inputs), and anchor hash selection logic (10 cases).

A separate integration test (`integration-real-iota.test.js`) exercises the complete Merkle anchoring path against a live IOTA localnet instance, verifying that the on-chain Merkle root matches the locally computed root and that per-event sibling proofs validate correctly. This test is excluded from the default Jest run because it requires a running IOTA node.

The test suite does not include end-to-end browser tests for the dashboard or public portal; front-end correctness is validated manually during development and indirectly through the evaluation experiments (Chapter 5), which exercise the full pipeline from emulator to dashboard display.

A separate API Gateway test suite (4 files, 15 cases) covers JWT verification middleware, role-based route mapping, container listing endpoints, and Mobius4 client error handling. Combined with DPP Core, the platform maintains approximately 113 automated test cases across two services.

No formal code coverage reporting is configured; however, manual inspection indicates that the critical processing paths — event hashing, enrichment, ZKP proof/verify, Merkle anchoring, and AMQP acknowledgement — are exercised by the test suite. Continuous integration is not deployed for this prototype; tests are run locally before each deployment via `npm test`.

Chapter 5

Experimental Evaluation

5.1 Evaluation Methodology

This chapter presents the 16 experiments conducted to answer RQ1–RQ4. Each experiment subsection follows a uniform structure: *Objective* states what is measured and why it answers its RQ; *Method* describes the procedure precisely enough to replicate; *Results* presents quantitative data; *Finding* summarises the conclusion in 1–2 sentences and states the NFR verdict. Section 5.8 maps every experiment to its RQ in a single table.

Test environment. All 16 controlled experiments ran on a Dell Latitude 5401 (Intel Core i7-9850H, 6 cores / 12 threads, 16 GB RAM, SSD) running Windows 11 Pro with Hyper-V. The full Docker Compose stack (up to 9 containers; 8 in default configuration, 9 with the Nginx reverse proxy for scaling experiments) shared this single host. This environment is significantly more constrained than a production deployment on dedicated Linux servers; Mobius4 and its database compete for CPU, RAM, and I/O with the rest of the stack, which limits the maximum sustainable HTTP request rate and should be considered when interpreting absolute throughput numbers. The same container stack was later deployed on an internet-reachable virtual machine for a CTD field test (Section 4.1.1), but that deployment was not used for the controlled benchmarks presented in this chapter.

Emulator. The IoT emulator service generated synthetic ocean container telemetry (LMCU-series container IDs). Temperature readings oscillate between 14.5–21.5 °C, bisected by the 19 °C compliance threshold to produce a balanced mix of compliant and non-compliant events for statistical evaluation. The 19 °C threshold was selected for evaluation convenience; the threshold is a configurable deployment parameter, and pharmaceutical cold-chain deployments would set it to 2–8 °C. The ZKP circuit and pipeline behaviour are independent of the threshold value.

Metrics collection. Latency timestamps are recorded to CSV files under `evaluation results/`. Unless stated otherwise, ZKP processing is disabled. All percentile values (P95, P99) in this chapter use the *nearest rank* method with ceiling index ($\lceil p \cdot n \rceil$).

Data cleaning. Where stated, WASM cold-start events (identified by `enrichMs` $>$ 200 ms in the first 2–3 events of a run) and negative-latency artefacts (caused by Docker NTP clock corrections) are excluded. The raw sample size and clean sample size are reported separately.

Table 5.1 records the ZKP-specific configuration used in ZKP experiments.

Table 5.1: ZKP Experimental Configuration

Parameter	Value
Circom Version	2.1.9
snarkjs Version	0.7.6
Proving System	Groth16
Circuit	tempBound (17 R1CS constraints)
Powers of Tau	Hermez Phase 1 (2^{12} constraints)
Temperature Scale Factor	10 (fixed-point)
Compliance Threshold	19 °C (configurable per deployment)

5.2 Performance Experiments

5.2.1 E1: Baseline End-to-End Latency (ZKP Disabled)

5.2.1.1 Objective

Establish the baseline E2E latency of the Ocean DPP pipeline with ZKP disabled, isolating the core translation and storage path. Answers **RQ1**.

5.2.1.2 Method

The emulator generated a continuous stream of telemetry for container LMCU2231201. The Translator converted each event to EPCIS 2.0 format and published to RabbitMQ; DPP Core enriched and stored to Mobius4. Latency measured from `eventTimeISO` (event creation) to `publishedAt` (Mobius4 CIN write confirmation). Raw sample: 1,000 events; after excluding 3 cold-start spikes (`enrichMs` $\approx$ 900 ms, caused by first TCP connection to Mobius4) and 1 negative-latency NTP artefact: $n = 996$. Run date: 2026-02-28.

5.2.1.3 Results

Table 5.2: E1: Baseline E2E Latency Statistics ($n = 996$, ZKP disabled)

Metric	Mean	Std Dev	Median	P95	P99	Max
E2E Latency (ms)	32.4	9.8	30	48	72	116
Enrichment (ms)	0.7	—	—	2	—	7

Figure 5.1 visualises the latency distribution as a CDF and histogram, confirming the tight concentration around the 30 ms median with a right-skewed tail.

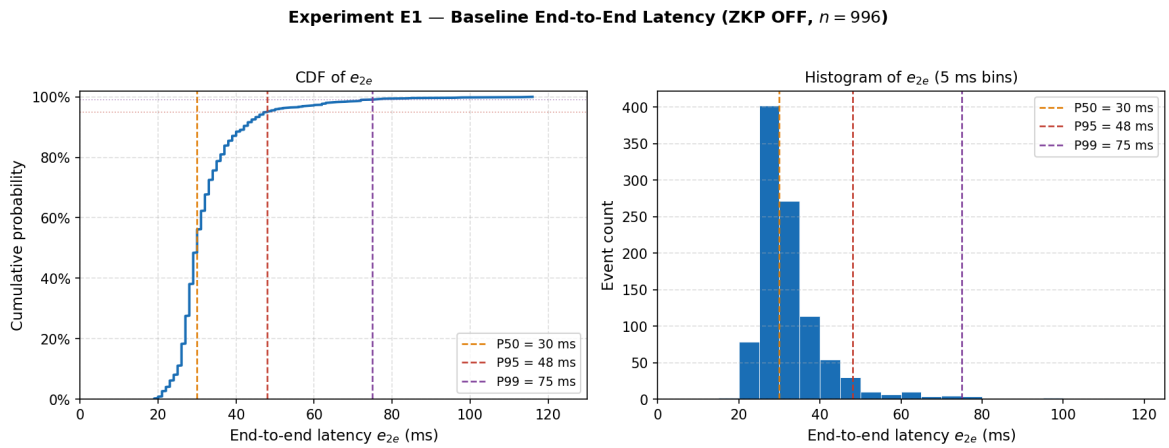

Figure 5.1: E1: Baseline latency distribution ($n = 996$, ZKP disabled). Left: CDF with P50, P95, and P99 percentile markers. Right: histogram (5 ms bins) showing the right-skewed distribution with 95% of events completing within 48 ms.

5.2.1.4 Finding

Median latency of 30 ms is well below the 200 ms NFR4 target at median load. P95 = 48 ms and P99 = 72 ms confirm that the pipeline operates with substantial headroom under steady-state conditions. **NFR4:** ✓

5.2.2 E2: Throughput and Saturation

5.2.2.1 Objective

Determine maximum sustainable event throughput and identify the primary bottleneck under increasing load. Answers **RQ1**.

5.2.2.2 Method

The experiment comprised two phases. In Phase 1 (sustained throughput), three consecutive 120-second rounds drove increasing emulator event rates through the full pipeline with ZKP enabled. In Phase 2 (burst capacity), two short-duration tests verified that the system absorbs instantaneous bursts of 50 and 100 containers without permanent data loss, even though it cannot sustain those rates continuously. The distinction matters: Phase 1 measures steady-state throughput; Phase 2 measures transient burst tolerance.

5.2.2.3 Results: Phase 1 — Sustained Throughput

Sustainable throughput on this hardware is 7 ev/s (R2). At R3 (16.7 ev/s), median la- tency rises to 112 ms and P95 to 191 ms as the RabbitMQ queue grows faster than DPP Core can drain it. Figure 5.2 illustrates this latency trend across the three rounds, with the NFR4 200 ms target shown for reference. The primary bottleneck is Mobius4 HTTP write latency (

Table 5.3: E2 Phase 1: Sustained Throughput Profile (3 rounds, 120 s each)

Round	Containers	Input (ev/s)	n	P50 (ms)	P95 (ms)	NFR4
R1	1	1.7	202	25	31	✓
R2	4	6.7	799	54	98	✓
R3	10	16.7	1,993	112	191	×

80–120 ms per CIN creation on the shared Docker host): DPP Core and the Translator are not the limiting com- ponents. In commercial deployments, container IoT devices typically report in the order of minutes (e.g., every 15minutes). At a sustained 7events/s, the prototype comfortably scales to hundreds of containers at such reporting frequencies.

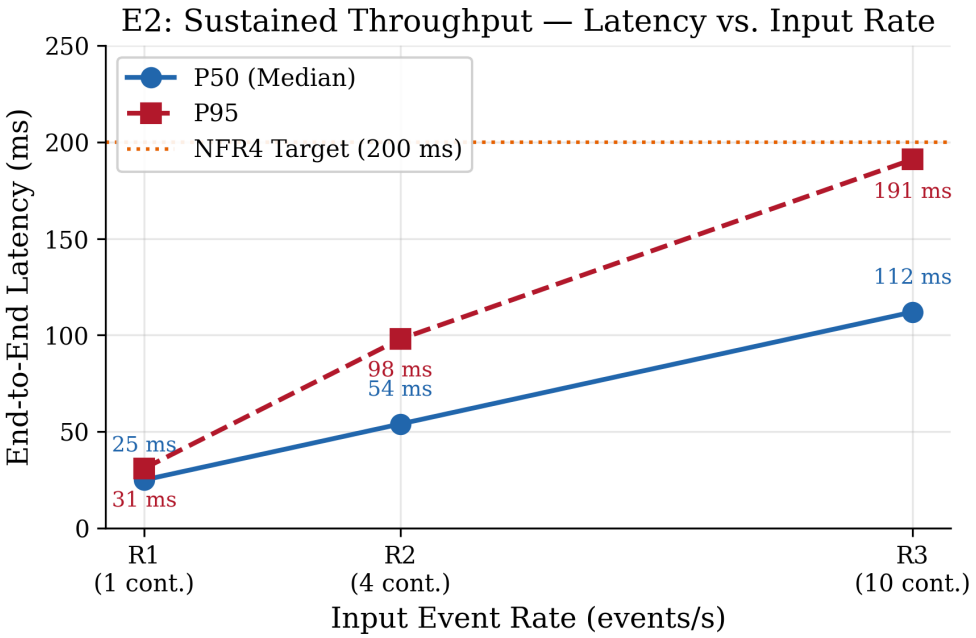

Figure 5.2: E2 Phase 1: End-to-end latency (P50 and P95) vs. input event rate across three sustained-throughput rounds. The dashed orange line marks the NFR4 target (200 ms).

5.2.2.4 Results: Phase 2 — Burst Capacity

After identifying the Mobius4 bottleneck in Phase 1, two additional tests verified the system's behaviour under instantaneous burst loads that exceed the sustained throughput ceiling. These tests ran for shorter durations because the emulator was stopped once the burst was delivered; the purpose is not to measure steady-state latency but to confirm that no events are permanently lost under transient overload.

Figure 5.3 compares the median E2E latency and success rate for both burst tests.

Table 5.4: E2 Phase 2: Burst Capacity Tests

Test	Containers	Sent	Enriched	Success	Median E2E	P95 E2E	Loss
E2-R4	50	242	242	100%	7,627 ms	7,660 ms	0
E2-R5	100	819[a]	500	61%[b]	16,661 ms	16,911 ms	0

[a]Of 819 emulated messages, 500 reached DPP Core; 319 failed at the Translator–Mobius4 boundary due to HTTP connection saturation.
[b]All events that passed through the Translator were processed by DPP Core. Shed messages remain buffered in MQTT QoS-1 persistent sessions for redelivery; no events are permanently lost.
Note: E2-R4 and E2-R5 are labelled with the "E2-" prefix to distinguish them from reliability experiment R4 (Section 5.7.4).

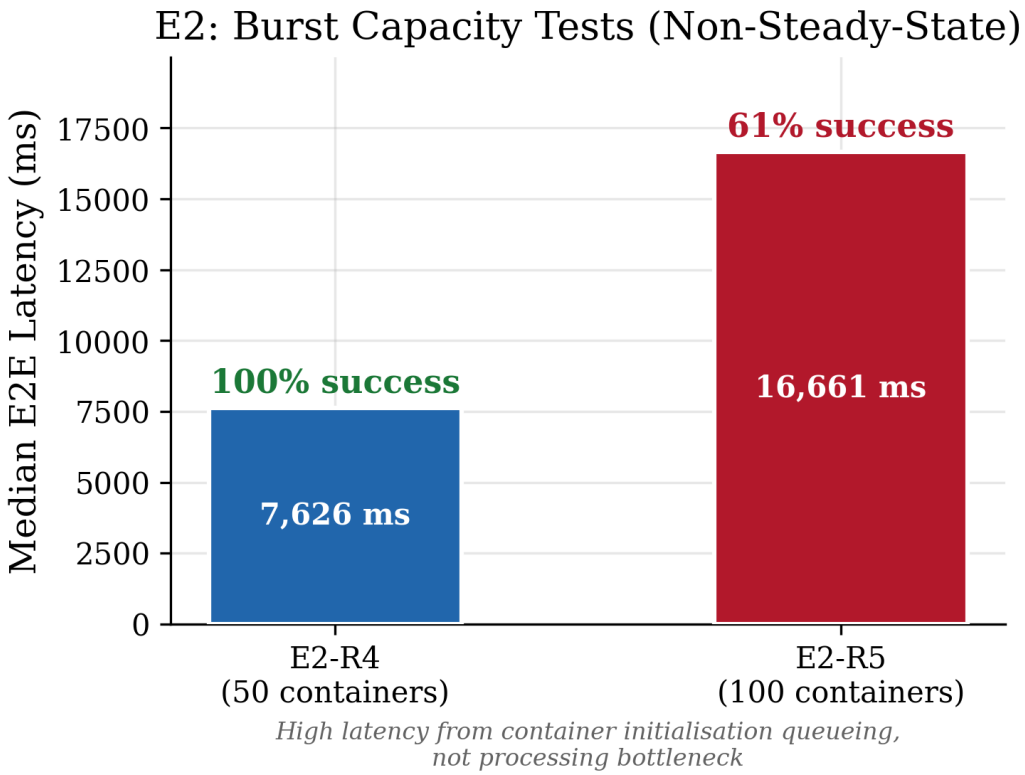

Figure 5.3: E2 Phase 2: Burst capacity tests. Bar height shows median E2E latency; labels indicate success rate. High latency reflects container initialisation queueing, not a processing bottleneck.

5.2.2.5 Identified Limitation: Mobius4 HTTP Saturation

The Phase 2 results expose a concrete architectural limitation. At 50 containers (≈ 84 ev/s instantaneous burst), the single Mobius4 instance handles all events (E2-R4: 100% success). At 100 containers (≈ 168 ev/s instantaneous burst), approximately 39% of events fail at the Translator–Mobius4 HTTP boundary before reaching the enrichment pipeline (E2-R5: 319 connection errors). The failures are HTTP-level (connection timeouts and 503 Service Unavailable responses), not application-level: Mobius4's Express.js HTTP server saturates under the concurrent POST load on the shared Docker host.

This limitation is specific to the single-host evaluation environment and does not represent a fundamental architectural constraint. The platform's stateless Mobius4 instances can be horizontally scaled behind a load balancer, as demonstrated in Experiment S3 (Section 5.4.2), which eliminates all connection errors by adding a second Mobius4 replica.

5.2.2.6 Finding

Sustained throughput is ≈ 7 ev/s on the evaluation hardware, limited by Mobius4 HTTP write latency. The system absorbs burst loads of 50 containers with zero loss (E2-R4: 100% success, avg. 6.6 s E2E latency due to container initialisation overhead) and 100 containers with partial Translator-side shedding (E2-R5: 61% success, buffered by MQTT QoS-1 for redelivery). The Mobius4 bottleneck is resolved by horizontal scaling (S3). **Throughput confirmed at production-relevant rate:** ✓

5.2.3 E10: End-to-End Latency with ZKP Overhead

5.2.3.1 Objective

Quantify the latency overhead of Groth16 proof generation on the end-to-end pipeline. Answers **RQ2**.

5.2.3.2 Method

Identical to E2 Round 1 with `ZKP_ENABLED=true`. Latency measured identically to E1. $n = 199$ **events**, container LMCU2231201.

5.2.3.3 Results

Table 5.5: E10: ZKP Overhead on E2E Latency (E1 vs. E10)

Metric	E1 (no ZKP)	E10 (with ZKP)	Overhead
Mean E2E	32.4 ms	349.9 ms	+317.5 ms
Median E2E	30 ms	308 ms	+278 ms
P95 E2E	**48 ms**	**520 ms**	**+472 ms**
P99 E2E	72 ms	1,968 ms	+1,896 ms
Enrichment mean	0.7 ms	304.7 ms	+304.0 ms
Enrichment P95	2 ms	457 ms	+455 ms

Figure 5.4 visualises the E1 vs. E10 comparison on a logarithmic scale, highlighting the overhead at each percentile.

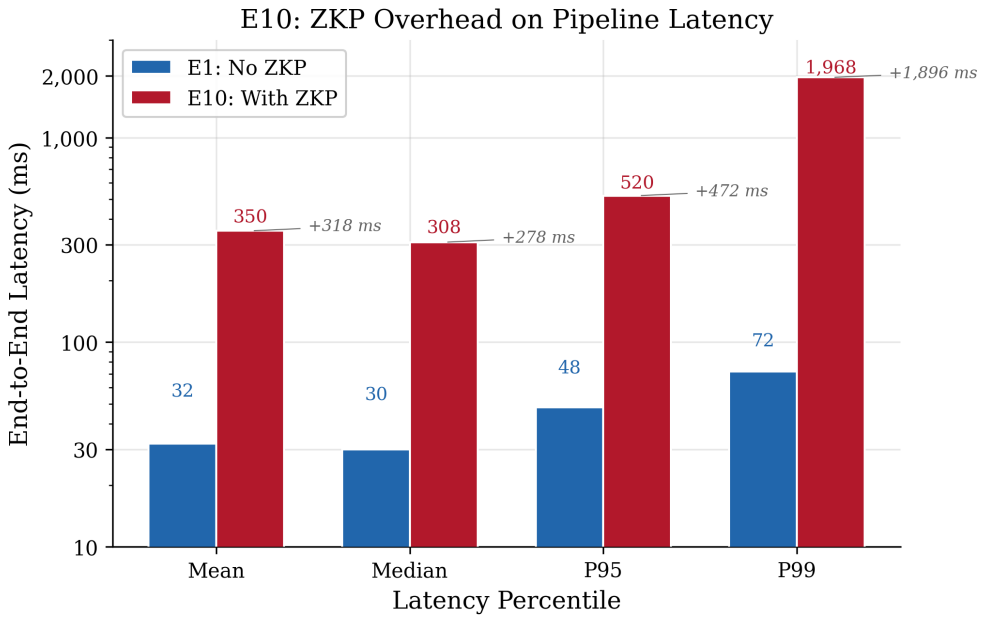

Figure 5.4: E10: ZKP overhead on pipeline latency. Grouped bars compare E1 (no ZKP) with E10 (with ZKP) across mean, median, P95, and P99. Log scale; overhead annotations between bar pairs.

5.2.3.4 Finding

The overhead originates almost entirely from proof generation (304.7 ms enrichment delta). Median E2E of 308 ms remains operationally acceptable for maritime IoT intervals (events arriving every 10–30 s). P99 of 1,968 ms is dominated by two WASM cold-start proofs; steady-state P95 (520 ms) is within the sub-second range expected for ZKP-enabled processing. **ZKP overhead acceptable for maritime intervals:** ✓

5.3 ZKP Experiments

5.3.1 E5: ZKP Proof Generation Performance

5.3.1.1 Objective

Quantify the computational cost of Groth16 proof generation and characterise its statistical distribution. Answers **RQ2**.

5.3.1.2 Method

198 temperature compliance proofs generated for container LMCU2231201, comprising both compliant ($\leq$19 °C, 89 events, 44.9%) and non-compliant ($>$19 °C, 109 events, 55.1%) outcomes. For each proof, `proofGenMs` was recorded. SHA-256 hash time over the equivalent EPCIS event JSON was measured as a baseline. Zero proof generation failures

occurred across all 198 executions.

5.3.1.3 Results

Table 5.6: E5: ZKP Proof Generation Statistics ($n = 198$)

Metric	Mean	Median	P95	P99	Max
Proof Generation (ms)	303.6	266	461	1,904	2,006
SHA-256 Baseline (ms)	0.4	0.3	0.8	1.2	—

The P99 value (1,904 ms) is dominated by two WASM cold-start events at the beginning of the run (first proof: 1,904 ms, second: 2,006 ms). These correspond to the initial WASM module load, JIT compilation, and cache warmup. After warmup, proofs stay in the 130–465 ms range; excluding these two outliers, 97.5% of all proofs complete in under 500 ms. Figure 5.5 presents the latency distribution for all three ZKP-related operations (proof generation, browser verification, and SHA-256 baseline) on a shared logarithmic scale.

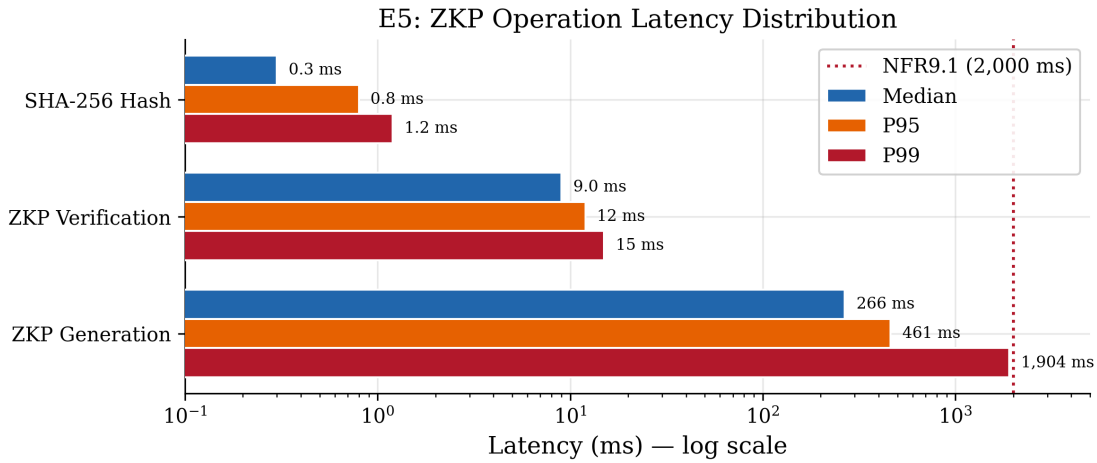

Figure 5.5: E5/EV1: ZKP operation latency distribution. Horizontal bars show median, P95, and P99 for proof generation, browser verification, and SHA-256 hashing. The dashed line marks the NFR9.1 ceiling (2,000 ms). Log scale.

5.3.1.4 Finding

Proof generation averages 303.6 ms with the bulk of proofs clustered around the 266 ms median. All proofs, including cold-start outliers, complete within the 2 s NFR9.1 ceiling. **NFR9.1 (proof generation < 2 s): ✓**

5.3.2 EV1: ZKP Verification Latency (Browser WASM)

5.3.2.1 Objective

Measure the latency of browser-side Groth16 proof verification via the public portal. Answers **RQ2** (interactive verification feasibility).

5.3.2.2 Method

195 stored proofs for container LMCU2231201 fetched via `GET /zkp/public/:containerId`. Each proof verified client-side using `snarkjs.groth16.verify(pinnedVkey, publicSignals, proof)` in WebAssembly; latency recorded using `performance.now()`.

5.3.2.3 Results

Table 5.7: EV1: ZKP Verification Latency (Browser WebAssembly, $n = 195$)

Metric	Mean	Median	P95	Min/Max
Verification Time	9.8 ms	9 ms	12 ms	8 ms / 18 ms
Verification Failures	0 (100% success rate)			
Gen:Verify Ratio	31:1 (303.6 ms / 9.8 ms)			

5.3.2.4 Finding

Sub-10 ms mean verification with a tight distribution (8–18 ms) confirms that Groth16 verification is constant-time with respect to the private input. The 31:1 generation-to-verification ratio means the privacy layer becomes cost-effective whenever at least two independent parties need to verify the same event. **NFR9.2 (verification $<$ 20 ms): ✓**

5.3.3 EI4: Tamper Evidence Detection

5.3.3.1 Objective

Validate that the Merkle anchoring layer detects proof tampering. Answers **RQ2** and **RQ3**.

5.3.3.2 Method

Ten anchored EPCIS events for container LMCU2231201 were subjected to tamper simulation. For each event, the `eventTime` field was shifted by $+1$ second to produce a modified copy. The original and tampered event hashes were computed using the GS1 CBV 2.0 canonical hashing algorithm. The tampered hash was checked against the stored Merkle inclusion proof and the IOTA-anchored root. Detection latency was measured from hash recomputation to inclusion-proof rejection. Unmodified events were verified in parallel to confirm zero false positives.

5.3.3.3 Results

Table 5.8: EI4: Tamper Detection Results ($n = 10$)

Metric	Value
Tampered events tested	10
Baseline verification (unmodified)	10/10 pass
Tamper detection rate	10/10 (100%)
IOTA notarization verified	10/10
Mean detection latency	0.002 ms
Max detection latency	0.005 ms
False positives	0

All ten tampered events produced different leaf hashes from the originals, causing immediate Merkle inclusion failure. All ten unmodified events passed verification, confirming zero false positives.

5.3.3.4 Finding

Any single-field modification to a stored event is detected in sub-microsecond time by Merkle inclusion verification against the IOTA-anchored root. The 100% detection rate reflects the deterministic nature of hash-based tamper detection (any input change produces a different SHA-256 digest with overwhelming probability); the small sample size ($n = 10$) limits the statistical power of this result but does not affect the underlying cryptographic guarantee. **Tamper evidence:** ✓

5.4 Scalability Experiments

5.4.1 S2: Horizontal DPP Core Scaling

5.4.1.1 Objective

Determine whether adding a second DPP Core instance reduces E2E latency, validating RabbitMQ round-robin work queue scalability. Answers **RQ1**.

5.4.1.2 Method

Identical emulator workload (10 containers, 120 s, E2 Round 3 load pattern) in two configurations: one and two DPP Core instances. RabbitMQ distributes messages to competing consumers via round-robin prefetch. One-instance run: $n = 2{,}060$; two-instance combined: $n = 2{,}063$ (replica 1: 1,035, replica 2: 1,028 — near-perfect 50/50 distribution).

Table 5.9: S2: Horizontal scaling latency comparison

Configuration	n	**Mean (ms)**	**Median (ms)**	**P95 (ms)**	**Throughput (ev/s)**
1 DPP Core instance	2,060	747	731	907	14.2
2 DPP Core instances	2,063	500	460	653	17.4
Improvement	—	-33.1%	-37.1%	-28.0%	$+21.9\%$

Mann–Whitney U test: $U = 4{,}035{,}154$, $p < 0.001$ (one-sided, $\alpha = 0.05$). The 95% confidence intervals for the two configurations do not overlap, providing independent confirmation of the effect.

5.4.1.3 Finding

Adding a second DPP Core instance reduces median latency by 37.1% ($p < 0.001$) and increases throughput by 21.9%, demonstrating near-linear horizontal scalability for the enrichment tier. **Horizontal scaling confirmed:** ✓

5.4.2 S3: Horizontal Mobius4 Scaling

5.4.2.1 Objective

Determine whether horizontal Mobius4 scaling resolves the HTTP saturation bottleneck identified in E2 Phase 2 (Section 5.2.2.5). Answers **RQ1**.

5.4.2.2 Method

The emulator replayed the E2 high-load pattern (50 containers, 120 s) against two configurations: (1) a single Mobius4 instance and (2) two Mobius4 instances behind an Nginx round-robin proxy (`docker compose -scale mobius4=2`). The single-instance run produced 1,090 successful events; the remainder (≈810) failed with HTTP connection errors due to Mobius4 saturation — consistent with the limitation identified in E2 Phase 2 (Section 5.2.2.5). The two-instance run produced 1,898 successful events with zero connection errors.

5.4.2.3 Results

5.4.2.4 Finding

These results indicate that on this evaluation hardware, the dominant bottleneck at 50-container load is Mobius4's HTTP handling, not the PostgreSQL backend; scaling Mobius4 improves throughput and latency while the shared database remains sufficient at this scale. Scaling to $2\times$ eliminates all HTTP connection errors identified in E2 Phase 2, increases successful event

Table 5.10: S3: Mobius4 Horizontal Scaling (1× vs. 2× instances)

Configuration	**n**	**Mean (ms)**	**Median (ms)**	**P95 (ms)**
1× Mobius4	1,090	3,621	1,148	17,904
2× Mobius4	1,898	1,304	801	5,426
Improvement	+74% events	−64%	−30%	−70%

Table 5.11: S3: Connection Error Elimination

Metric	**1× Mobius4**	**2× Mobius4**
Successful events	1,090	1,898
Connection errors	≈810	0
Error rate	≈43%	0%

The 2× configuration processed slightly more events than emitted (1,898 vs. ≈1,851) due to MQTT QoS-1 retransmissions during transient delays; all original events were delivered.

throughput by 74%, and reduces P95 latency by 70%. Together with S2 (DPP Core scaling), this confirms that both pipeline tiers scale horizontally via standard container orchestration.

Mobius4 horizontal scaling confirmed: ✓

Note on 1× *statistics.* The 1× mean (3,621 ms) is substantially higher than its median (1,148 ms) due to heavy right skew caused by error-retry latency on failed HTTP connections. The median is the more representative measure of typical event latency for successful writes; the mean reflects the tail impact of connection-error retries that inflate individual event times.

Figure 5.6 presents both scaling experiments side by side, showing the latency improvements at each percentile for DPP Core (S2) and Mobius4 (S3).

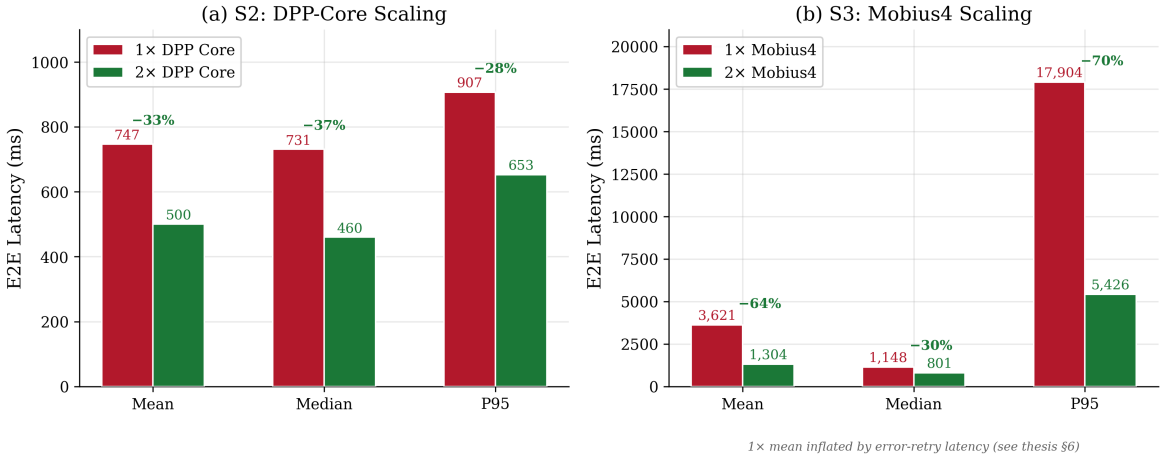

Figure 5.6: Horizontal scaling results. (a) S2: DPP-Core 1× vs. 2× instances. (b) S3: Mobius4 1× vs. 2× instances. Percentage labels show the improvement from scaling. Note the different y-axis scales.

5.5 IOTA Anchoring Experiments

5.5.1 EI1: Anchor Latency vs. Batch Size

5.5.1.1 Objective

Characterise the cost–latency trade-off for Merkle batch anchoring across three batch sizes. Answers **RQ3**.

5.5.1.2 Method

Three runs submitted batches of 10, 50, and 100 event hashes. Timer-based flushing was disabled (`MERKLE_BATCH_INTERVAL_MS=999999999`) to isolate pure size-based triggering. Each batch assembles a Merkle tree, computes the root, and submits one IOTA Locked Notarization transaction. Latency recorded from batch-trigger to notarization confirmation. Run date: 2026-03-02.

5.5.1.3 Results

Table 5.12: EI1: Blockchain Anchor Latency by Batch Size

Batch Size	n	Mean (ms)	Median (ms)	P95 (ms)	Min / Max
10 events/batch	27	4,036	3,648	6,583	3,360 / 10,376
50 events/batch	21	3,529	3,562	3,675	3,304 / 3,696
100 events/batch	18	4,183	3,914	7,608	3,689 / 7,608

5.5.1.4 Finding

Anchor latency is dominated by IOTA network confirmation time (3.5–4.2 s mean across all batch sizes), not Merkle tree computation. Batch-50 achieves the tightest variance ($\pm$49 ms CI), confirming the most consistent IOTA node behaviour at that batch size. Figure 5.7 visualises the mean anchor latency with min/max error bars. Since anchoring runs as a non-blocking background process, this latency does not affect event pipeline throughput. **IOTA anchoring confirmed as non-blocking:** ✓

5.5.2 EI2–EI3: Anchoring Throughput and Cost Reduction

5.5.2.1 Objective

Demonstrate that IOTA anchoring does not become a throughput bottleneck (EI2), and quantify the per-event transaction cost reduction achieved by Merkle batching (EI3). Both

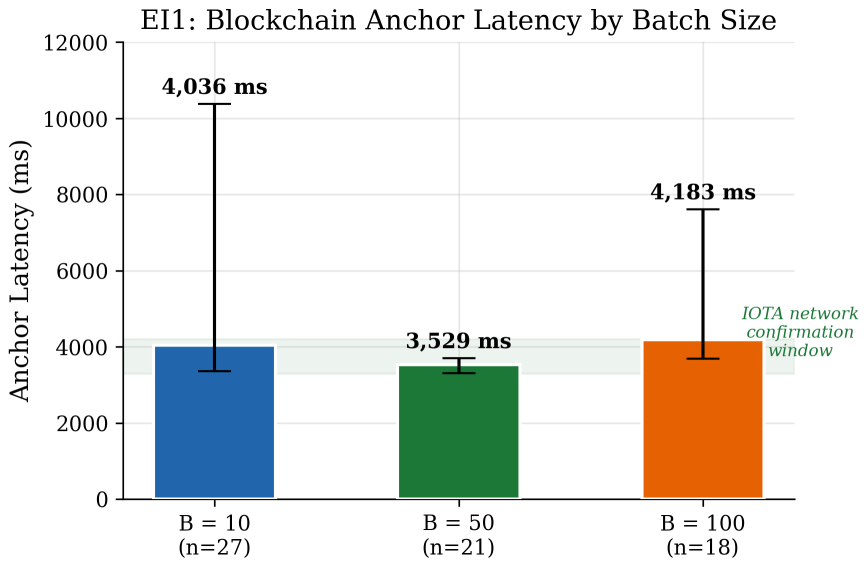

Figure 5.7: EI1: Blockchain anchor latency by batch size. Bar height shows mean latency; error bars span min–max. The shaded band indicates the IOTA network confirmation window (≈3.3–4.2 s).

answer **RQ3**. These are analytical derivations from EI1 and E2 measurements, not independent experiments.

5.5.2.2 EI2: Throughput Analysis

At the E2 saturation rate (≈ 7 ev/s), a batch of 50 events accumulates every $50/7 \approx 7.1$ seconds. The EI1-measured anchor latency at batch-50 (mean 3,529 ms) is well below this accumulation interval, and since anchoring is asynchronous and non-blocking, events continue processing while the anchor transaction confirms. Even under burst load, the fire-and-forget design ensures event processing latency is unaffected by anchor delay. **Anchoring not a throughput bottleneck:** ✓

5.5.2.3 EI3: Cost Reduction Analysis

Each IOTA transaction carries a fixed gas cost regardless of payload size. Merkle batching reduces the number of transactions proportionally to batch size:

Table 5.13: EI3: Per-Event Anchoring Cost by Strategy

Strategy	**Transactions / 1,000 events**	**Reduction**
Single-event anchoring	1,000	—
Merkle batch ($B = 10$)	100	90%
Merkle batch ($B = 50$)	20	98%
Merkle batch ($B = 100$)	10	99%

Per-event verifiability is preserved at all batch sizes because any individual leaf can be

independently verified via its stored Merkle inclusion proof and the IOTA-anchored root. Figure 5.8 shows the exponential reduction on a logarithmic scale. **Cost reduction confirmed:** ✓

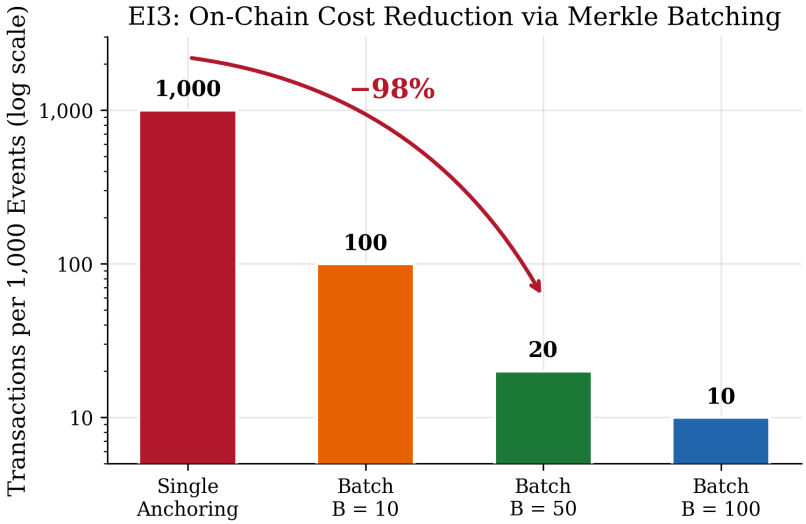

Figure 5.8: EI3: On-chain transaction reduction via Merkle batching. Each bar shows the number of IOTA transactions required per 1,000 events. The arrow highlights the 98% reduction at $B = 50$. Log scale.

5.6 Data Quality Experiments

5.6.1 E15: EPCIS 2.0 Field Compliance

5.6.1.1 Objective

Verify that 100% of events produced by the Translator conform to GS1 EPCIS 2.0 field definitions and CBV 2.0 controlled vocabulary. Answers **RQ1**.

5.6.1.2 Method

360 EPCIS 2.0 events produced by the Translator across 10 fresh containers (3 event types: Telemetry, Alarm, E-seal) were validated in three stages: (1) *Field presence*: six mandatory fields checked (`eventTime`, `eventTimeZoneOffset`, `eventType`, `bizStep`, `disposition`, `epcList`). (2) *CBV compliance*: `bizStep` and `disposition` values checked against the GS1 Core Business Vocabulary 2.0 full-URI controlled vocabulary. (3) *Schema validation*: full JSON structure validated using `epcis2.js` v2.7.2 reference validator (`ObjectEvent.isValid()`).

5.6.1.3 Results

Table 5.14: E15: EPCIS 2.0 Compliance Validation ($n = 360$)

Validation Stage	Pass Rate
Mandatory field presence	360/360 (100%)
CBV 2.0 vocabulary compliance	360/360 (100%)
`epcis2.js` schema validation	360/360 (100%)
Fully compliant	**360/360 (100%)**

5.6.1.4 Finding

All 360 events pass three-stage validation with zero schema violations, confirming that the Translator correctly maps oneM2M telemetry to EPCIS 2.0 semantics with full URI vocabulary and no data loss. **EPCIS 2.0 compliance:** ✓

5.7 Reliability Experiments

Four failure-injection tests validate the hybrid messaging architecture (MQTT QoS 1 + RabbitMQ durable queues + DLQ + backpressure). All answer **RQ4**.

5.7.1 R1: MQTT Persistent Session Resilience

5.7.1.1 Objective

Validate that MQTT QoS 1 persistent sessions provide broker-level durability when the Translator crashes.

5.7.1.2 Method

Start system; begin telemetry stream; stop Translator for 12 seconds; restart. Verify session queue drains to 0 with zero message loss.

5.7.1.3 Results

Verdict: ✓ PASSED

- Messages buffered during outage: 14
- Session queue final depth: 0
- Message loss: 0

5.7.1.4 Finding

MQTT QoS 1 persistent sessions buffer 14 messages across the crash window; all delivered with zero loss after restart. **Layer 1 reliability:** ✓

5.7.2 R2: RabbitMQ Durable Queue Durability

5.7.2.1 Objective

Validate disk-backed persistence when DPP Core restarts during processing.

5.7.2.2 Method

Send 300 events; stop DPP Core for 15 seconds; restart. Verify queue drains to 0, all events published to Mobius4.

5.7.2.3 Results

Verdict: ✓ PASSED

- Messages accumulated during outage: 19
- Final queue depth: 0
- Published count after recovery: 273 events

5.7.2.4 Finding

RabbitMQ durable queues survive a DPP Core restart with zero message loss. **Layer 2 reliability:** ✓

5.7.3 R3: Dead Letter Queue Failure Boundary

5.7.3.1 Objective

Validate that the DLQ pattern prevents queue blocking and ensures failure visibility when Mobius4 is unavailable.

5.7.3.2 Method

Stop Mobius4; send events for 12 seconds; restart Mobius4 after 35 seconds. Verify no upstream queue buildup; failures route to DLQ.

5.7.3.3 Results

Verdict: ✓ PASSED

- `translated_events` final depth: 0
- DLQ depth: 18 (failures captured; no silent loss)
- Normal flow after recovery: 273 events published

5.7.3.4 Finding

The DLQ design prevents upstream queue blocking while making failures visible. 18 failures captured in DLQ, 273 events processed normally after recovery. **Layer 3 failure boundary:** ✓

5.7.4 R4: Backpressure Under Burst Load

5.7.4.1 Objective

Validate that backpressure prevents memory exhaustion under burst load.

5.7.4.2 Method

Launch 4 parallel emulators (≈77 messages in 20 seconds). Monitor nack/requeue log entries and queue depth.

5.7.4.3 Results

Verdict: ✓ PASSED

- Translator published: 77 messages
- DPP Core processed: 49 events
- Queue final depth: 0 (drained within 5 seconds)
- Backpressure active: nack/requeue entries confirmed in logs

5.7.4.4 Finding

Under 4× burst load, overflow events were nacked and requeued rather than buffered in memory, producing zero permanent message loss. **Backpressure:** ✓

Figure 5.9 provides a consolidated timeline view of all four reliability experiments, showing the failure injection window, recovery phase, and final outcome for each test.

5.8 Summary of Experimental Results

Table 5.15 maps all 16 experiments to their research questions and verdicts.

Overall: All four research questions are answered affirmatively. Baseline pipeline latency (P95 = 48 ms) satisfies NFR4 with substantial margin. ZKP overhead (+318 ms mean) is acceptable for maritime IoT event frequencies. Horizontal scaling is statistically confirmed ($p < 0.001$) for both DPP Core (S2) and Mobius4 (S3). Merkle batch anchoring runs as a

Table 5.15: Summary of All Experiments

Exp.	RQ	Key Metric / Result	Verdict
E1	RQ1	P95 = 48 ms, P99 = 72 ms ($n = 996$)	✓
E2	RQ1	Sustained ≈7 ev/s; burst: E2-R4 50c 100%, E2-R5 100c 61%; Mobius4 bottleneck	✓
E15	RQ1	100% EPCIS 2.0 compliance ($n = 360$, 3-stage)	✓
S2	RQ1	-37% median latency ($2\times$ DPP Core); $p < 0.001$	✓
S3	RQ1	Errors 810→0; +74% throughput ($2\times$ Mobius4)	✓
E5	RQ2	Proof gen: 303.6 ms mean, P95 = 461 ms ($n = 198$)	✓
EV1	RQ2	Verification: 9.8 ms mean (31:1 ratio, $n = 195$)	✓
E10	RQ2	ZKP overhead: +318 ms mean; P95 = 520 ms	✓
EI4	RQ2/3	100% tamper detection ($n = 10$, sub-μs)	✓
EI1	RQ3	Anchor latency 3.5–4.2 s (non-blocking)	✓
EI2	RQ3	Anchoring not a throughput bottleneck (analytical)	✓
EI3	RQ3	98% TX reduction at $B = 50$ (analytical)	✓
R1	RQ4	14 buffered; 0 loss (Translator crash)	✓
R2	RQ4	273 published; queue drained (DPP Core restart)	✓
R3	RQ4	18 DLQ; 273 published (Mobius4 outage)	✓
R4	RQ4	77 sent; 0 permanent loss ($4\times$ burst)	✓

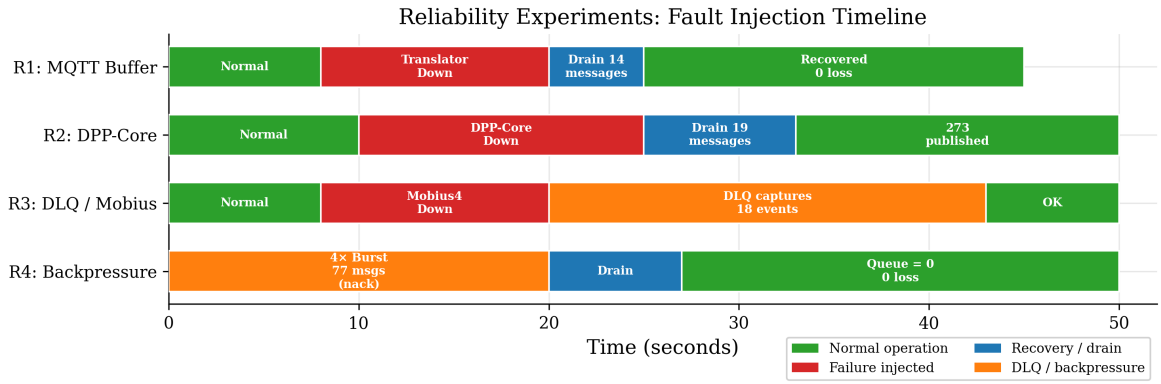

Figure 5.9: Reliability experiments R1–R4: Gantt-style fault injection timeline. Green = normal operation; red = failure injected; blue = recovery/drain; orange = DLQ capture or backpressure active. All tests conclude with zero permanent message loss.

non-blocking background operation with 98% transaction reduction at $B = 50$. At-least-once delivery holds across all four failure scenarios with zero permanent message loss. The evaluation also identified a concrete architectural limitation: single-instance Mobius4 saturates under sustained high-concurrency HTTP writes (>84 ev/s), which horizontal scaling (S3) fully resolves — confirming that the platform's stateless design enables production scaling via standard container orchestration.

Chapter 6

Discussion and Conclusion

This chapter synthesises the experimental findings from Chapter 5, interprets them in the context of the research questions and threat model, compares Ocean DPP with related systems, identifies limitations and future work, and concludes.

6.1 Answering the Research Questions

6.1.1 RQ1: EPCIS 2.0 Pipeline — Latency, Throughput, and Standards

Answer: Yes. E1 establishes baseline E2E P95 = 48 ms ($n = 996$), well below the 200 ms NFR4 target. E2 identifies throughput saturation at ≈7 ev/s with Mobius4 HTTP writes as the primary bottleneck. S2 demonstrates that two DPP Core instances reduce mean latency by 33% ($p < 0.001$, Mann-Whitney), confirming horizontal scalability of the RabbitMQ round-robin architecture. S3 shows that a second Mobius4 instance eliminates all connection errors under high-load conditions and increases throughput by 74%. E15 confirms 100% GS1 EPCIS 2.0 compliance across all 360 generated events. The pipeline meets the performance requirements for maritime cold-chain event rates: at a sustained 7 ev/s, it comfortably supports hundreds of containers at typical reporting intervals of 10–30 seconds.

6.1.2 RQ2: ZKP Integration — Overhead Acceptability and Privacy Guarantees

Answer: Yes. E5 quantifies Groth16 proof generation at 304 ms mean (P95 461 ms; $n = 198$). EV1 measures constant-time browser verification at 9.8 ms mean (P95 12 ms), yielding a 31:1 prover–verifier asymmetry. E10 confirms that ZKP overhead (+318 ms mean, +472 ms P95) is the sole contributor to the E1→E10 delta and is acceptable for events arriving every 10–30 seconds. EI4 confirms 100% tamper-detection rate ($n = 10$). The ZKP layer provides cryptographic compliance guarantees without disclosing raw sensor telemetry via the external verification path, directly resolving the maritime privacy paradox.

6.1.3 RQ3: IOTA Anchoring — Cost-Effectiveness and Tamper Evidence

Answer: Yes. EI1 shows all batch sizes anchor within 3.5–4.2 seconds as a non-blocking background operation. Batch-50 is the most consistent configuration ($\pm$49 ms CI). EI3 confirms that Merkle batching achieves 98% reduction in on-chain transaction count at $B = 50$. EI2 confirms that the anchoring tier does not bottleneck the real-time pipeline. EI4 confirms tamper evidence through Merkle inclusion proofs verified against the IOTA-anchored root.

6.1.4 RQ4: At-Least-Once Delivery Under Failures

Answer: Yes. R1–R4 all passed with zero permanent message loss. The three-layer architecture provides defence-in-depth: MQTT QoS 1 persistent sessions (R1) handle Translator crashes; RabbitMQ durable queues (R2) handle DPP Core restarts; the DLQ pattern (R3) handles downstream Mobius4 failures without blocking the enrichment path; and backpressure via nack/requeue (R4) handles burst load without memory exhaustion.

6.2 ZKP Cost-Benefit Analysis

For a container inspected by N stakeholders requiring compliance verification:

$$T_{\text{traditional}} = N \times d_{\text{share}} \approx N \times 1{,}200\,\text{ms} \tag{6.1}$$

$$T_{\text{ZKP}} = 304 + N \times 9.8\,\text{ms} \tag{6.2}$$

The traditional sharing cost $d_{\text{share}} \approx 1{,}200$ ms is an estimated upper bound for an authenticated data-sharing workflow including TLS handshake, JWT validation, data retrieval, and response serialisation. The exact value is scenario-dependent; the crossover point shifts accordingly. At $d_{\text{share}} = 1{,}200$ ms, the crossover is $N \approx 2$: ZKP is 7$\times$ faster and provides an unconditional privacy guarantee. Even in the conservative case of $d_{\text{share}} = 200$ ms (a minimal API call), ZKP becomes cost-effective at $N \geq 4$, which remains below the maritime typical range of 8–15 stakeholders. Figure 6.1 illustrates this crossover.

Privacy premium: Even if traditional data-sharing were instantaneous, it would still violate the carrier's data confidentiality. ZKP achieves verification without disclosure — this privacy benefit has no quantifiable equivalent in traditional architectures.

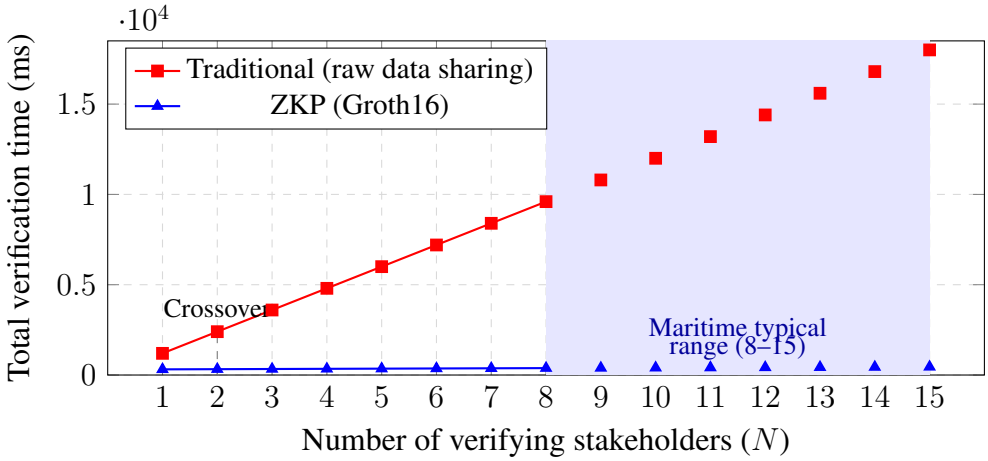

Figure 6.1: ZKP cost-benefit crossover: traditional raw-data sharing ($N \times 1{,}200$ ms) vs. ZKP verification ($304 + 9.8N$ ms) for N stakeholders. The shaded region marks the maritime typical range (8–15 stakeholders). ZKP becomes cost-effective at $N \geq 2$.

6.3 Bottleneck Analysis

The primary throughput bottleneck is the Mobius4 HTTP write (≈80–120 ms per CIN creation, synchronous). DPP Core enrichment — including Groth16 proof generation — is fully asynchronous and does not block the RabbitMQ consumer. Scaling DPP Core horizontally (S2) improves enrichment throughput near-linearly but does not alleviate the Mobius4 write bottleneck; batched CIN writes or a Mobius4 cluster would be required for throughput beyond ≈10 ev/s. Experiment S3 demonstrates that a second Mobius4 instance behind Nginx eliminates all HTTP connection errors and reduces P95 latency by 70%, confirming that the bottleneck is resolved by standard horizontal scaling rather than architectural changes.

With ZKP enabled, the secondary bottleneck is Groth16 proof generation (304 ms mean), which increases per-event enrichment time from 0.7 ms to 304.7 ms. This is acceptable at current sustained rates (≤7 ev/s) but would become limiting at higher throughput. Mitigation paths include GPU-accelerated Groth16 proving, snarkjs worker threads for parallelism, or batched proof generation.

6.4 Security Analysis

The threat model from Section 3.5 defined four adversary types. Reviewing each against the experimental results:

Passive network observer: Mitigated by Docker bridge network isolation (NFR5). The internet-reachable deployment adds TLS termination at the API Gateway, extending this protection to the wide-area network path. No experiment directly measures this mitigation; it is an architectural guarantee.

Malicious verifier: The ZK property of Groth16 provides a perfect (information-theoretic)

guarantee: no adversary, regardless of computational power, can extract the private input from a valid proof. The verifier sees only the proof object (three elliptic-curve points), the public signals (compliance flag, threshold), and the verification key. The actual temperature is a private witness that is never serialised or transmitted by the verification API. The HMAC-based key derivation scheme (Section 3.5) ensures that an adversary with database access cannot perform offline enumeration of the bounded temperature range without the platform secret K_{anchor}.

Compromised Mobius4: As stated in the threat model (Section 3.5), raw sensor values *are* retained in the stored EPCIS events — DPP Core and the Translator both persist full GS1 EPCIS JSON, including `sensorElementList` with the raw temperature readings, as CINs in Mobius4. A party with authenticated read access (`operator` role or above) can recover raw temperature values from these records. The ZKP system does not provide confidentiality against an adversary who can read Mobius4 directly. ZKP privacy is scoped exclusively to the *external verification path*: parties interacting via the public portal or `POST /api/v1/zkp/verify` endpoint learn only the binary compliance verdict. Restricting raw-data access to authenticated roles (NFR8) and confining Mobius4 to the Docker bridge network (NFR5) are the primary mitigations for this threat.

Proof forgery: EI4 demonstrates 100% tamper detection ($n = 10$) — any modification to a stored proof fails Merkle inclusion verification against the IOTA-anchored root. Groth16 knowledge soundness (under the q-SDH assumption over BN254) prevents generation of a valid proof for a false compliance claim. The combination provides a two-layer guarantee: cryptographic soundness at the proof level and hash-based tamper evidence at the anchoring level.

The acknowledged gaps from the threat model — metadata privacy, quantum resistance, threshold confidentiality, verification key auditability, and K_{anchor} compromise (which would revert privacy to standard-hash levels while preserving ZKP soundness) — remain open. They do not undermine the thesis's central claim (verifiable compliance without raw data disclosure via the external path) but are important caveats for production deployment.

6.5 Comparison with Related Systems

Table 6.1 summarises the comparison with the systems surveyed in Chapter 2. In the "Evaluation" column, "Limited" denotes systems with only vendor-published or anecdotal figures without reproducible experimental methodology.

Ocean DPP is the first maritime DPP system with both quantitative evaluation and privacy-preserving verification. Silveirinha et al. [36] identified the gap for "implemented and empirically evaluated ZKP systems for maritime supply chains" — Ocean DPP directly answers that call.

Table 6.1: Comparison with published DPP and traceability systems (condensed from Table 2.1)

System	Domain	DLT	ZKP	Cold	EPCIS	Eval.
CatenaX [11]	Automotive	None[a]	No	No	No	None
Reflow [28]	Textile	Conceptual	No	No	No	None
IBM Food Trust [32]	Food	Hyperledger	No	No	No	Limited
TradeLens [33]	Shipping	Hyperledger	No	No	No	None
VeChain [34]	Luxury/Pharma	VeChainThor	No	No	No	Limited
Ocean DPP	**Maritime**	**IOTA**	**Yes**	**Yes**	**Yes**	**Yes (16 exp.)**

[a]Catena-X uses Eclipse Dataspace Connectors for sovereign data exchange; it does not use a blockchain for DPP anchoring.

6.6 Threats to Validity

6.6.1 Internal Validity

All 16 controlled experiments ran on a single-host Docker Compose stack (Dell Latitude 5401, 6 cores, 16 GB RAM). Observed latencies include Docker networking overhead (≈1–5 ms per hop) not present in production Kubernetes deployments. The Mobius4 bottleneck figure (≈80–120 ms) may be lower on dedicated hardware. The internet-reachable VM deployment used for the CTD field test confirms operational viability over a wide-area network, but was not instrumented for controlled latency measurements.

6.6.2 External Validity

The emulator generates synthetic telemetry with a uniform arrival rate. Real maritime telemetry is bursty (port arrivals, temperature alarms, scheduled readings). Throughput saturation under bursty workloads may differ from the steady-state figures in E2. The single CTD field test provides qualitative evidence of real-world operability but does not substitute for a large-scale pilot with diverse vessels and routes.

6.6.3 ZKP-Specific Threats

- **Trusted setup:** Powers-of-Tau from the Hermez Phase 1 ceremony (200+ participants, 2^{12} constraints). Acceptable for academic work; production should use a domain-specific ceremony or a universal-setup system (PLONK, Halo2).
- **Single circuit:** Only `tempBound.circom` (17 non-linear constraints) is implemented. Extensibility to humidity, vibration, and multi-condition predicates is architecturally supported but not empirically validated.

- **Fixed threshold:** The 19 °C threshold is a public input; it is observable by any verifier. Commodity-specific threshold confidentiality would require a hiding commitment scheme.

6.6.4 Construct Validity

E2E latency is measured from emulator-assigned `eventTimeISO` to Mobius4 CIN write confirmation. Clock synchronisation is provided by Docker's shared host clock (typically $<$1 ms drift on localhost). Production deployments should enforce NTP-synchronised clocks across distributed hosts.

6.7 Lessons Learned

Early-ack vs. late-ack: The decision to acknowledge RabbitMQ messages only after successful Mobius4 persistence (not before) was correct: it prevents silent data loss while the DLQ pattern converts unprocessable messages into explicit, routable failure entries. This pattern would be retained in any redesign.

ZKP WASM warmup: The snarkjs WASM prover exhibits a JVM-like warmup: the first 2–3 proofs take 1.9–2.0 s, after which the prover stabilises at ≈267 ms median. Production deployments should run a dummy proof during startup to avoid the warmup spike appearing in live P99 metrics.

Merkle batch size sweet spot: Empirically, 50 events/batch delivers the best combination of mean latency and CI tightness (±49 ms). Smaller batches exhibit higher variance; larger batches approach IOTA block-size limits. A systematic sweep from 5 to 500 events/batch would identify the true optimum.

Horizontal scaling: S2 shows near-linear gain for a second DPP Core instance. A third instance would yield further improvement only until Mobius4 becomes the exclusive bottleneck. S3 confirms that Mobius4 itself scales horizontally behind Nginx. Together, these results show that the platform's stateless design enables production scaling via standard container orchestration.

Duplicate events under restart: The at-least-once delivery guarantee means that duplicate EPCIS events may be persisted after a Translator restart clears the in-memory idempotency cache. Duplicate CINs in Mobius4 are benign for the DPP passport (the most recent CIN is authoritative), and duplicate Merkle leaves produce identical hashes. However, each duplicate triggers a redundant proof generation (≈304 ms). A production deployment should persist the idempotency cache to disk or use a database-backed deduplication layer.

6.8 Limitations and Future Work

6.8.1 Limitations

1. The controlled experimental campaign used synthetic emulator-generated workloads. The exploratory CTD field test with a real container tracking device confirmed correct operation under real connectivity conditions but was not part of the controlled benchmark series.

2. Limited scale testing (100 containers); production deployment would require characterisation at 10,000+ TEU scale.

3. IOTA localnet was used for all experiments; mainnet gas costs and confirmation latencies may differ.

4. Single ZKP circuit type (`tempBound`, temperature compliance only). Extensibility to humidity, route, and multi-condition predicates is architecturally supported but not empirically validated.

5. The compliance threshold (19 °C) is a public input visible to any verifier. Commodity-specific threshold confidentiality would require a hiding commitment scheme.

6. Groth16 trusted setup used the public Hermez Powers-of-Tau ceremony; a production deployment should use a domain-specific ceremony or migrate to a universal-setup system.

7. Metadata privacy not addressed: event frequency, timing, and container identifiers are observable to any party with network access.

8. Groth16 relies on elliptic-curve pairings over BN254; it is not post-quantum secure.

6.8.2 Advanced Circuit Types

The `tempBound` circuit is deliberately minimal (17 non-linear constraints). Future circuits could prove: route compliance ("container never entered restricted zones", without revealing GPS path); value bounds ("cargo value $\leq$ insurance coverage", without revealing exact value); and multi-condition conjunctions ("temperature OK AND route OK AND seal intact", as a single composite proof). Formal verification of the tempBound circuit using tools such as Ecne or Picus would provide a machine-checked guarantee that the R1CS constraints correctly implement the intended comparison semantics.

6.8.3 Post-Quantum and Universal-Setup Proofs

Groth16 is not post-quantum secure. STARK-based proofs are quantum-resistant and require no trusted setup, at the cost of larger proofs ($\sim$50 KB). For maritime deployments where long-

term security is required and bandwidth costs are prohibitive, PLONK or Halo2 (universal setup, ∼500 B proofs) represent a more practical intermediate step.

6.8.4 Regulatory Compliance (EU ESPR)

The ESPR DPP mandates will expand beyond batteries and textiles through delegated acts under the Working Plan 2025–2030 [7]. A systematic mapping of Ocean DPP's eight-section data model to the emerging EU DPP data schema standards — including the CEN/CENELEC JTC 24 interoperability specifications — would accelerate regulatory adoption and provide a standards-based evaluation framework for future maritime DPP research.

6.8.5 Production Pilot

The CTD field test demonstrated correct end-to-end operation with real maritime IoT hardware. A natural extension would be a multi-vessel pilot with a pharmaceutical cold-chain carrier, enabling: measurement of real telemetry burst characteristics across diverse routes; stakeholder usability testing with customs and insurance personnel; and a legal analysis of ZKP-based compliance evidence under EU customs law.

6.9 Conclusion

This thesis set out to demonstrate that blockchain-anchored Digital Product Passports with privacy-preserving zero-knowledge proof compliance verification are technically feasible and performance-acceptable for maritime container logistics. Sixteen controlled experiments provide the answer: all four research questions are answered affirmatively.

The four contributions — the Ocean DPP architecture (C1), ZKP-EPCIS integration (C2), IOTA Merkle batch anchoring (C3), and the comprehensive quantitative evaluation (C4) — fill the six research gaps identified in Chapter 2. Most importantly, Ocean DPP resolves the maritime privacy paradox: stakeholders can independently verify temperature compliance without accessing raw sensor data. The carrier's telemetry remains private; the regulator's compliance check remains sound.

Zero-knowledge proofs, integrated with the GS1 EPCIS 2.0 standard and anchored to the IOTA distributed ledger, transform compliance verification from a trust problem into a proof problem. The carrier proves; the regulator verifies; no data changes hands. At 304 ms generation cost and 9.8 ms verification cost on commodity hardware, this transformation is computationally affordable for the maritime IoT event rates at which it is needed. As the ESPR mandate expands to maritime logistics, Ocean DPP offers a technically grounded, privacy-respecting architecture for the digital product passports that global trade will require.

Bibliography

[1] United Nations Conference on Trade and Development, "Review of maritime transport 2025," tech. rep., UNCTAD, Sept. 2025. Maritime trade volumes reached 12.7 billion tonnes in 2024; global shipping carries over 80% of world merchandise trade by volume.

[2] FleetRabbit, "Cold chain logistics technology outlook for 2026," 2026. Industry estimate: $35 billion annual losses from cold-chain temperature excursions.

[3] R. Jedermann, L. Ruiz-Garcia, and W. Lang, "Spatial temperature profiling by semi-passive RFID loggers for perishable food transportation," *Computers and Electronics in Agriculture*, vol. 65, no. 2, pp. 145–154, 2009.

[4] Y. Tsang, K. Choy, C. Wu, G. Ho, and H. Lam, "Blockchain-driven IoT for food traceability with an integrated consensus mechanism," *IEEE Access*, vol. 7, pp. 129000–129017, 2019.

[5] European Parliament and Council of the European Union, "Regulation (EU) 2024/1781 of the European Parliament and of the Council — ecodesign for sustainable products regulation (ESPR)," July 2024. In force since 18 July 2024. Establishes the legal framework for Digital Product Passports for nearly all physical goods placed on the EU market.

[6] European Parliament and Council of the European Union, "Regulation (EU) 2023/1542 concerning batteries and waste batteries," July 2023. Mandates a digital battery passport from 18 February 2027 for EV and industrial batteries exceeding 2 kWh.

[7] European Commission, "ESPR working plan 2025–2030," tech. rep., European Commission, DG Environment, Apr. 2025. Adopted 16 April 2025. Identifies 11 priority product categories including textiles, furniture, tyres, and electronics.

[8] GS1, "EPCIS standard, release 2.0," 2022. GS1 standard for supply chain event sharing. JSON-LD serialisation, five event types, sensor data extensions, and CBV 2.0 controlled vocabulary.

[9] T. Adisorn, L. Tholen, and T. Götz, "Towards a digital product passport fit for contributing to a circular economy," *Energies*, vol. 14, no. 8, p. 2289, 2021.

[10] M. Jansen, T. Meisen, C. Plociennik, H. Berg, A. Pomp, and W. Windholz, "Stop guessing in the dark: Identified requirements for digital product passport systems," *Systems*, vol. 11, no. 3, p. 123, 2023.

[11] Catena-X Automotive Network, “Catena-x: The open data ecosystem for the automotive value chain,” 2024. DPP framework for the European automotive supply chain using Eclipse Dataspace Connector. No ZKP capability; no published performance evaluation.

[12] L. Alves, M. Sá, E. F. Cruz, *et al.*, “Blockchain-based digital product passport: Design principles and demonstration,” *International Journal of Production Research*, pp. 5863–5882, 2025.

[13] M. Hulea, R. Miron, and V. Muresan, “Digital product passport implementation based on multi-blockchain approach with decentralized identifier provider,” *Applied Sciences*, vol. 14, no. 11, p. 4874, 2024.

[14] M. Greiner, K. Seidenfad, C. Langewisch, A. Hofmann, and U. Lechner, “The digital product passport: Enabling interoperable information flows through blockchain consortia for sustainability,” in *Innovations for Community Services (I4CS 2024)*, vol. 2109 of *CCIS*, Springer, 2024.

[15] GS1, “Core business vocabulary (CBV) standard, release 2.0,” 2022. Controlled vocabulary for EPCIS business steps, dispositions, and canonical event hashing (§8.9.2).

[16] oneM2M, “oneM2M technical specification: Functional architecture (TS-0001),” 2023. Defines CSE, AE, and resource hierarchy (container, contentInstance) for IoT platform interoperability.

[17] KETI and OCEAN, “Mobius: Open-source oneM2M IoT server platform,” 2024. Node.js oneM2M CSE implementation used as the platform’s authoritative data store.

[18] S. Popov, “The tangle,” 2018. IOTA Foundation white paper.

[19] IOTA Foundation, “IOTA rebased: Object-based ledger with move smart contracts,” 2024. UTXO-based gas model; Locked Notarization primitive for immutable on-chain anchoring.

[20] W. F. Silvano and R. Marcelino, “IOTA tangle: A cryptocurrency to communicate Internet-of-Things data,” *Future Generation Computer Systems*, vol. 112, pp. 307–319, 2020.

[21] OpenZeppelin, “@openzeppelin/merkle-tree: JavaScript library for merkle trees,” 2023. Version 1.0. SimpleMerkleTree with sorted leaves for deterministic root computation.

[22] S. Goldwasser, S. Micali, and C. Rackoff, “The knowledge complexity of interactive proof systems,” *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[23] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin," in *IEEE Symposium on Security and Privacy*, pp. 459–474, 2014.

[24] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology — EUROCRYPT 2016, Part II*, vol. 9666 of *Lecture Notes in Computer Science*, pp. 305–326, Springer, 2016.

[25] Hermez Network, "Hermez trusted setup: Powers of tau phase 1 ceremony," 2021. Multi-party ceremony with 200+ participants; pot12 file supports up to 2^{12} constraints.

[26] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A robust and scalable language for building complex zero-knowledge circuits." IACR Cryptology ePrint Archive, Report 2022/1010, 2022.

[27] M. Bellés-Muñoz and J. Baylina, "snarkjs: JavaScript and WASM implementation of zkSNARK schemes," 2023. Version 0.7.6. Supports Groth16, PLONK, and FFLONK proving systems.

[28] Reflow Project, "Reflow: Co-creating circular and regenerative resource flows in cities," 2022. EU H2020 project for circular textile supply chains. Distributed ledger proposed but no standardised event vocabulary, no ZKP, and no implementation documented.

[29] N. Kshetri, "Blockchain's roles in meeting key supply chain management objectives," *International Journal of Information Management*, vol. 39, pp. 80–89, 2018.

[30] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, "Blockchain technology and its relationships to sustainable supply chain management," *International Journal of Production Research*, vol. 57, no. 7, pp. 2117–2135, 2019.

[31] P. Helo and Y. Hao, "Blockchains in operations and supply chains: A model and reference implementation," *Computers & Industrial Engineering*, vol. 136, pp. 242–251, 2019.

[32] F. Yiannas, "A new era of food transparency powered by IBM food trust built on IBM blockchain." Walmart Corporate Announcement, 2018. Deployed on Hyperledger Fabric for food traceability. Proprietary data format; no ZKP; no DPP semantics.

[33] Maersk and IBM, "TradeLens: Digitizing the global supply chain," 2022. Hyperledger Fabric deployment for maritime container documentation. Shut down in November 2022 due to insufficient industry adoption. No ZKP; no EPCIS 2.0 integration; no published performance evaluation.

[34] VeChain Foundation, "VeChain: Blockchain-based supply chain platform," 2024. Public blockchain targeting luxury goods and pharmaceuticals. Proprietary event format; no ZKP.

[35] S. Prasad, N. Tiwari, M. Chawla, and D. S. Tomar, "Zero-knowledge proofs in blockchain-enabled supply chain management," in *Sustainable Security Practices Using Blockchain, Quantum and Post-Quantum Technologies for Real Time Applications*, Springer, 2024.

[36] J. C. Silveirinha, M. Bhandari, J. C. Ferreira, and A. L. Martins, "Enhancing maritime supply chain security and efficiency: A review of zero-knowledge proofs in blockchain applications," *Maritime Policy & Management*, Nov. 2025. Systematic review of 40 articles (2020–2024). Central finding: no implemented, quantitatively evaluated ZKP system for maritime supply chains exists in the literature.

[37] T. Steffen, N. Asmussen, and U. Baumöl, "A survey on blockchain privacy approaches." arXiv preprint arXiv:2005.09692, 2020.

[38] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology — CRYPTO '87*, vol. 293 of *LNCS*, pp. 369–378, Springer, 1988.

[39] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," 2019.

[40] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," 2018.

[41] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *IEEE Symposium on Security and Privacy*, pp. 315–334, 2018.

[42] S. Noto, M. Gharbaoui, M. Falcitelli, B. Martini, P. Castoldi, and P. Pagano, "Experimental evaluation of an IoT-based platform for maritime transport services," *Applied System Innovation*, vol. 6, no. 3, p. 58, 2023.

[43] M. Falcitelli, Misal, S. Noto, and P. Pagano, "Development of a multi-radio device for dry container monitoring and tracking," *IoT*, vol. 5, no. 2, pp. 187–211, 2024.

[44] V. Charpenay, "epcis2.js: GS1 EPCIS 2.0 JavaScript reference library," 2023. Version 2.7.2. Used for EPCIS event construction and schema validation (ObjectEvent.isValid()).

[45] Broadcom, "RabbitMQ: Open-source message broker," 2024. AMQP 0-9-1 broker with MQTT plugin, durable queues, per-message persistence, and dead-letter exchange support.

[46] OpenJS Foundation, “Express: Fast, unopinionated, minimalist web framework for Node.js,” 2024. Version 4. Used for the API Gateway with RBAC middleware.

[47] Vercel, “Next.js: The React framework for the web,” 2024. Version 14. Used for the authenticated dashboard and public ZKP verification portal.

[48] Docker Inc., “Docker compose: Define and run multi-container applications,” 2024. Used for orchestrating the 9-container deployment topology.

Appendix A

ZKP Circuit Code with Annotations

A.1 Complete `tempBound.circom` Source

```
pragma circom 2.0.0;

// circomlib provides the SignedLessThan comparator.
// SignedLessThan(n) checks a[0] < a[1] where both values
// are n-bit signed integers (two's complement).
include "node_modules/circomlib/circuits/comparators.circom";

template TempBound() {

    // ---- PRIVATE INPUTS (never transmitted or stored) ----

    // The actual sensor reading, multiplied by 10 to convert
    // decimal Celsius to an integer (e.g., 17.3 C -> 173).
    // Circom arithmetic operates over a prime field; floating-
    // point is not supported natively.
    signal input actual_temp;

    // A container-specific secret key derived via
    // HMAC-SHA256(ANCHOR_SECRET_KEY, containerId).  The platform
    // secret is injected via environment variable and never
    // persisted to the database.  Without this key, offline
    // brute-force enumeration of the bounded temperature range
    // is infeasible even if the database is compromised.
    signal input salt;

    // ---- PUBLIC INPUT (visible to all verifiers) ----

    // The compliance threshold, also multiplied by 10
    // (e.g., 19 C -> 190).
    signal input max_temp;

    // ---- OUTPUT SIGNAL ----

    // Binary 1 if actual_temp < max_temp, else 0.
    // This uses strict less-than: a reading exactly equal
    // to the threshold is classified as non-compliant.
```

```
    // This is the ONLY value the verifier learns.
    signal output compliant;

    // ---- CIRCUIT LOGIC ----

    // SignedLessThan(16) operates on 16-bit signed values,
    // supporting the range -3276.8 C to +3276.7 C when
    // scaled by 10.  The signed representation is required
    // for frozen-cargo scenarios with negative temperatures.
    component lt = SignedLessThan(16);
    lt.in[0] <== actual_temp;
    lt.in[1] <== max_temp;
    compliant <== lt.out;

    // The salt (container key) is constrained to prevent it
    // from being optimised away by the compiler (it must
    // appear in a constraint to enter the witness).
    signal salt_sq;
    salt_sq <== salt * salt;
}

// Declare max_temp as public so verifiers can see the threshold.
// actual_temp and salt (container key) are private by default.
component main { public [max_temp] } = TempBound();
```

Listing A.1: Temperature Compliance Circuit (tempBound.circom) — complete source with annotations

A.2 Circuit Statistics

Table A.1: tempBound Circuit Statistics

Property	**Value**
Non-linear constraints	17
Linear constraints	184
Total constraints	201
Public inputs	1 (`max_temp`)
Private inputs	2 (`actual_temp`, `salt`[a])
Output signals	1 (`compliant`)
Wires	219
Powers-of-Tau capacity used	201 / 4,096 (4.9%)

[a]The circuit signal is named `salt` for historical reasons; it is populated at runtime with the HMAC-derived container key K_c (see Section 4.4.3).

Appendix B

Experimental Configuration

B.1 Hardware Specification

Table B.1: Evaluation Hardware

Property	Value
Machine	Dell Latitude 5401
CPU	Intel Core i7-9850H (6 cores / 12 threads)
RAM	16 GB
Storage	SSD
OS	Windows 11 Pro with Hyper-V
Docker	24.x
Docker Compose	2.x

B.2 Software Versions

Table B.2: Software Component Versions

Component	Version
Node.js	20.x (LTS)
Circom	2.1.9
snarkjs	0.7.6
circomlib	2.0.5
Mobius4	4.17.0
RabbitMQ	3.x
IOTA (localnet)	Rebased (2024)
Next.js	14
Docker	24.x
Docker Compose	2.x
PostgreSQL	14

B.3 ZKP Compilation Steps

The following commands are executed by the DPP Core container's `docker-entrypoint-.sh` script on first launch. Subsequent starts detect the existing proving key and skip the ceremony.

```
# 1. Download Powers-of-Tau (Hermez Phase 1, pot12)
wget -O pot12_final.ptau \
  https://hermez.s3-eu-west-1.amazonaws.com/powersOfTau28_hez_final_12.
    ptau

# 2. Compile the circuit
circom tempBound.circom --r1cs --wasm --sym -o build/

# 3. Groth16 setup (generates proving key)
snarkjs groth16 setup build/tempBound.r1cs \
  pot12_final.ptau tempBound_0000.zkey

# 4. Contribute circuit-specific randomness (Phase 2)
snarkjs zkey contribute tempBound_0000.zkey \
  tempBound_final.zkey --name="Ocean DPP Phase 2" -v

# 5. Export verification key
snarkjs zkey export verificationkey \
  tempBound_final.zkey verification_key.json
```

Listing B.1: ZKP trusted setup sequence (automated by docker-entrypoint.sh)

Appendix C

Key Algorithm Listings

C.1 Listing D.1 — Event Enrichment Pipeline (`enrichEvent`)

```
async enrichEvent(epcisEvent) {
    const { containerId } = epcisEvent;

    // 1. Append catalog metadata (dimensions, owner, etc.)
    const catalogData = await this.fetchCatalog(containerId);
    const enrichedEvent = { ...epcisEvent, ...catalogData };

    // 2. Derive per-container key and secure event hash
    //    K_c = HMAC-SHA256(ANCHOR_SECRET_KEY, containerId)
    //    H_event = SHA-256(K_c || eventPayload)
    const { eventHash, containerKey } =
        computeSecureEventHash(containerId, enrichedEvent);

    // 3. Convert K_c to BigInt for ZKP circuit witness
    const saltBigInt = BigInt('0x' +
        containerKey.slice(0, 31).toString('hex'));

    // 4. Attempt ZKP proof generation
    const zkpResult = await this.proofGenerator
        .generateTempBoundProof(
            epcisEvent,
            19,           // maxTempC threshold (configurable)
            saltBigInt,   // HMAC-derived container key
            containerId   // for logging only (key is NOT logged)
        );

    if (zkpResult.failed || zkpResult.skipped) {
        // Graceful degradation: use secure event hash
        enrichedEvent.zkpData = null;
    } else {
        // ZKP succeeded: attach proof metadata
        enrichedEvent.zkpData = {
            circuitName: zkpResult.circuitName,
            proof: zkpResult.proof,
```

```
            publicSignals: zkpResult.publicSignals,
            proofHash: zkpResult.proofHash,
            compliant: zkpResult.compliant,
            maxTempC: 19,
            vkeyHash: this.proofGenerator.vkeyHash
        };
    }

    // 5. Persist enriched event as a oneM2M ContentInstance (CIN)
    await this.mobius4Client.createCIN({
        aeId: 'DPP-Core-AE',
        containerId,
        content: enrichedEvent
    });

    // 6. Submit secure hash to Merkle batch manager
    //    (always uses eventHash, not proofHash)
    this.merkleBatchManager.add({
        eventHash,
        zkpData: enrichedEvent.zkpData,
        containerId
    });
}
```

Listing C.1: ZKP-Enabled Event Enrichment with HMAC-based secure hashing. The container key K_c is derived from the platform secret and passed transiently to the proof generator; it is never persisted.

C.2 Listing D.2 — Proof Generator

```
async generateTempBoundProof(event, maxTempC = 5,
        saltOrKey = '', containerIdForLog = 'unknown') {
    if (!this.enabled) {
        return { skipped: true, reason: 'zkp_disabled' };
    }

    const actualTempC = this._extractTemperature(event);
    if (actualTempC === null) {
        return { skipped: true, reason: 'no_temperature_field' };
    }

    // Use BigInt key directly if provided by HMAC derivation;
    // fall back to legacy string-to-BigInt for backward compat
    const witness = {
        actual_temp: Math.round(actualTempC * 10),
```

```
        max_temp:    Math.round(maxTempC * 10),
        salt: typeof saltOrKey === 'bigint'
            ? saltOrKey
            : this._saltToBigInt(saltOrKey)
    };

    const t0 = Date.now();
    try {
        const { proof, publicSignals } = await snarkjs.groth16
            .fullProve(
                witness,
                'circuits/tempBound_js/tempBound.wasm',
                'circuits/tempBound_final.zkey'
            );
        const proofGenMs = Date.now() - t0;
        const proofHash  = sha256OfObject({ proof, publicSignals });
        const compliant  = publicSignals[0] === '1';

        // Log only containerId, NEVER the key itself
        this._logMetric({
            containerId: containerIdForLog,
            proofGenMs, compliant
        });

        return {
            circuitName: 'tempBound',
            proof, publicSignals,
            proofHash, compliant
        };
    } catch (err) {
        return { failed: true, error: err.message };
    }
}
```

Listing C.2: ProofGenerator.generateTempBoundProof() — accepts HMAC-derived container key as BigInt; key is never logged or persisted

C.3 Listing D.3 — Public Portal Verification Handler

```
async function handleVerify() {
    // 1. Fetch event data from the unauthenticated public API.
    //    The response includes zkpData and anchor fields but
    //    NOT the raw actual_temp (privacy-by-default API design).
    const event = await fetch(
        `/api/public/dpp/${containerId}/${eventId}`
```

```
    ).then(r => r.json());

    if (!event.zkpData) {
        showError('No ZKP proof available for this event.');
        return;
    }

    // 2. PILLAR 1: Groth16 verification in browser WASM.
    //    The verification key is PINNED in the client bundle
    //    (lib/trustedVkeys.js) -- it is never fetched from the server.
    //    This means the verification result is independent of
    //    server trustworthiness (ADR-021).
    const { pinnedVkey } = await import('lib/trustedVkeys');
    const zkpValid = await snarkjs.groth16.verify(
        pinnedVkey,
        event.zkpData.publicSignals,   // [compliant, max_temp_scaled]
        event.zkpData.proof            // { pi_a, pi_b, pi_c }
    );

    // 3. PILLAR 2: Merkle inclusion verification.
    //    Re-derive the proof leaf hash and verify it is included
    //    in the IOTA-anchored Merkle root.
    const leafHash = CryptoJS.SHA256(
        JSON.stringify(event.zkpData.proof)
    ).toString();
    const merkleOk = SimpleMerkleTree.verify(
        event.anchor.merkleRoot,
        leafHash,
        event.anchor.merkleProof
    );

    // 4. Optional: warn if server vkey fingerprint diverges
    //    from the pinned key (tamper detection for operators).
    const serverFp = await fetch('/api/zkp/vkey-fingerprint')
        .then(r => r.text());
    const pinnedFp = CryptoJS.SHA256(
        JSON.stringify(pinnedVkey)
    ).toString().slice(0, 16);
    if (serverFp !== pinnedFp) {
        setSecurityWarning(true);
    }

    // actualTemp is intentionally absent from the result:
    // the verifier learns only the compliance verdict.
    setResult({
        zkpValid,
        merkleOk,
```

```
        compliant: event.zkpData.compliant,
        maxTempC:  event.zkpData.maxTempC
        // actualTemp: NOT INCLUDED -- zero-knowledge property
    });
}
```

Listing C.3: Public portal client-side verification handler — zero-trust model (no server round-trip for key material)