End-to-End Integrity for Wind Farm SCADA Telemetry Using a Permissioned Blockchain Framework

Arvin Asrari Ershad

A Thesis

in

The Department

of

Concordia Institute for Information Systems Engineering (CIISE)

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

March 2026

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Arvin Asrari Ershad**

Entitled: **End-to-End Integrity for Wind Farm SCADA Telemetry Using a Permissioned Blockchain Framework**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Jun Yan

_____ Examiner
Dr. Farnoosh Naderkhani

_____ Supervisor
Dr. Mohsen Ghafouri

Approved by _____
Chun Wang, Chair
Department of Concordia Institute for Information Systems Engineering (CIISE)

_____ 2026

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

End-to-End Integrity for Wind Farm SCADA Telemetry Using a Permissioned Blockchain Framework

Arvin Asrari Ershad

Due to their numerous advantages, Wind Farms (WFs) have become more and more integrated into modern power systems. In such a cyber-physical system, secure integration requires addressing challenges, such as data security, integrity, and trust between distributed control units. Specifically, while being vulnerable to tampering or loss, the integrity of Wind Turbine (WT) measurement data is critical for reliable monitoring, fault diagnosis, and cyber-physical security. On this basis, to address this issue, we propose two blockchain-based frameworks to ensure the integrity of (i) data transfer within the WF from the point of measurement to the control room and (ii) from the control rooms of WFs to the grid operator, both augmented with a per-turbine hash chain across measurement windows for stream integrity. To do so, in the first architecture, Hyperledger Fabric is used to provide a tamper-evident communication and recording layer for transmitting WT data within the WF network, designed based on IEC 61400-25. Turbine side industrial PCs (IPCs) act as edge client nodes that submit transactions, while control room servers support monitoring and verification. To handle the high sampling rate of turbine measurements, raw data samples are grouped into fixed-duration time windows and submitted as a single transaction per window. Each window carries integrity information that links it to the previously accepted window of the same turbine, forming a per-turbine hash chain that is enforced during endorsement before commit. This complements the Hyperledger Fabric's block hash chain by providing stream-level continuity for each turbine.

Acknowledgments

I want to begin by expressing my deepest gratitude to my supervisor, Dr. Mohsen Ghafouri. Throughout this work, he supported me far beyond what I could have expected, with patience, care, and constant guidance. He helped me grow as a researcher and supported me through the publication process, step by step. His mentorship shaped not only this research, but also my confidence, perspective, and growth throughout this journey, and for that I will always be deeply grateful.

I am also sincerely grateful to Dr. Farnoosh Naderkhani. At a critical time in my studies, she helped me secure supervision, encouraged me to keep going, and supported me until the finish line.

Susan Ayerman and Behrouz Asrari Ershad, thank you for being the steady foundation of my life. Your love, sacrifices, and support have been with me in every chapter of this journey, including the days when I was exhausted, discouraged, or unsure of myself.

Mojgan Ayerman, thank you for your kindness, your quiet support, and the comforting way you have always been there for me. Your care has meant a lot to me, more than I can properly express.

To my beautiful little sister, Arina Asrari, thank you for bringing warmth and happiness into my life. You have a way of making stressful days feel lighter, simply by being you.

I also want to thank Saeed Ayerman, Mahnaz Jamali, Sina, and Sayna, and especially Mahnaz, who listened to me day after day, even when all I talked about was my thesis, my stress, and the same repeating worries. I am truly sorry for my boring conversations. Your patience, support, and help carried me through this period.

Golnaz Mirzaei, thank you for being beside me through this journey and for everything you did to help me reach this point. Thank you for your time, your support, and the way you helped me keep going when it was hard.

To Ali Raeisdanaei and Alireza Toghiani, thank you for your friendship and support through all stages of my studies. Having you in my life made this long path easier and less lonely. In moments of stress, uncertainty, and exhaustion, knowing that I had your encouragement and understanding meant more than I can fully express. I am deeply grateful for your kindness, your companionship, and for all the ways you helped make this path more bearable and meaningful.

Finally, I dedicate this thesis to my beloved Aziz Jon. She is no longer with us, but her love remains with me always, and I carry her memory in everything I do.

Contents

List of Figures **ix**

List of Tables **xi**

List of Acronyms xii

1 Introduction **1**

1.1 Motivation and Context 1

1.2 Problem Statement 2

1.2.1 Research Objectives 4

1.3 Thesis Contributions 6

1.4 Thesis Organization 7

2 Literature Review **8**

2.1 WF Security 8

2.2 Blockchain for Integrity and Auditability 9

2.3 Blockchain in Energy Streams 10

2.4 Blockchain in Wind Energy 11

2.5 Research Gap 12

3 Wind Farm Telemetry and System Context **13**

3.1 WF Physical Layer 13

3.1.1 Benchmark system used in this thesis 14

3.1.2 Transmission-system benchmark used for Framework II 14

3.2 WF Data Transfer and SCADA Architecture 17
3.2.1 Layered monitoring and control 17
3.2.2 Communication segments within the telemetry path 17
3.3 Data Transmission Characteristics 18
3.3.1 Telemetry signals used in this thesis 18
3.3.2 Typical sensor/telemetry rates 18
3.4 Trust Zones and Attack Surface 20

4 Background on Blockchain and Platform Selection 22

4.1 Blockchain Fundamentals 22
4.1.1 Blocks, hashes, and tamper-evidence 22
4.1.2 Core roles in a blockchain system 23
4.2 Permissioned vs. Permissionless Networks and Rationale for Fabric 24
4.2.1 Fit to wind-farm telemetry and operational constraints 24
4.2.2 Indicative performance and settlement characteristics 25
4.3 Hyperledger Fabric Components 25
4.4 Transaction Processing in Hyperledger Fabric 26
4.4.1 Endorse–order–validate–commit pipeline 27
4.4.2 Where integrity checks can be enforced 28

5 Methodology and prototype 29

5.1 Framework I & II overview 29
5.2 Framework I Implementation 31
5.3 Framework II Implementation 32
5.4 WF data simulation and ingestion 32
5.5 Windowing and per-window hash-chain construction (Framework I) 33
5.6 Chaincode and ledger data model 34

6 Evaluation 35

6.1 Experimental setup 35

6.2 Performance evaluation 38
6.3 Integrity evaluation 43

7 Conclusion **49**

Bibliography **51**

List of Figures

Figure 3.1 Single-mode detailed EMTP WF benchmark: four clusters (I–IV) connected at the POI. Cluster modeling granularity follows [1]. 15

Figure 3.2 Detailed turbine-group subsystem used in this thesis: a 12-WT model inside Cluster IV of the EMTP benchmark. 16

Figure 3.3 MATPOWER `case39` (New England 39-bus) benchmark used to generate WF bus telemetry for Framework II. 16

Figure 3.4 Reference WF SCADA communication architecture and layered data and command flow (adapted from [2, 3]). 19

Figure 4.1 Conceptual blockchain structure (block header and body) illustrating hash-based linking between blocks. 23

Figure 4.2 High-level Hyperledger Fabric architecture: clients submit proposals, peers simulate and endorse, the ordering service batches transactions into blocks, and peers validate and commit blocks on the channel ledger [4]. 27

Figure 5.1 Framework deployment and data path in the implementation. 30

Figure 6.1 Hyperledger Explorer dashboard used to monitor the Fabric network during experiments (channel status, blocks, transactions, and chaincode activity). 37

Figure 6.2 Commit-latency percentile summary for Evaluation #1 (baseline, sequential commits) and Evaluation #2 (24-turbine concurrent load) under the two orderer batching configurations (Config L and Config T). Bars report p50, p95, and p99 end-to-end commit latency measured at the client using synchronous invoke (`--waitForEvent`) over `VALID` transactions only. 39

Figure 6.3 Adversary model and attack surface considered in the integrity experiments. 43

Figure 6.4 Evaluation #6 evidence: controlled post-commit corruption of the victim peer block store (`blockfile_000009`) and detection via cross-peer comparison of QSCC `GetBlockByNumber` raw outputs for block $B{=}8741$. 45

Figure 6.5 Evaluation #7 evidence log for hash-chain enforcement against a forged stream continuation (A) tail before, (B) forged payload, (C) endorsement rejection, (D) tail after. 46

Figure 6.6 Evaluation #8 evidence log for continuity enforcement in Framework II against a forged continuation attempt. The submitted `previous_hash` does not match the current tail state, so the transaction is rejected before ordering and commit. 47

Figure 6.7 Evaluation #9 representative evidence log for delayed submission handling in Prototype II. Buffered records can be committed after communication recovery when replayed sequentially with correct hash linkage, whereas out-of-sequence replay is rejected. 48

List of Tables

Table 3.1 WF measurement parameters and transmission characteristics (sampling rate). Turbine-side sensor sampling rates follow Table 7 in [5]. For context on SCADA aggregation, [6] notes that conventional SCADA commonly uses 10-min averages, while high-frequency SCADA can be recorded at 100 Hz. 20

Table 4.1 Comparison of selected blockchain platforms (throughput and settlement characteristics). 25

Table 5.1 Hyperledger Fabric network topology in Framework I 32

Table 5.2 Hyperledger Fabric network topology in Framework II. 32

Table 6.1 Orderer batching configurations used in evaluation. 37

Table 6.2 Evaluation environment (software and host configuration). 37

Table 6.3 Evaluation #1 baseline commit latency comparison (200 sequential windows, T=1). 38

Table 6.4 Evaluation #2 concurrent workload results (T=24, warmup=30 s, measure=120 s, Δ=1 s). 39

Table 6.5 Evaluation #3 window duration sweep (Config L, T=24, warmup 60 s, measure 180 s). 40

Table 6.6 Evaluation #4 payload size sweep under concurrent load (Config L, T=24, Δ=1 s, 3 repeats; values averaged across repeats). 41

Table 6.7 Evaluation #5 sampling-rate sweep using per-window aggregation (Config L, T=24, warmup 60 s, measure 180 s, Δ=1 s). 42

Table 6.8 Evaluation #5 sampling-rate sweep using per-window aggregation (Config T, T=24, warmup 60 s, measure 180 s, Δ=1 s). 42

List of Acronyms

CA Certificate Authority

CLI Command-Line Interface

CPU Central Processing Unit

CPS Cyber-Physical System

DFIG Doubly-Fed Induction Generator

DoS Denial-of-Service

EMT Electromagnetic Transient

EMTP Electromagnetic Transients Program

GW Gigawatt

Hz Hertz

ICS Industrial Control System

IEC International Electrotechnical Commission

IIoT Industrial Internet of Things

IPC Industrial PC

IoBC Internet of Blockchain

LAN Local Area Network

MSP Membership Service Provider

OT Operational Technology

PF Power Factor

PKI Public Key Infrastructure

PMU Phasor Measurement Unit

POI Point of Interconnection

PMSG Permanent Magnet Synchronous Generator

QSCC Query System Chaincode

RAM Random Access Memory

SCADA Supervisory Control and Data Acquisition

SHA-256 Secure Hash Algorithm (256-bit)

TLS Transport Layer Security

TPS Transactions Per Second

TRS Trace Replay Service

WAN Wide Area Network

WF Wind Farm

WFAPC Wind Farm Active Power Control

WFC Wind Farm Controller

WT Wind Turbine

WTCP Wind Turbine Control Panel

Chapter 1

Introduction

1.1 Motivation and Context

The global transition towards renewable energy has accelerated the deployment of Wind Farms (WFs) as key contributors to sustainable power generation [7]. In 2024, global renewable capacity additions increased by an estimated 25% to around 700 GW, with wind accounting for about 17% of these additions [7]. Consistent with this trend, the wind industry installed a record 117 GW of new capacity in 2024, bringing global total installed wind capacity to approximately 1,136 GW by the end of 2024 [8]. Wind capacity is expected to keep growing in 2025, and 981 GW of additional wind capacity is forecast globally by 2030 [8].

Beyond deployment scale, wind power has become a practical pillar of power-system decarbonization because it delivers low-emission electricity at increasingly competitive cost. Recent global assessments indicate that the unit costs of wind energy decreased by 55% from 2010 to 2019 [9]. Consistent with this trend, newly commissioned onshore wind projects have frequently been reported as cheaper than the most cost-competitive fossil-fuel-fired alternatives, strengthening the economic case for sustained WF expansion [10].

Modern WFs increasingly rely on Supervisory Control and Data Acquisition (SCADA) systems and Industrial Internet of Things (IIoT) devices to monitor turbine performance, collect sensor data, and support operational decision-making [11, 12]. As telemetry exchange occurs over networked infrastructures, ensuring integrity and trust in operational telemetry transfer becomes a practical

requirement for reliable monitoring and cyber-physical security. A detailed review of integrity risks, representative incidents, and prior work in WF OT and SCADA environments is provided in Chapter 2, which motivates the need for decentralized, tamper-evident mechanisms for operational telemetry delivery in wind-energy networks.

In practice, this telemetry path can be viewed in two segments. The first segment is *within* a WF, where turbine-side devices generate measurements that pass through the OT network and are consumed in the WF control room. The second segment is *across* WFs, where WF control-room services forward selected operational telemetry to a centralized control center for monitoring and archival. These segments differ in scale and trust assumptions, but the integrity goal is the same. After telemetry is recorded for monitoring, it should not be possible to change past records without detection. It should also not be possible to append a forged next record that breaks the true sequence. In this thesis, Framework I addresses the first segment (WT-to-control-room), while Framework II addresses the second segment (WF-to-control-center).

In operational environments, WF activities are increasingly supported by continuous data analytics, and SCADA telemetry is used directly for condition monitoring, performance assessment, and operational decision making at both the individual turbine and WF levels [13]. If telemetry data is altered, monitoring algorithms and analytical models can be misled, and post-event diagnosis can become less reliable. These operational dependencies highlight the need for integrity mechanisms that can be deployed within the WF while operating efficiently under practical communication bandwidth and data-rate limitations [14].

1.2 Problem Statement

Wind turbine (WT) measurement data are essential for reliable monitoring, fault diagnosis, and performance supervision in WF operation. In practice, WF SCADA telemetry is a primary input for data-driven condition and performance monitoring, and it supports operational decision-making at both turbine and WF levels [13].

These measurements are generated and transported within the WF's *operational technology*

(OT) environment, i.e., the industrial control and monitoring infrastructure that interfaces with physical processes and exchanges SCADA data [14]. Within the WF boundary, IEC 61400-25 defines structured information exchange for monitoring and control of wind power plants, implying that telemetry is produced by turbine-side devices and delivered through internal communication and supervisory services to control-room consumers [11]. As a result, the end-to-end telemetry transfer path typically spans multiple components (turbine-side acquisition, communication links, and control-room SCADA services), and the integrity of the data can be impacted at any stage of this pipeline.

This thesis studies integrity across both segments of the telemetry path. Framework I focuses on *in-farm* telemetry transfer from WT-side acquisition and local communication services to the WF control room. Framework II extends the framework to a *multi-farm* setting, where each WF submits telemetry records to a shared control-center blockchain network. Framework II is used to evaluate the same integrity idea under a broader network topology.

However, OT environments are exposed to integrity and availability risks that can affect telemetry transfer, including unauthorized *modification* or *injection* of measurements, as well as *omission* or *re-ordering* of records due to failures or adversarial actions in communication and storage components [14]. For WF telemetry specifically, these risks create two practical integrity requirements for control-room monitoring and audit: (i) the ability to verify that recorded telemetry has not been altered after it is accepted for storage, and (ii) the ability to ensure that submitted telemetry records follow an enforceable sequence so that out-of-order or forged continuations are rejected at submission time.

Therefore, there is a need for a practical, WF tailored framework that enables tamper-evident recording and integrity verification of operational telemetry transfer *within* the WF boundary, while preserving audit-relevant context (e.g., turbine identity and time bounds) and remaining feasible under realistic SCADA data-rate and deployment constraints [11, 14]. This thesis also evaluates the same integrity mechanism in a multi-farm setting, where telemetry is submitted from WFs to a shared control-center ledger.

1.2.1 Research Objectives

The main goal of this thesis is to strengthen the integrity and auditability of WT telemetry delivery when measurements are exchanged over networked infrastructures. In particular, telemetry can be exposed to modification, or reordering *before* it becomes part of an immutable ledger block and is used for monitoring or stored for audit. To address this, the thesis proposes recording telemetry together with an on-chain hash link that ties each submitted record to the previously accepted record of the same turbine. The work is developed in two frameworks. Framework I targets *in-farm* WT-to-control-room transfer and uses fixed-duration windows to reduce on-chain load under multi-turbine submissions. Framework II evaluates a second framework in a *multi-farm* setting, where WFs act as peers and control-center servers provide the ordering service under a BFT configuration. In Framework II, telemetry records are submitted directly together with the same on-chain hash-link mechanism. Compared to Framework I, Framework II changes the node abstraction (WF-level peers instead of WT-level peers) and the ordering assumption by using a BFT ordering configuration at the control center. It also removes window formation and submits records directly, since the workload is lower than the in-farm multi-turbine setting.

Accordingly, the specific research objectives for Framework I are to:

(1) **Model the in-farm telemetry context and data path:** Establish the WF telemetry scope inside the WF boundary and describe the WT-to-control-room transfer path using the SCADA setting and IEC 61400-25 context as system grounding.

(2) **Define a windowed telemetry submission method:** Represent telemetry as fixed-duration windows of length Δ and submit one ledger transaction per turbine per window in order to reduce per-sample on-chain load while preserving audit-relevant metadata (turbine identity and window time bounds).

(3) **Enforce per-turbine stream continuity before commit:** Construct a per-turbine hash link across successive windows (e.g., using `window_hash` and `previous_hash`) and enforce correct linkage during endorsement so that invalid continuations are rejected before ordering and commit.

(4) **Implement the end-to-end workflow on a permissioned ledger:** Implement the framework on

Hyperledger Fabric, including the turbine-side client submission pipeline and application chaincode that stores accepted window records and performs endorsement-time continuity checks.

(5) **Quantify performance overhead under multi-turbine workload:** Measure end-to-end commit latency and sustained throughput under concurrent turbine submissions, and study sensitivity to practical framework parameters including orderer batching configuration, window duration Δ, and application payload size.

(6) **Validate integrity behavior through targeted experiments:** Demonstrate (i) post-commit tamper evidence by showing that disk-level modification of a peer's stored blocks yields detectable inconsistencies, and (ii) pre-commit rejection of invalid stream continuation by showing endorsement-time failure when the submitted hash link does not match the last committed state of the same turbine.

The following additional objectives are addressed in Framework II:

(1) **Adapt the framework to a multi-farm topology:** Configure a network where WFs act as peer nodes and control-center servers provide the ordering service under a BFT configuration.

(2) **Record direct telemetry records with continuity links:** Store raw telemetry records together with an on-chain hash link for per-turbine continuity.

(3) **Validate integrity behavior in the multi-farm setting:** Examine whether forged or out-of-order continuations are rejected and whether delayed records can be accepted through correct sequential replay.

In this thesis, *integrity* is used in the operational sense of *tamper-evident recording* and *per-turbine continuity* for accepted telemetry windows. For Framework II, the same integrity meaning applies to accepted telemetry *records* submitted at the WF level, with continuity enforced per turbine identifier. The proposed framework does not attempt to validate the physical correctness of measurements if an attacker can submit correctly linked but semantically false values, and it does not claim availability guarantees against denial-of-service or communication outages.

1.3 Thesis Contributions

The contributions of this thesis are summarized as follows. First, it proposes and formalizes a WF-tailored workflow for tamper-evident recording of in-farm turbine telemetry on a permissioned ledger using fixed-duration measurement windows. It also extends the same integrity concept to a multi-farm setting, where WF peers submit telemetry to a shared control-center blockchain network under a BFT ordering configuration.

Second, it extends ledger-level tamper evidence with an application-level continuity mechanism by introducing a per-turbine hash chain across windows that is enforced during endorsement before commit, both across fixed-duration windows in Framework I and across direct telemetry records in Framework II. Third, it implements the proposed design on Hyperledger Fabric v2.5.4 in multi-organization frameworks driven by an EMT-based WF benchmark and a trace-replay pipeline for Framework I, and by a MATPOWER-based multi-farm telemetry dataset for Framework II. Finally, it provides an experimental evaluation that quantifies performance overhead in Framework I through throughput and end-to-end commit latency, explores sensitivity to practical parameters (orderer batching, window duration, and payload size), and validates integrity behavior in both frameworks through controlled post-commit block-store tampering, endorsement-time rejection of invalid stream continuation, and continuity checks in the multi-farm setting.

Concretely, this thesis contributes:

- a permissioned-ledger-based telemetry transfer design for *in-farm* WT-to-control-room monitoring that records telemetry as fixed-duration windows and preserves audit-relevant context;
- a per-turbine hash-chain mechanism across windows, together with chaincode logic that enforces continuity during endorsement before commit to prevent invalid stream continuation from being committed;
- a working Hyperledger Fabric framework with supporting client-side ingestion, chaincode, and monitoring setup, driven by simulated multi-turbine telemetry traces from an EMT-based WF benchmark; and
- an experimental evaluation that (i) measures throughput and end-to-end commit latency under concurrent turbine submissions and (ii) validates integrity through post-commit tamper-evidence

checks and pre-commit continuity enforcement experiments, and (iii) examines the same continuity mechanism in a multi-farm setting.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews related work and motivates the research gap addressed in this thesis. Chapter 3 establishes the WF telemetry and system context used throughout the work. Chapter 4 provides the required blockchain background and motivates the selection of a permissioned platform, with emphasis on Hyperledger Fabric and where integrity checks can be enforced in its transaction flow. Chapter 5 presents the proposed methodology and framework, including telemetry ingestion, window formation (Framework I) and direct record submission (Framework II), per-turbine hash chaining, and the chaincode data model. Chapter 6 reports the experimental setup and evaluation results for both performance and integrity under controlled adversarial conditions. Finally, Chapter 7 concludes the thesis and outlines directions for future work.

Chapter 2

Literature Review

2.1 WF Security

WFs are cyber-physical energy systems in which power-generation assets are distributed across large geographic sites and interconnected through electrical and supervisory infrastructures [15, 16]. Modern WFs rely on Supervisory Control and Data Acquisition (SCADA) systems and Industrial Internet of Things (IIoT) devices to monitor turbine performance, collect sensor data, and support operational decision-making [11, 12]. Within the WF boundary, SCADA provides the operational monitoring and supervisory control layer above turbine-local controllers, and turbine-side devices expose measurements to supervisory applications through structured information models and communication services, as standardized in IEC 61400-25 [17, 11]. As a result, the end-to-end telemetry transfer path spans turbine-side acquisition, communication links, and control-room SCADA services, and integrity can be impacted at any stage of this pipeline [14].

As telemetry exchange increasingly occurs over networked infrastructures, concerns related to data integrity, security, and trust have increased significantly in WFs. Because cyber and physical layers are interdependent, disruption of communication links can directly reduce the observability and controllability of physical components, amplifying integrity and trust concerns in operational telemetry transfer [18]. OT environments are exposed to integrity and availability risks that can affect telemetry transfer, including unauthorized modification or injection of measurements, as well as omission (loss) or re-ordering of records due to failures or adversarial actions in communication

and storage components [14]. These concerns are consistent with broader threat reporting in the energy sector, where 67% of organizations in the energy, oil and gas, and utilities sector reported ransomware impacts in 2024 [19]. Representative disruption incidents have also been reported in wind operations: in March 2019, a denial-of-service (DoS) incident affected communication between a control center and wind generation sites in Utah, U.S., and caused unexpected device reboots after a firewall vulnerability was exploited [20]; and in February 2022, the KA-SAT/Viasat incident disrupted remote monitoring access to about 5,800 WTs in Germany [21].

Beyond external threats, existing centralized data management can expose WFs to single points of failure [14], unauthorized data modification, and limited transparency across operational stakeholders [22]. Related single-point-of-failure risk is also noted in distributed energy control architectures that rely on a central controller [23]. These characteristics motivate integrity mechanisms that remain feasible under realistic SCADA data-rate and deployment constraints [14] and support auditability for operational telemetry used in condition monitoring, performance supervision, and post-event diagnosis [13].

2.2 Blockchain for Integrity and Auditability

More generally, blockchain has been used to protect the integrity and auditability of logs, records, and evidence histories in distributed systems. In [24], a permissioned-blockchain-based logging infrastructure is proposed for preserving the integrity of generated log records and supporting log auditing. The system stores non-repudiable proofs of existence for log records and is implemented on Exonum. In [25], a blockchain-based audit-log architecture is proposed to secure enterprise audit logs that record changes made to data in business systems. The design is implemented on a custom-built blockchain based on Practical Byzantine Fault Tolerance (PBFT). In [26], a tamper-proof distributed logging system is presented for protecting distributed log files in byzantine settings. The implementation is built on Hyperledger Fabric. Blockchain has also been used to secure audit trails, provenance records, and digital evidence. In [27], a blockchain-based audit trail mechanism is designed and implemented for enterprise audit trails. The prototype uses a permissioned Quorum network and smart contracts. In [28], a blockchain-based platform is designed and

implemented for provenance tracking of electronic healthcare records exchanged across multiple healthcare facilities. The solution uses Hyperledger Fabric and manages provenance according to the W3C PROV standard. In [29], a blockchain cloud forensic logging framework is designed and implemented to preserve the trustworthiness and integrity of digital log evidence in a cloud ecosystem. The framework uses Hyperledger Fabric and maintains a chain of custody for log evidence. Taken together, these studies show that blockchain has been used for security purposes in systems where tamper resistance, verifiable history, and auditability are central requirements. The main uses include secure logging, audit trails, provenance tracking, and digital-evidence preservation. However, these works focus on generic computing or cloud-forensics settings rather than the end-to-end integrity of continuous WF SCADA telemetry considered in this thesis.

2.3 Blockchain in Energy Streams

Blockchain-based mechanisms have been increasingly explored as a trust-enabling layer in generic energy systems to support decentralized coordination and tamper-evident record keeping in networked infrastructures [30]. In the smart-grid cybersecurity domain, [31] surveys blockchain-based approaches with an emphasis on permissioned blockchains and smart contracts and discusses major scalability considerations. Blockchain has also been studied for renewable-energy monitoring and event-level control to improve tamper resistance and resilient information exchange for distributed energy resources [32]. However, the scope in these settings is commonly event-driven rather than continuous high-rate operational telemetry [32, 33].

A related line of work considers blockchain-based energy ecosystems for cyberattack identification and classification. For example, Internet of Blockchain (IoBC) architectures using a Solana-based blockchain communication layer are presented in [34]; however, the monitoring resolution is low (15-minute intervals) and the design relies on a centralized database [34]. Overall, prior energy-stream studies commonly focus on events or aggregated measurements rather than continuous high-frequency operational streams [32, 33], leaving open practical questions on how to provide tamper-evident logging and auditability for operational telemetry under sustained streaming workloads.

2.4 Blockchain in Wind Energy

Within the wind-energy domain, wind-focused blockchain studies span several distinct application directions. A major stream emphasizes traceability and accountability across the lifecycle of physical assets. In [35], Ionita *et al.* presented a prototype in which each wind-turbine bolt is assigned a unique identifier and key lifecycle events (registration, fastening, inspection, replacement, and recycling) are recorded on an Ethereum-based blockchain using Solidity smart contracts. In [36], a permissioned blockchain is proposed for the wind-turbine supply chain, where batch-level QR codes serve as the field "access point" to shared ledger records (supplier documentation and subsequent service updates). Blockchain-based quality control has also been proposed for wind-turbine blade supply chains to maintain traceable quality records across stakeholders [37]. More recently, blockchain-based traceability has been proposed for end-of-life wind blade recycling to track wind blades and derived recycled materials while addressing transparency and data privacy, including alternatives that leverage zero knowledge proofs to keep sensitive attributes private while remaining verifiable [38].

A second direction combines blockchain with wind-specific data intelligence and coordination. In [39], a permissioned blockchain was used to support decentralized sharing and validation of model updates in a federated blade-icing estimation workflow, with experiments using a 5-second sampling rate. At the WF information-system level, blockchain has been explored for operation and maintenance (O&M) data management [40]. Across these wind-focused studies, the dominant emphasis remains on traceability, coordination, or relatively low-rate monitoring/workflow support, rather than end-to-end integrity for high-frequency turbine measurement flows under realistic operational constraints [35, 36, 37, 39, 40]. Consequently, despite substantial activity on wind-related blockchain applications, there remains a lack of wind-farm-tailored blockchain mechanisms specifically designed to enhance the integrity and auditability of in-farm turbine telemetry transfer.

2.5 Research Gap

The literature on WF security consistently identifies integrity of key measurement and command signals as a central risk in WF OT environments, and develops detection and mitigation strategies against integrity and availability attacks on communicated signals [41, 42, 43, 44, 45, 46, 47]. In parallel, energy-sector threat reporting and operational incidents underscore that disruption and compromise of telemetry and monitoring pathways remain practical concerns in modern energy and wind operations [19, 20, 21]. At the same time, centralized data management in OT settings can introduce single points of failure and limit transparency across stakeholders [14, 22, 23].

Blockchain-based approaches have been widely studied to enable tamper-evident recording and audit in energy systems [30, 31, 32, 34], but prior work commonly focuses on events or aggregated measurements rather than continuous high-frequency operational streams [32, 33]. Wind-energy blockchain studies predominantly emphasize asset traceability and cross-organizational lifecycle logging [35, 36, 37, 38] or workflow support and relatively low-rate monitoring [39, 40], rather than end-to-end integrity for turbine measurement flows under realistic operational constraints [35, 36, 37, 39, 40].

Taken together, these findings motivate a practical, wind-farm-tailored integrity framework for operational telemetry transfer, both within the WF boundary and in a broader multi-farm WF-to-control-center setting [11, 14]. In particular, wind-farm telemetry integrity requires (i) the ability to verify that recorded telemetry has not been altered after it is accepted for storage, and (ii) the ability to ensure that submitted telemetry records follow an enforceable sequence so that out-of-order or forged continuations are rejected at submission time [14]. The above gap motivates mechanisms that provide tamper-evident recording and enforceable per-turbine continuity for accepted telemetry records, while remaining feasible under practical SCADA data-rate and deployment constraints [14].

Chapter 3

Wind Farm Telemetry and System Context

This chapter describes the WF operating context and the telemetry pathway that the proposed integrity framework targets. The purpose is to (i) define the physical and cyber layers that generate and carry WT measurements, (ii) clarify the supervisory monitoring and control architecture (SCADA and related services), and (iii) characterize typical data rates and reporting conventions that shape practical design constraints. Framework I, the focus is on *in-farm* telemetry transfer between the WT layer and the WF control room; wide-area/grid interfaces are discussed only to establish context. Framework II additionally uses a transmission-system benchmark to generate WF-to-control-center telemetry at selected grid buses, which is introduced in this chapter for completeness.

3.1 WF Physical Layer

A WF consists of multiple WTs distributed over a large geographic area and interconnected through both electrical infrastructure (for power collection and delivery) and supervisory infrastructure (for monitoring and control) [15, 16]. At the turbine level, each WT converts wind energy to electrical power through a rotor–generator system and incorporates local control and protection functions to ensure safe and reliable operation under changing wind and grid conditions [48]. From an electrical perspective, turbine outputs are collected through the WF collector system and routed

to a plant substation, where voltage is stepped up and power is delivered to the transmission network at the point of interconnection (POI). In parallel with power collection, the WF supervisory infrastructure transfers WT measurements and supervisory control signals between turbines and the control room to support monitoring, alarms, diagnostics, and plant-level control actions [16].

3.1.1 Benchmark system used in this thesis

To ground the telemetry discussion in a realistic multi-turbine setting, this thesis uses the electromagnetic transient (EMT) WF benchmark shown in Fig. 3.1 [1]. The benchmark models a doubly-fed induction generator (DFIG)-based WF with 268 turbines, each rated at 1.5 MW (575 V, 60 Hz), organized into four clusters of 67 turbines. Turbines within each cluster are connected through a 34.5 kV collector feeder, and clusters interface through a three-winding transformer (500/34.5/34.5 kV) before connecting to the external power system at the POI.

A main WF controller (WFC) receives POI measurements (e.g., voltage and current) and supports plant-level regulation objectives such as reactive power control, voltage regulation, and power-factor control. For simulation efficiency, the benchmark uses different modeling granularities across clusters; however, the detailed portion contains explicit turbine-group subsystems that allow realistic telemetry generation at the WT level. In this thesis, the detailed turbine-group structure illustrated in Fig. 3.2 is used as the telemetry source model. For realism, spatial variability in wind speed across turbines is modeled using a Gaussian distribution.

3.1.2 Transmission-system benchmark used for Framework II

To evaluate WF-to-control-center telemetry delivery in Framework II, this thesis also uses the MATPOWER `case39` benchmark (New England 39-bus system) as a grid-level context. In this setting, four buses are treated as WF injection points, i.e., Buses 2, 8, 11, and 21. A time-series dataset is generated by combining load patterns for system demand with wind generation profiles, producing synchronized measurements for the four WF buses that are submitted to the blockchain network.

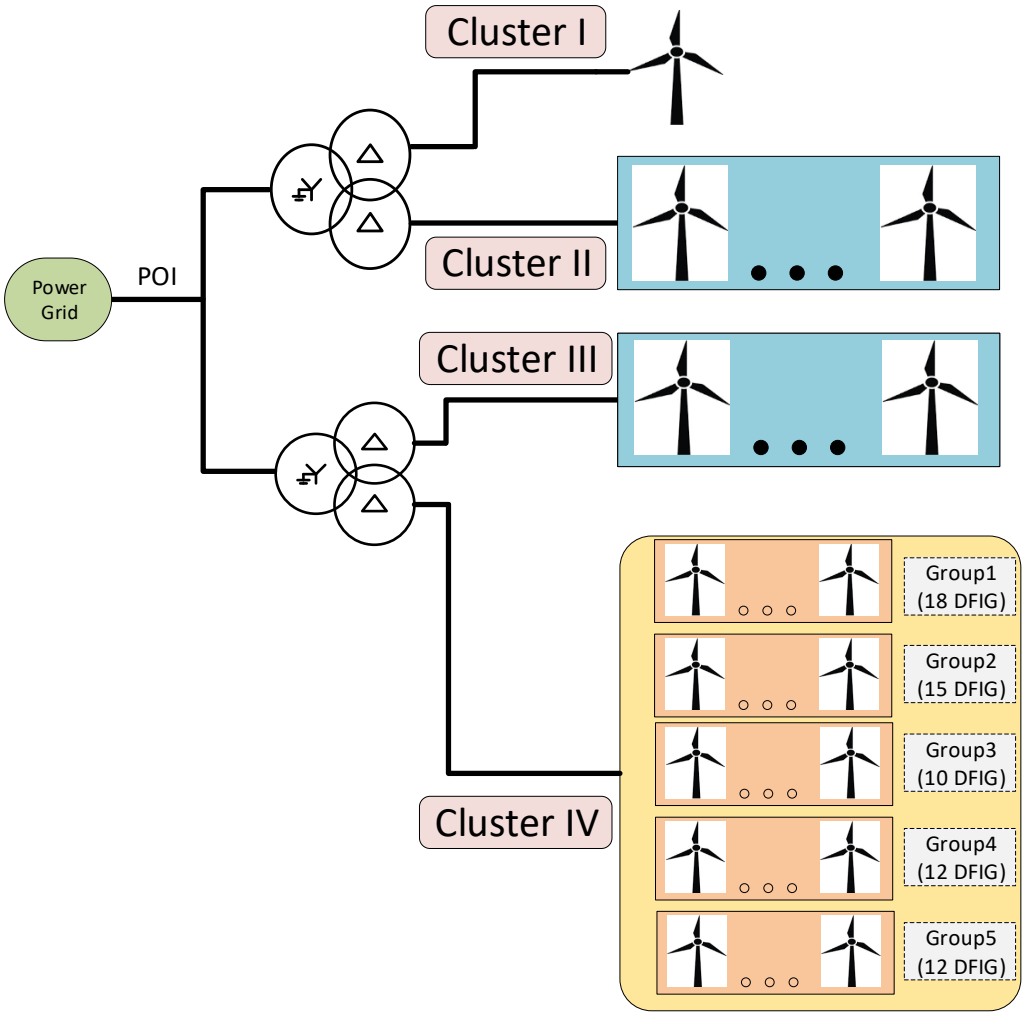

Figure 3.1: Single-mode detailed EMTP WF benchmark: four clusters (I–IV) connected at the POI. Cluster modeling granularity follows [1].

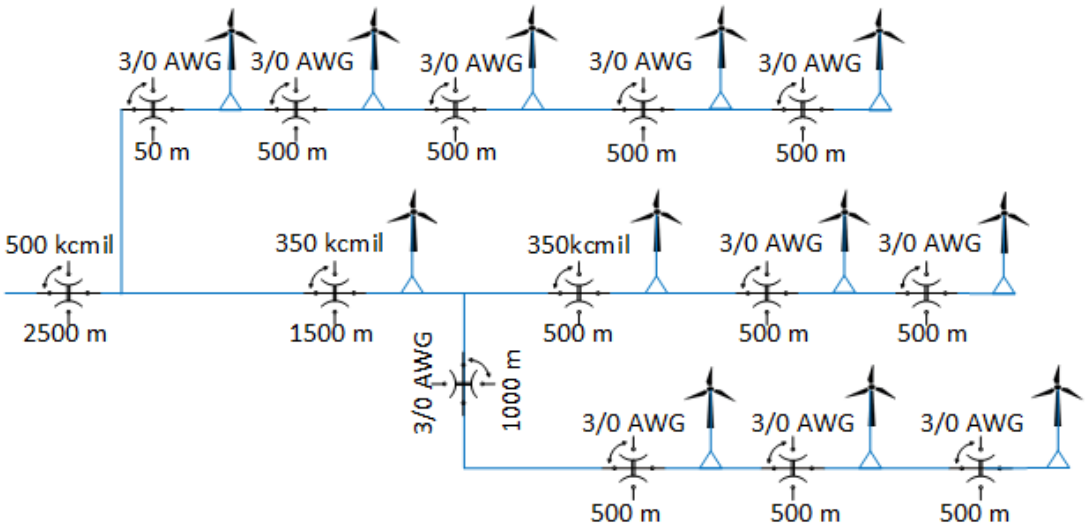

Figure 3.2: Detailed turbine-group subsystem used in this thesis: a 12-WT model inside Cluster IV of the EMTP benchmark.

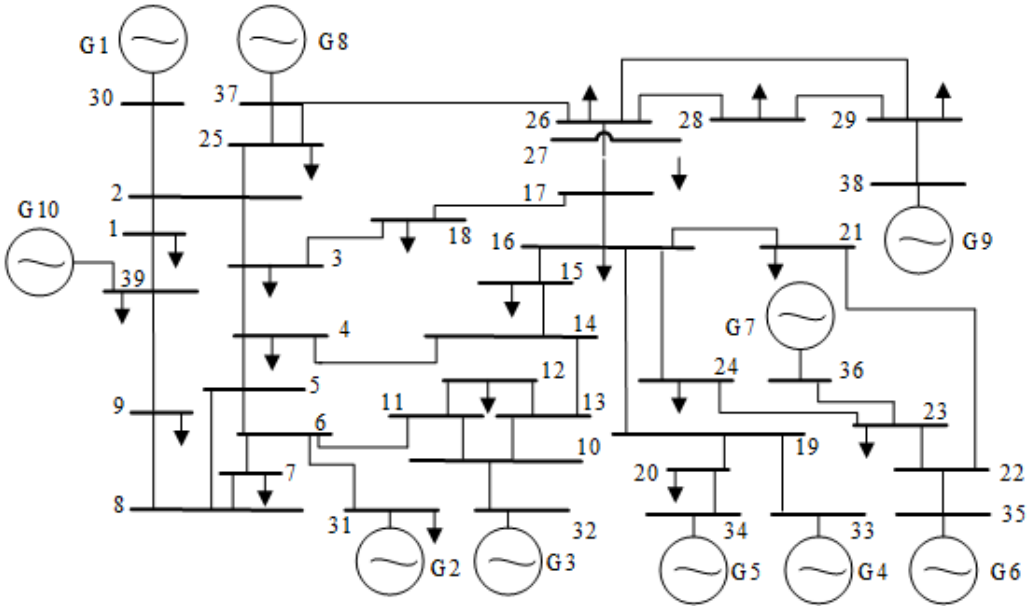

Figure 3.3: MATPOWER `case39` (New England 39-bus) benchmark used to generate WF bus telemetry for Framework II.

3.2 WF Data Transfer and SCADA Architecture

Operational monitoring and supervisory control in a WF are commonly implemented through a SCADA architecture layered above turbine-local controllers [17]. At the turbine level, embedded controllers and field devices continuously acquire measurements from electrical and mechanical subsystems (e.g., power conversion, rotor and drivetrain, pitch actuation). These measurements are exposed to supervisory applications through structured information models and communication services, as standardized in IEC 61400-25 [17]. Within the WF boundary, SCADA telemetry is time-stamped and used for real-time visualization (status, alarms, trends) and stored in historian services to support longer-horizon diagnostics, performance assessment, and analytics [2, 16].

3.2.1 Layered monitoring and control

A useful abstraction for WF monitoring and control separates functionality into three layers [2, 3, 49]:

(1) **WT level (field layer):** sensors and embedded turbine controllers measure and regulate local electrical/mechanical variables; measurements are collected by turbine-side devices (e.g., a WT control panel) that support local monitoring and control.

(2) **WF level (plant supervisory layer):** the control room aggregates turbine information for operator visualization, plant supervision, and execution of plant-level control objectives.

(3) **Grid level (wide-area layer):** aggregated WF information and commands are coordinated with upstream grid entities (e.g., dispatch and control centers) via wide-area networks.

3.2.2 Communication segments within the telemetry path

Fig. 3.4 summarizes a reference WF SCADA communication architecture and highlights the telemetry and control-command flow across practical network segments. In simplified form, four segments commonly appear in operational deployments: (i) turbine-internal sensing and control networks terminating at the turbine controller and WF local area network (LAN), (ii) the WF control-room LAN hosting supervisory services (e.g., SCADA servers and historians), (iii) equipment and

gateways at the POI substation, and (iv) a centralized operator/control-center LAN that may oversee multiple WFs. These segments provide a concrete system context for defining communication boundaries, trust relationships, and where telemetry can be observed, modified, delayed, or dropped during transfer. Framework I targets segments (i) and (ii) (WT-to-control-room), while Framework II targets segment (iv) (WF-to-control-center).

3.3 Data Transmission Characteristics

WF telemetry exists across multiple time scales. At the grid interface, reporting may incorporate phasor measurement units (PMUs) and wide-area communication, where reporting rates can reach up to 120 samples per second depending on configuration and use case [50]. Within the WF boundary, supervisory telemetry is typically generated at the turbine controller level and forwarded to the WF control room for visualization and archiving. Although operational reporting and long-term storage often rely on aggregated quantities (e.g., 10-minute averages), multiple studies note that turbine SCADA signals can also be acquired at higher temporal resolution (often on the order of 1 Hz) for monitoring and analytics [51, 52].

3.3.1 Telemetry signals used in this thesis

In the experimental workflow developed later in this thesis, WT telemetry is treated as a per-turbine supervisory time series sampled at 1 Hz [53, 54]. The exported measurements represent common supervisory variables used for monitoring and performance assessment: terminal voltage (V_t), current (I_t), and frequency (f), active and reactive power (P and Q), power factor (PF), rotor speed (ω_r), and pitch angle (β). These variables define the per-turbine telemetry stream used for integrity and performance experiments in Framework I. In Framework II, telemetry is defined at the WF level and reported at a 5-minute interval for WF-to-control-center delivery.

3.3.2 Typical sensor/telemetry rates

To contextualize the range of data rates that can exist in WT monitoring, Table 3.1 summarizes representative turbine-side measurement parameters and sampling rates reported in prior work. The

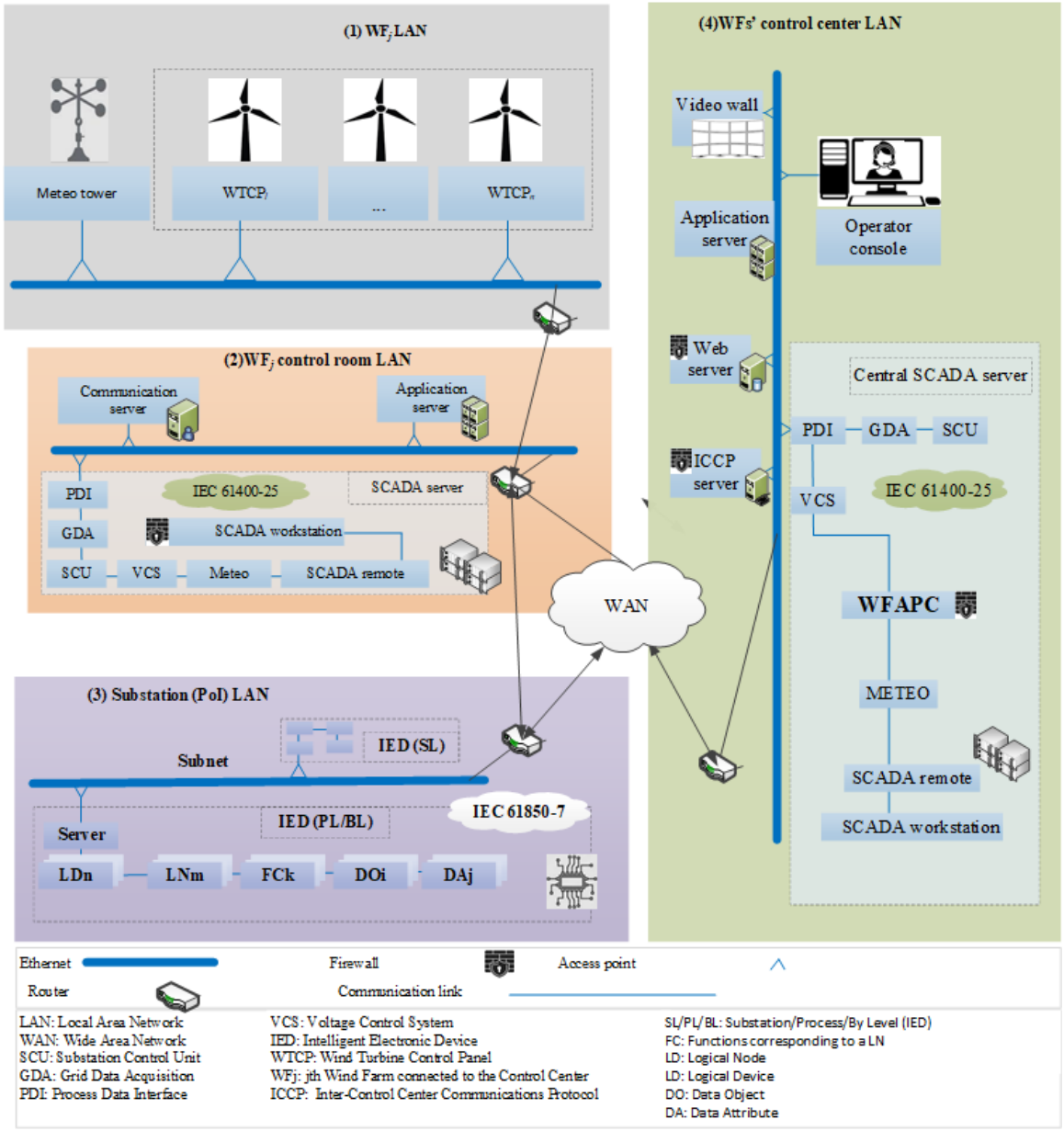

Figure 3.4: Reference WF SCADA communication architecture and layered data and command flow (adapted from [2, 3]).

Table 3.1: WF measurement parameters and transmission characteristics (sampling rate). Turbine-side sensor sampling rates follow Table 7 in [5]. For context on SCADA aggregation, [6] notes that conventional SCADA commonly uses 10-min averages, while high-frequency SCADA can be recorded at 100 Hz.

Parameter Name	**Sampling Rate**
Voltage	2000 Hz (0.5 ms)
Current	2000 Hz (0.5 ms)
Vibration	200 Hz (5 ms)
Pressure	100 Hz (10 ms)
Torque	50 Hz (20 ms)
Frequency	10 Hz (0.1 s)
Displacement	10 Hz (0.1 s)
Power	5 Hz (0.2 s)
Speed	3 Hz (0.333 s)
Pitch Angle	3 Hz (0.333 s)
Wind Speed	3 Hz (0.333 s)
Wind Direction	3 Hz (0.333 s)
Temperature	1 Hz (1 s)
Oil Level	1 Hz (1 s)
Power Factor	1 Hz (1 s)
Humidity	1 Hz (1 s)
Status Information (SI)	1 Hz (1 s)

table illustrates why system designs often avoid per-sample on-chain logging for high-rate sensors and instead rely on aggregation and windowing when a secure, auditable record is required at the supervisory layer. This motivation applies directly to Framework I, whereas Framework II operates at a 5-minute reporting interval and submits records directly without window formation.

3.4 Trust Zones and Attack Surface

The segmentation implied by Fig. 3.4 naturally defines practical trust zones that shape the telemetry attack surface. Turbine-internal networks and controllers form the first zone, where measurements originate and are first processed. The WF LAN and control-room LAN form a second zone, where telemetry is aggregated, visualized, and stored (e.g., SCADA servers and historians). The POI substation and any operator-side networks form additional zones that may introduce gateways, protocol translation, and external connectivity.

From a telemetry-integrity perspective, the most relevant exposure points are those where WT measurements can be modified, replayed, reordered, delayed, or selectively dropped before they are archived or used for control/diagnostics. These risks motivate integrity mechanisms that (i) provide

an auditable record of what was transmitted and when, and (ii) support continuity checks so that forged "stream continuation" or missing intervals become detectable. The subsequent chapters build on this context to define the integrity goal and to present a tamper-evident logging and verification approach tailored to in-farm WT telemetry transfer. A second evaluation is reported for WF-to-control-center delivery in Framework II under a multi-farm topology.

Chapter 4

Background on Blockchain and Platform Selection

This chapter introduces blockchain concepts needed for the remainder of the thesis and motivates the choice of a permissioned platform for wind-farm telemetry integrity. It first reviews core blockchain primitives and integrity properties, then contrasts permissionless and permissioned deployment models. Finally, it summarizes the key architectural components of Hyperledger Fabric and the transaction-processing pipeline that governs endorsement, ordering, validation, and commit.

4.1 Blockchain Fundamentals

Blockchain technology was popularized by the pseudonymous Satoshi Nakamoto as a decentralized mechanism for recording transactions without a centralized trusted authority [55]. At a high level, a blockchain is an append-only ledger replicated across multiple nodes. The ledger is organized as an ordered sequence of blocks, where each block bundles a set of transactions (records) together with metadata that links the block to the previous one via cryptographic hashes [22].

4.1.1 Blocks, hashes, and tamper-evidence

A block can be viewed as a *header* plus a *body*. The body contains the ordered transactions, and the header contains integrity-critical fields, including: (i) a reference to the previous block

(typically the previous block's hash), and (ii) a digest that commits to the block contents (often a Merkle root or an equivalent data hash) [22]. The block hash is computed by hashing the header fields. Because cryptographic hashes are deterministic and highly sensitive to input changes, even a single-bit modification to a past block changes its hash. Since that hash is embedded in the next block header, tampering breaks the chain linkage and can be detected when the chain is validated [22].

This property provides *tamper-evidence* (detectability of modification) after data are committed and replicated. In practice, a blockchain system additionally requires rules that determine who can submit transactions, which transactions are valid, and how all nodes converge on a consistent order.

4.1.2 Core roles in a blockchain system

While blockchain designs differ, most systems include: (i) *transactions* (submitted records), (ii) a *ledger* (the ordered history), (iii) *nodes* (participants maintaining and/or writing to the ledger), and (iv) a *validation/consensus process* that determines acceptance and ordering of transactions/blocks.

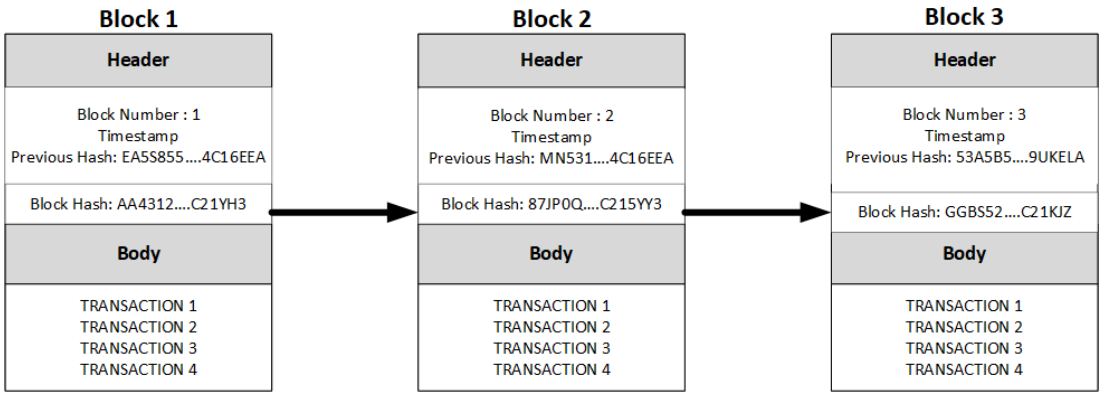

Figure 4.1: Conceptual blockchain structure (block header and body) illustrating hash-based linking between blocks.

4.2 Permissioned vs. Permissionless Networks and Rationale for Fabric

Blockchain networks are often categorized by their membership and trust model [22]. In *permissionless* (public) systems, participation is open, meaning that any user can join the network, submit transactions, and help maintain the ledger without prior approval. Because participants are not assumed to be known or trusted, these systems rely on consensus mechanisms designed for open and potentially adversarial environments, which often leads to higher resource cost and less predictable confirmation behavior [22]. In contrast, *permissioned* systems restrict participation to authenticated and authorized entities, and enforce explicit policies for identity, authorization, and data access. This enables tighter governance, greater privacy, and more predictable performance, which are important in enterprise and industrial settings [22] For this thesis, a permissioned blockchain is the more suitable choice. The targeted WF telemetry environment is not an open public network; instead, it consists of known operational entities such as turbine-side clients, WF servers, and control-center services. As a result, the system requires controlled membership, authenticated identities, policy-based authorization, and dependable transaction processing. Hyperledger Fabric was selected because it is an enterprise-grade permissioned blockchain platform that supports these requirements through managed identities, access-control policies, and modular transaction processing [4].

4.2.1 Fit to wind-farm telemetry and operational constraints

This thesis targets an operational wind-farm environment in which participants are known entities (e.g., turbine-side clients, on-site servers, and operator-side services), and telemetry may be sensitive. The platform therefore must support:

- **Controlled membership and authentication:** only authorized clients/nodes can submit and maintain ledger state;
- **Policy-based authorization:** explicit rules define which identities can read/write and which organizations must endorse updates;
- **Predictable commit behavior:** operational logging benefits from deterministic finality and stable confirmation times;

Table 4.1: Comparison of selected blockchain platforms (throughput and settlement characteristics).

Blockchain	TPS	Finality	Network type
Ethereum	$\sim$15	$\sim$12.8 min	Public
Tezos	$\sim$180	$\sim$16 s	Public
Bitcoin	$\sim$7	$\sim$60 min	Public
Hyperledger Fabric	$>$3500	$<$ 1 s	Permissioned
Corda	$\sim$13	$\sim$23–32 s	Permissioned
Quorum	$\sim$1650	$\sim$0.4–1.0 s	Permissioned

Sources: Ethereum consensus finality (2 epochs $\approx$ 12.8 min) [56]; Tezos TPS benchmark and deterministic finality [57, 58]; Bitcoin confirmation practice (6 conf. $\approx$ 1 hour) [59]; Hyperledger Fabric throughput/latency benchmark [60]; Corda benchmark [61]; Quorum performance evaluation [61].

- **Practical throughput:** the platform should tolerate sustained transaction rates under concurrent turbine submissions.

These needs align naturally with permissioned platforms. Hyperledger Fabric, in particular, provides an identity and access-control layer using PKI-issued X.509 certificates and organization-scoped membership service providers (MSPs). It also provides deterministic finality in the sense that validated and committed transactions are not reverted due to chain forks. Finally, prior evaluations report high throughput in permissioned deployments, making Fabric a practical choice for high-rate anchoring of telemetry windows in the targeted setting [4].

4.2.2 Indicative performance and settlement characteristics

Table 4.1 summarizes representative throughput and settlement/finality characteristics of selected platforms as reported in prior work. The values are indicative and can vary with configuration, workload, and network conditions; they are included to motivate the practical trade-off between open participation and predictable, high-throughput logging for operational telemetry.

4.3 Hyperledger Fabric Components

Hyperledger Fabric is a permissioned blockchain framework designed for multi-organization deployments with explicit identity and policy control. A Fabric deployment is partitioned into *organizations*, each representing an administrative trust domain. Organizations operate network nodes and manage identities through a public key infrastructure (PKI). Fabric also introduces *channels*,

which define logical partitions with their own membership and ledger, enabling data separation across application contexts.

Fig. 4.2 summarizes the primary Fabric roles, and the list below provides a concise definition of each component used throughout this thesis.

- **Organizations:** administrative domains that own identities and operate Fabric nodes.
- **Clients:** applications that create transaction proposals, collect endorsements, and submit transactions for ordering.
- **Peers:** nodes that host the channel ledger, execute chaincode during proposal simulation, validate ordered blocks, and commit valid transactions.
- **Ordering service (orderers):** nodes that establish a single total order of transactions and batch them into blocks distributed to peers.
- **PKI / CAs / MSP:** the identity and trust layer; certificates authenticate entities, and MSPs define which certificate authorities and roles are trusted within each organization.
- **Channels:** logical partitions with their own membership, policies, and ledger.
- **Ledger and world state:** each channel maintains an immutable transaction log (blockchain) and a world state storing the latest value for each key for efficient queries.
- **Chaincode and endorsement policy:** application logic (smart contract) executed by endorsing peers; an endorsement policy specifies which organizations must endorse a proposal for it to be considered valid [4].

4.4 Transaction Processing in Hyperledger Fabric

Fabric uses a transaction pipeline that separates execution (simulation) from ordering and from validation/commit. This structure supports policy-based endorsement while ensuring all peers converge on the same ordered history [4]. In this thesis, the pipeline is central because integrity checks for turbine telemetry submissions are enforced during endorsement (pre-commit), while the ledger's block structure provides tamper-evidence after commit.

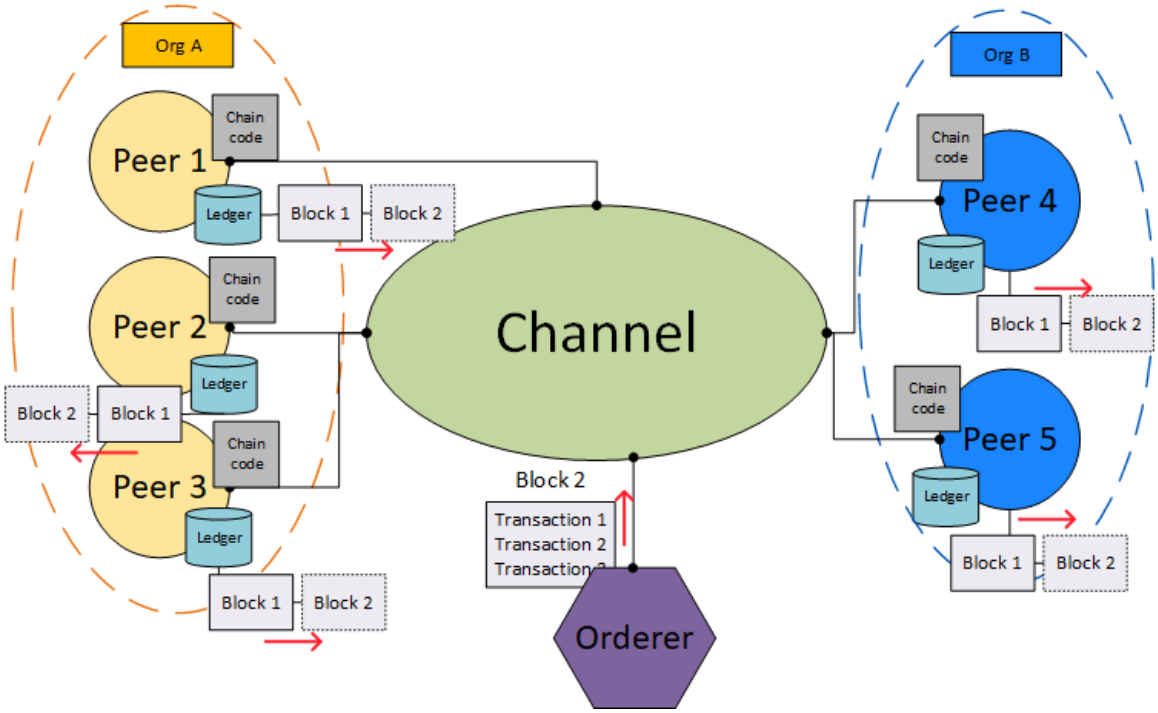

Figure 4.2: High-level Hyperledger Fabric architecture: clients submit proposals, peers simulate and endorse, the ordering service batches transactions into blocks, and peers validate and commit blocks on the channel ledger [4].

4.4.1 Endorse–order–validate–commit pipeline

At a high level, Fabric processes a transaction as follows:

(1) **Proposal creation (client):** a client constructs a transaction proposal that invokes a chaincode function with arguments.

(2) **Simulation and endorsement (endorsing peers):** selected peers execute the chaincode against their current world state to produce a read/write set (RW-set) and return signed endorsements.

(3) **Submission to ordering service (client):** the client assembles the endorsed transaction and submits it to the ordering service.

(4) **Ordering and block cutting (orderers):** the ordering service establishes a total order over transactions and packages them into blocks.

(5) **Validation and commit (peers):** peers validate each transaction against channel rules (including endorsement policy satisfaction and consistency checks) and then commit valid transactions

by appending the block to the ledger and updating the world state.

4.4.2 Where integrity checks can be enforced

Two points in the pipeline are especially relevant for this thesis: (i) **pre-commit enforcement during endorsement**, where chaincode logic can reject a proposal if application-level integrity conditions are not satisfied; and (ii) **post-commit tamper-evidence**, where the hash-linked block structure allows detection of ledger modification after commit through cross-peer consistency checks [22]. Chapter 5 (Methodology and Prototype) builds directly on these mechanisms to implement tamper-evident logging and stream-continuity enforcement for in-farm wind-turbine telemetry.

Chapter 5

Methodology and prototype

5.1 Framework I & II overview

This subsection summarizes the proposed methodology for tamper-evident transfer of WT telemetry from the turbine site to the WF control room using a permissioned blockchain (Framework I). The WF telemetry path is described in Section II, and the Hyperledger Fabric transaction model is described in Section III. Fig. 5.1 labels the main processing points with circled step numbers ①–⑤, which are referenced below. In the proposed framework, WT measurements are first delivered to a turbine-side industrial PC (IPC). The IPC acts as the client-side node that prepares blockchain submissions. The IPC groups incoming samples into fixed-duration windows of length Δ ①. Each turbine produces one window at a time, and each window is submitted as one blockchain transaction. Therefore, each turbine generates its own sequence of transactions over time. To ensure continuity of each turbine stream, each window includes integrity fields that link it to the previously accepted window of the same turbine. This creates a per-turbine hash chain across windows ②. The submission is then sent to the Hyperledger Fabric network on the application channel, i.e., `turbine-chan` ③. During endorsement, endorsing peers execute the chaincode (smart contract), i.e., `turbinecc`, and verify that the submitted link matches the latest committed state for that turbine. If the linkage is invalid, endorsement fails and the transaction cannot be committed. If it is valid, the client forwards the endorsed transaction to the ordering service, where transactions are batched into blocks ④. The ordering service distributes blocks to channel peers for validation and

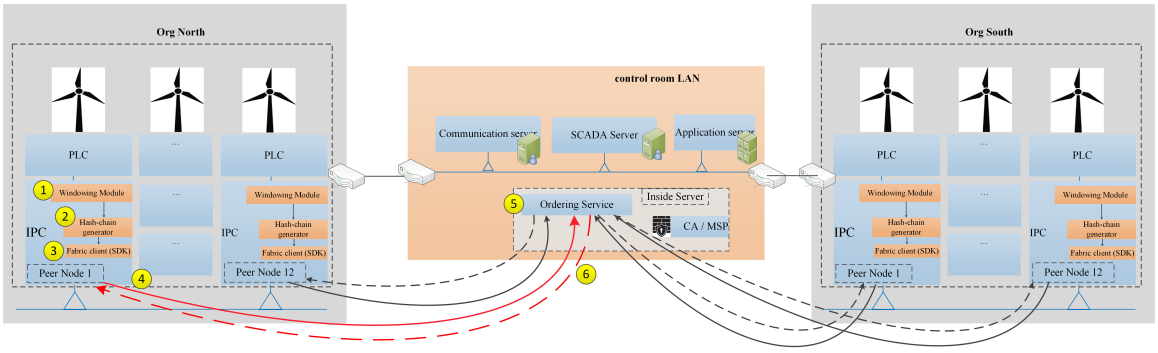

Figure 5.1: Framework deployment and data path in the implementation.

commit ⑤. After commit, the recorded window transactions can be queried from the control-room layer for monitoring and audit. Integrity is provided in two layers. Hyperledger Fabric provides tamper-evidence after commit through its block hash chain. The per-turbine hash chain enforces stream continuity by rejecting forged or out-of-order windows during endorsement before commit. A second prototype (Framework II) evaluates the same integrity mechanism for WF-to-control-center telemetry delivery under a multi-farm topology and BFT ordering, and is summarized later in this chapter.

Framework II extends the same integrity concept to a broader WF-to-control-center setting involving multiple WFs. In this framework, telemetry is generated at the WF level and submitted from WF-side clients to a shared control-center blockchain network rather than from turbine-side IPCs to a local WF ledger. Unlike Framework I, records in Framework II are submitted directly rather than grouped into fixed-duration windows; however, each record still carries a hash link to the previously accepted record for the same turbine identifier. This preserves per-turbine continuity across successive submissions while adapting the workflow to a multi-farm topology. The Fabric deployment in Framework II uses one peer per WF organization and a control-center ordering service under a BFT configuration. As in Framework I, integrity is enforced in two layers: Hyperledger Fabric provides tamper-evidence after commit, while the application-level hash chain prevents forged or out-of-order stream continuation before commit.

5.2 Framework I Implementation

This subsection summarizes the framework implementation of the proposed tamper-evident telemetry transfer framework in Framework I. The framework is implemented as three blocks.

First, *telemetry generation and replay* produces a repeatable input stream: an EMT-based WF simulator models realistic DFIG turbines and WF control and exports measurement traces, and a Trace Replay Service (TRS) replays the traces and streams them as samples.

Second, a *turbine-side IPC client pipeline* aggregates samples into fixed-duration windows and prepares one transaction per turbine per window, including the sequence number and a per-window hash-chain link.

Third, a *permissioned Hyperledger Fabric backend* provides authenticated submission, ordering, and commit on a single application channel (`turbine-chan`), and an application chaincode (`turbinecc`) stores accepted window records and enforces per-turbine hash-chain continuity during endorsement before commit.

The Hyperledger Fabric network is deployed with two organizations, OrgNorth and OrgSouth. The telemetry workload is derived from Cluster IV of the EMTP benchmark (Fig. 3.1); two 12-WT (DFIG) turbine groups (Fig. 3.2) are used, with one group assigned to each organization. Each turbine is represented by one peer node, so each organization hosts 12 peers (`peer0--peer11`) for a total of 24 peers. All peers participate in a single application channel, `turbine-chan`, which defines the shared ledger for the experiments. The ordering service uses Raft-based ordering (*etcdraft*) with four orderer nodes, and peers use *CouchDB* as the world-state database. The resulting topology is summarized in Table 5.1.

The following subsections describe: (1) telemetry trace generation and replay, (2) IPC-side window formation with per-turbine hash links, and (3) chaincode-side validation and storage with endorsement-time continuity checks.

5.3 Framework II Implementation

Framework II evaluates WF-to-control-center telemetry delivery using a separate workload and network topology. Telemetry is generated from the MATPOWER `case39` benchmark (New England 39-bus system) as described in Chapter 3. Four buses are treated as WF injection points (WF1–WF4). A time-series dataset is produced at a 5-minute reporting interval by (i) applying a normalized load pattern to the base `case39` bus loads, (ii) assigning wind generation profiles to the four WF buses, and (iii) running a power-flow snapshot for each time step to export synchronized WF-bus measurements for submission.

The Fabric deployment in Framework II includes four organizations, OrgWF1–OrgWF4, with one peer per organization. All peers participate in a single application channel, `wfchan`. The ordering service is hosted at the control-center side using four orderer nodes configured with a BFT ordering protocol. The application chaincode stores accepted WF-level records and applies the same hash-link continuity check across successive submissions.

Table 5.1: Hyperledger Fabric network topology in Framework I

Component	Count	Role
Organizations	2	OrgNorth, OrgSouth
Peers (OrgNorth)	12	`peer0--peer11`
Peers (OrgSouth)	12	`peer0--peer11`
Ordering nodes	4	Ordering service
Application channel	1	`turbine-chan`
World state DBs	24	CouchDB

.

Table 5.2: Hyperledger Fabric network topology in Framework II.

Component	Count	Role
Organizations (WFs)	4	OrgWF1, OrgWF2, OrgWF3, OrgWF4
Peers	4	One peer per WF (OrgWF1–OrgWF4)
Ordering nodes	4	Control-center ordering service (BFT)
Application channel	1	`wfchan`

5.4 WF data simulation and ingestion

WT telemetry is generated using the EMTP-based WF benchmark described in Section II. The simulator produces measurement traces for the selected turbines and exports them as input data

for the experiments. A TRS replays the input data and emits time-stamped samples per turbine, parsing the structured headers and converting values to numeric types to provide a consistent input across runs. The emitted samples are then delivered to the turbine side IPC pipeline in Fig. 5.1 for windowing, hash-chain construction, and submission to the ledger. For Framework II, WF-level telemetry is generated from the MATPOWER `case39`-based time-series dataset at a 5-minute interval for four WF buses. These records are submitted from WF-side clients to the permissioned ledger on channel `wfchan`, without window formation.

5.5 Windowing and per-window hash-chain construction (Framework I)

Submitting each raw sample as an on-chain transaction is not practical because it would generate a very large number of transactions per second and increase latency, bandwidth use, and ledger growth without proportional value for supervisory monitoring. Rapid ledger growth can also overwhelm storage on resource-constrained edge participants in cyber-physical energy deployments, reinforcing the need to avoid per-sample on-chain logging [62]. Therefore, the client-side pipeline uses a windowing module that packages telemetry into fixed-duration windows of length Δ, and submits one transaction per turbine per window.

In Framework I, windowing aggregates multiple 1 Hz samples into a fixed-duration record to reduce on-chain transaction load while preserving audit-relevant time bounds and sample count. Windowing provides two additional benefits. First, it standardizes the on-chain record format (time span and `sample_count`) for supervisory monitoring and auditing. Second, it defines the unit over which integrity is enforced via a per-window hash link. More generally, when the sampling period is smaller than the window length ($T_s < \Delta$), multiple samples can be included in one window, which reduces the number of on-chain transactions by approximately a factor of Δ/T_s.

For each turbine, each window record includes the turbine identifier, window start and end timestamps, sample count, and the selected measurements. To preserve continuity of each turbine stream across successive windows, a per-turbine hash chain is constructed. For each window, a `window_hash` is computed from the window content, and a `previous_hash` is included to reference the previously accepted window of the same turbine. This creates an explicit linkage

between consecutive windows, so forged or out of order windows can be detected when the linkage is broken.

5.6 Chaincode and ledger data model

The chaincode (smart contract), i.e., `turbinecc`, is deployed on channel `turbine-chan`. It stores accepted window records on the ledger and enforces per-turbine hash-chain continuity during endorsement before commit. In Framework II, the same logic is applied to WF-level records on channel `wfchan`, without window formation. Each accepted submission writes a window record that stores window metadata, measurement content, and the hash-chain fields. Window records are stored under deterministic keys derived from the turbine identifier and window time bounds to support time-ordered retrieval per turbine. In addition, for each turbine the ledger maintains a small head state that stores the latest accepted sequence and hash, which is used as the reference for validating the next submission. During endorsement, endorsing peers execute `turbinecc` to read the turbine head and compare the submitted `previous_hash` against the latest stored hash. If the values do not match, the chaincode returns an error and endorsement fails, so the transaction cannot be committed. If the linkage is valid, `turbinecc` writes the new window record and updates the turbine head to the new hash.

Chapter 6

Evaluation

This section evaluates the two proposed frameworks from two perspectives. First, we quantify performance overhead for anchoring windowed turbine telemetry on Hyperledger Fabric, using end-to-end commit latency and sustained throughput under concurrency, and by examining how window size (Δ) and payload size affect these metrics. Second, we validate integrity in two layers: (i) post-commit tamper evidence when a peer's on-disk ledger blocks are modified, and (ii) prevention of forged or out-of-order turbine windows by enforcing a per-turbine hash link during endorsement before commit. Evaluations #1–#7 are carried out on Framework I, which targets in-farm WT-to-control-room telemetry transfer, whereas Evaluations #8–#9 are carried out on Framework II, which extends the same integrity concept to a multi-farm WF-to-control-center setting.

6.1 Experimental setup

The setup described first in this section applies to Framework I (Evaluations #1–#7). All experiments were run on a Hyperledger Fabric v2.5.4 network on channel `turbine-chan`, with TLS enabled and CouchDB as the world-state database. The feeder replays turbine telemetry at a 1 Hz per-turbine rate ($T_s = 1$ s) and submits one window transaction per turbine per interval. We use a baseline window length of $\Delta = 1$ s, so each on-chain window covers one second and carries a single

sample (`sample_count`=1). Evaluation #3 varies Δ, and Evaluation #5 varies T_s to study higher-rate sampling with window aggregation. We evaluate two orderer batching configurations, summarized in Table 6.1. Config L represents a low-latency setting by using a shorter `BatchTimeout`, which reduces the waiting time before a block is cut when transactions arrive one-at-a-time. Config T represents a throughput-oriented setting by using a longer `BatchTimeout` and a larger `MaxMessageCount`, allowing more transactions to accumulate before block cutting at the cost of higher batching delay. These parameters are part of Hyperledger Fabric's ordering-service configuration.

Commit latency was measured at the client using the Fabric CLI synchronous invoke option `--waitForEvent`; in this mode, the client returns only after receiving a `VALID` commit event. In this thesis, latency therefore refers to the *end-to-end commit latency* observed by the submitting client, measured from the time the transaction is submitted until the corresponding `VALID` commit event is received. This measurement captures the full delay along the Fabric transaction path as experienced by the turbine-side IPC, rather than only one internal stage. As a result, the reported latency reflects the cumulative delay of endorsement related processing, submission to the ordering service, block cutting and batching, peer-side validation and commit, and delivery of the commit event back to the client. We report this elapsed time as the practical end-to-end delay of the proposed workflow.

For monitoring and inspection, Hyperledger Explorer was connected to the network to visualize blocks, transactions, and chaincode activity during the experiments (Fig. 6.1). Table 6.2 summarizes the software and host environment for reproducibility.

Framework II (Evaluations #8–#9) uses a separate deployment on channel `wfchan`, where four WF organizations each host one peer and a control-center-side ordering service operates under a BFT configuration. In this setting, telemetry records are generated from the MATPOWER case39-based multi-farm dataset at a 5-minute reporting interval and are submitted directly, without window formation, from WF-side clients to the shared control-center ledger.

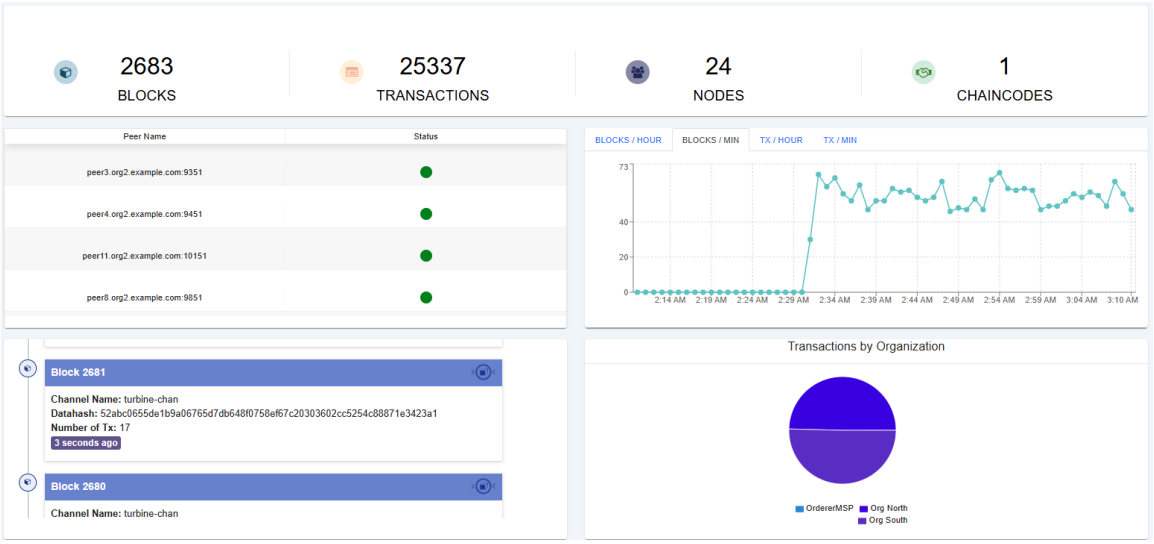

Figure 6.1: Hyperledger Explorer dashboard used to monitor the Fabric network during experiments (channel status, blocks, transactions, and chaincode activity).

Table 6.1: Orderer batching configurations used in evaluation.

Parameter	Config L	Config T
`BatchTimeout`	0.5 s	2 s
`BatchSize.MaxMessageCount`	50	200
`BatchSize.PreferredMaxBytes`	1 MB	2 MB
`BatchSize.AbsoluteMaxBytes`	10 MB	10 MB

Table 6.2: Evaluation environment (software and host configuration).

Item	Value
Blockchain platform	Hyperledger Fabric v2.5.4
Channel	`turbine-chan`
Chaincode	`turbinecc`
World state DB	CouchDB
TLS	Enabled (peer/orderer communication)
Orderer batching	Two configs: Config L (`BatchTimeout`=0.5 s) and Config T (`BatchTimeout`=2 s)
Client submission mode	Synchronous invoke with `--waitForEvent` (wait for `VALID` commit event)
Monitoring / inspection	Hyperledger Explorer (blocks, transactions, chaincode activity)
Host OS	Parrot Security 6.4 (Debian-based), x86_64
CPU	Intel Core i9-12900H (14 cores / 20 threads)
RAM	32 GB
Docker Engine	28.3.2
Fabric runtime image	`hyperledger/fabric-peer:2.5.4`

Table 6.3: Evaluation #1 baseline commit latency comparison (200 sequential windows, T=1).

Metric	Config L	Config T
Valid / Failures	200 / 0	200 / 0
Blocks created (Tx/Block)	200 (1.00)	200 (1.00)
Mean latency (ms)	846.108	2365.486
p95 / p99 (ms)	899.102 / 936.758	2423.167 / 2823.929

6.2 Performance evaluation

Evaluation #1 (Framework I) Baseline single-turbine commit latency

This experiment measures baseline end-to-end commit latency under a single-turbine workload and compares the impact of the two batching configurations before introducing concurrent turbines in later evaluations. One turbine client (T=1) submits N=200 windows sequentially (one in-flight transaction at a time). The window length is set to Δ=1 s.

Table 6.3 reports the baseline latency statistics for Config L and Config T. In this submission pattern, each transaction is committed before the next one is issued, so blocks contain approximately one transaction in both configurations. The difference between Config L and Config T is mainly driven by the orderer batching timeout: Config L uses a shorter `BatchTimeout` (0.5 s), while Config T uses a longer `BatchTimeout` (2 s), which increases the waiting time for block cutting when transactions arrive one-at-a-time.

Evaluation #2 (Framework I) Concurrent workload throughput and commit latency

This experiment measures sustained throughput and end-to-end commit latency under a concurrent workload and compares Config L and Config T under load. The feeder runs T=24 turbine clients in parallel, with a 30 s warmup to exclude startup transients so measurements reflect steady state behavior; followed by a 120 s measurement interval for computing metrics. Each turbine submits Δ=1 s windows and waits for a `VALID` commit event before issuing the next window. We report sustained throughput (VALID TPS), commit-latency percentiles (p50/p95/p99), and average transactions per block to relate results to orderer batching behavior.

Table 6.4 summarizes the concurrent workload results. Compared to the single-turbine baseline (Evaluation #1), concurrency increases both sustained throughput and tail latency because multiple

Table 6.4: Evaluation #2 concurrent workload results (T=24, warmup=30 s, measure=120 s, Δ=1 s).

Metric	Config L	Config T
Valid / Failures	1322 / 0	871 / 0
VALID TPS	11.017	7.258
Blocks (Tx/Block)	188 (7.03)	46 (18.94)
Mean latency (ms)	1506.822	2740.656
p95 / p99 (ms)	3222.109 / 5258.257	2895.290 / 5877.558

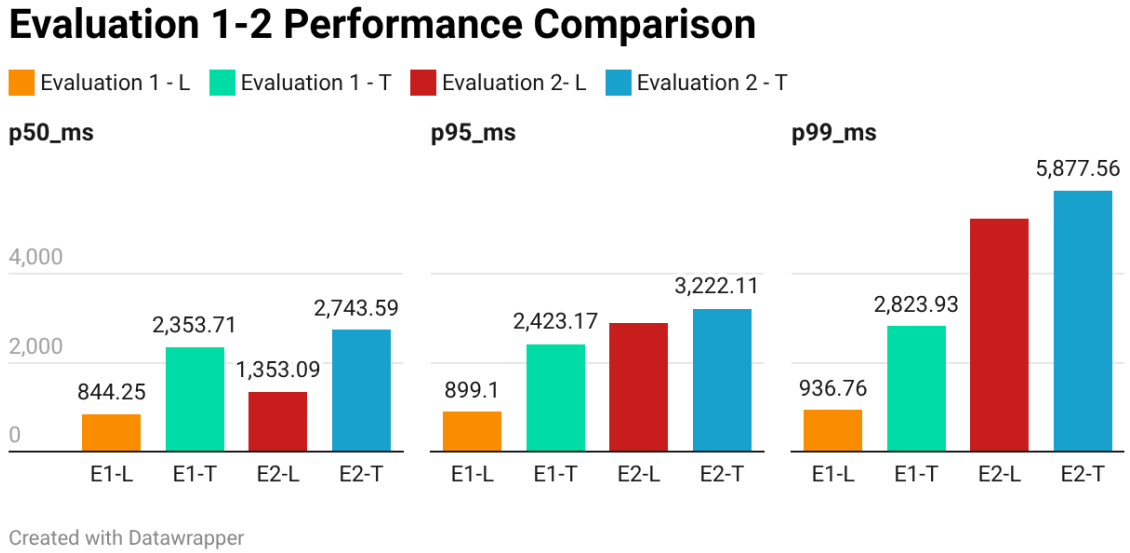

Figure 6.2: Commit-latency percentile summary for Evaluation #1 (baseline, sequential commits) and Evaluation #2 (24-turbine concurrent load) under the two orderer batching configurations (Config L and Config T). Bars report p50, p95, and p99 end-to-end commit latency measured at the client using synchronous invoke (`--waitForEvent`) over `VALID` transactions only.

turbines contend for endorsement, ordering, and commit resources. Config T produces larger blocks on average (higher Tx/Block) because its longer `BatchTimeout` and larger `MaxMessageCount` (Table 6.1) allow more transactions to accumulate before block cutting occurs, while Config L cuts blocks more frequently. Because each turbine waits for a `VALID` commit event before submitting the next window, the sustained VALID TPS is limited by the per-window commit time under the available parallelism. Fig. 6.2 compares p50/p95/p99 commit-latency for the baseline sequential case (Evaluation #1) and the 24-turbine concurrent case (Evaluation #2) under Config L and Config T.

Table 6.5: Evaluation #3 window duration sweep (Config L, T=24, warmup 60 s, measure 180 s).

Δ (s)	VALID TPS	Tx/Block	mean (ms)	p95 / p99 (ms)
1.0	12.883	9.166	1352.902	2293.433 / 4954.933
2.0	12.772	9.422	1341.230	1597.092 / 4612.031

Evaluation #3 (Framework I) Window duration (Δ) sweep under concurrent load

Evaluation #3 quantifies the sensitivity of throughput and commit latency to the application window duration Δ while keeping the concurrent workload fixed. All runs use Config L (`BatchTimeout`=0.5 s, `MaxMessageCount`=50) so transactions spend less time waiting for block cutting than in larger-timeout settings, which reduces block-cut waiting compared to larger timeout settings, so the observed differences primarily reflect commit pipeline behavior under the concurrent workload.

We run T=24 turbines concurrently (warmup 60 s, measure 180 s) and sweep $\Delta \in \{1, 2\}$ s via `FEEDER_DELTA_S`. We keep the sensing/replay period fixed at $T_s = 1$ s (1 Hz); therefore each window spans Δ seconds and contains `sample_count` $= \Delta / T_s$ samples (e.g., 2 samples when $\Delta = 2$ s). Clients submit the next window immediately after receiving a `VALID` commit event (commit-limited loop), so the achieved transaction rate is governed by the commit pipeline rather than by Δ. All transactions committed successfully (0 failures). As a consistency check, the total number of blocks observed on the ledger across the two runs (height 5734→6479, i.e., 745 new blocks) matches the sum of the per-run `blocks_created` reported by the feeder logs.

Table 6.5 shows that across Δ = 1–2 s, throughput remains in the same range (about 11–13 VALID TPS) and mean commit latency remains around 1.34–1.46 s, indicating no strong performance degradation within this window-duration range under the tested concurrent load.

Evaluation #4 (Framework I) Payload size impact

This experiment measures how increasing the application payload size affects throughput, commit latency, and estimated ledger growth under a fixed concurrent workload. We use the same concurrent topology as Evaluation #2 (T=24 turbines), but run longer trials (warmup 60 s, measure 180 s) and repeat each setting three times for stability. We keep the orderer in Config L (`BatchTimeout`=0.5 s, `MaxMessageCount`=50) so blocks are cut more frequently and results are less dominated by batching delay. The window length is fixed to Δ=1 s (`FEEDER_DELTA_S`=1)

Table 6.6: Evaluation #4 payload size sweep under concurrent load (Config L, T=24, Δ=1 s, 3 repeats; values averaged across repeats).

Signals S	**VALID TPS**	**p95 (ms)**	**avg tx bytes (est.)**	**growth (MB/h) (est.)**
8	11.328	2179.0	249.0	10.154
16	12.189	2000.3	321.0	14.085
32	11.946	1743.1	495.0	21.288

so each transaction anchors one window, and we sweep the number of signals per sample $S \in \{8, 16, 32\}$.

Across all nine runs, all transactions committed successfully (0 failures). Table 6.6 shows that throughput remains around 11–12 VALID TPS across the sweep and p95 latency stays in the same order of magnitude, indicating that payload sizes in this range do not materially change sustained performance under the tested load. As expected, increasing S increases the estimated application payload size and the corresponding estimated ledger growth rate (approximately 10.2→21.3 MB/h from S=8 to S=32). The reported avg tx bytes and growth rates are application-level estimates based on serialized argument length and exclude Fabric envelope/protobuf overhead, signatures, and block metadata.

Evaluation #5 (Framework I) Higher-rate sampling via per-window aggregation (k sweep)

This experiment evaluates the effect of higher-rate sampling when multiple samples are aggregated into a fixed-duration window before submission. We fix the window length to $\Delta = 1$ s and vary the number of samples carried in each window, denoted by k (`sample_count`). With Δ fixed, k corresponds to an effective sampling period $T_s = \Delta/k$ and sampling rate $f_s = k/\Delta$ Hz. We sweep $k \in \{1, 5, 10, 20, 100\}$, i.e., $f_s \in \{1, 5, 10, 20, 100\}$ Hz, while keeping the concurrent workload fixed (T=24, warmup 60 s, measure 180 s) and using the same submission mode described in the experimental setup.

The key point of this evaluation is that increasing k increases the sensing rate carried by each committed window *without* increasing the on-chain transaction rate, since one window transaction is still submitted per turbine per second. Therefore, TPS and commit latency capture how well the system tolerates larger per-window payloads, while the effective sample throughput increases as samples/s $= k \times$ VALID TPS.

Table 6.7: Evaluation #5 sampling-rate sweep using per-window aggregation (Config L, T=24, warmup 60 s, measure 180 s, Δ=1 s).

k (Hz)	T_s (s)	VALID TPS	samples/s	mean (ms)	p95/p99 (ms)
1	1.00	11.856	11.856	1439.086	1657.923 / 4822.721
5	0.20	11.650	58.250	1445.980	2222.824 / 4686.071
10	0.10	11.844	118.440	1412.510	1642.593 / 4643.150
20	0.05	11.639	232.780	1457.729	2043.848 / 4834.540
100	0.01	11.856	1185.600	1417.974	1664.797 / 4669.062

Table 6.8: Evaluation #5 sampling-rate sweep using per-window aggregation (Config T, T=24, warmup 60 s, measure 180 s, Δ=1 s).

k (Hz)	T_s (s)	VALID TPS	samples/s	mean (ms)	p95/p99 (ms)
1	1.00	10.267	10.267	1600.775	3244.449 / 5150.723
5	0.20	11.589	57.945	1439.322	1986.777 / 4817.534
10	0.10	11.111	111.110	1482.304	1765.638 / 4566.243
20	0.05	11.861	237.220	1418.998	2349.729 / 4658.278
100	0.01	11.828	1182.800	1427.072	1770.053 / 5372.008

Table 6.7 and Table 6.8 show that VALID TPS stays in the same range across the sweep. However, because each committed Δ=1 s window carries more samples as k increases (i.e., higher sensing rate within the window), the amount of telemetry anchored per second rises from ≈ 12 samples/s (k=1, 1 Hz) to $\approx 1.18 \times 10^3$ samples/s (k=100, 100 Hz), without increasing the on-chain transaction rate. Commit latency remains on the same order of magnitude across k, indicating that higher-rate sensing can be supported by aggregating samples into each Δ-second window without increasing on-chain transaction rate under the tested workload.

The reported latency values should be interpreted as client-observed end-to-end commit latency, not as a measurement of only one internal blockchain stage. In other words, the reported delay reflects the full time required for a submitted telemetry transaction to be endorsed, ordered, included in a block, validated, committed, and acknowledged back to the client. This is an important metric in the present work because it represents the actual delay that the turbine-side IPC experiences before a window is durably recorded on the ledger. The results show that this delay is strongly influenced by batching configuration, particularly the orderer timeout, because larger batching settings increase the waiting time before block formation. At the same time, the measured delay remains within a practically usable range for supervisory telemetry logging and audit-oriented monitoring, which is the target scope of the proposed framework.

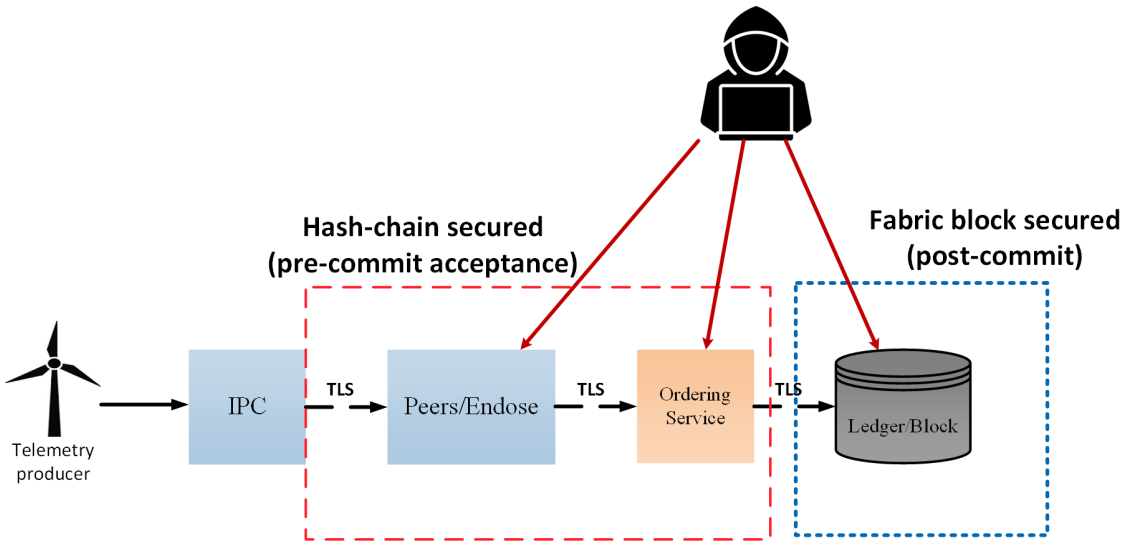

Figure 6.3: Adversary model and attack surface considered in the integrity experiments.

6.3 Integrity evaluation

Fig. 6.3 summarizes the adversary model and the two integrity layers evaluated in this section. We consider (i) an active attacker who can submit forged or out-of-order windows on the IPC to ledger path (handled pre-commit by per-turbine hash-chain enforcement during endorsement in Evaluation #7), and (ii) a disk-level attacker who tampers with a peer's local block store after commit (handled post-commit by tamper evidence of the peer ledger, evaluated in Evaluation #6).

Evaluation #6 (Framework I) Post-commit tamper-evidence of the peer ledger

This experiment evaluates post-commit integrity under a disk-level attacker who modifies a peer's on-disk block store after blocks have already been committed. We tamper with the victim peer's local channel block store for `turbine-chan`, where Fabric persists committed blocks in sequential `blockfile_00000x` segment files. The target block number in this test is B=8741, which resides in `blockfile_000009` on the victim peer.

We first restored a known-clean copy of `blockfile_000009`. As a consistency check, we queried block B=8741 via QSCC (Query System Chaincode) from (i) a clean peer and (ii) the victim peer, using `--raw` to capture the returned block bytes (a Protocol Buffers / protobuf-encoded

Fabric block message). The SHA-256 digests of the two raw outputs matched, confirming that both peers served identical bytes before tampering.

Next, the victim peer was stopped and a single byte was flipped inside the payload region of the last stored block record in the blockfile (i.e., not touching the record length prefix), so the file remains readable but the stored block content is corrupted. The resulting tampered blockfile artifact has a different SHA-256 digest from the clean copy. We then copied the tampered artifact into the victim container as `blockfile_000009` and verified the in-container SHA-256 digest matches the tampered artifact.

After restarting the victim peer, we repeated the QSCC query for $B{=}8741$ from both peers. The clean peer returned the same digest as before, while the victim peer returned different raw bytes (different SHA-256 digest) for the same block number. Fig. 6.4 summarizes the evidence chain. This demonstrates post-commit tamper-evidence: a disk attacker can corrupt a single peer's local history, but the corruption is detectable because it produces a cross-peer block-bytes mismatch, so clients can reject the result and rely on an untampered peer's replicated ledger.

Evaluation #7 (Framework I) Hash-chain enforcement against forged stream continuation

This experiment tests whether our on-chain per-turbine hash chain prevents an attacker from extending a turbine's data stream with a forged "next window". The evaluation steps include:

- We first query the turbine's *tail state* (the ledger's stored pointer to the latest accepted window for that turbine). The tail returns (i) `last_seq`, the latest window sequence number, and (ii) `last_hash`, the hash of that latest window.
- We then construct a malicious `PutWindowRaw` transaction for the *next* sequence number (`seq = last_seq + 1`), but we intentionally set `prev_hash` to an incorrect value. This models an active forgery where the attacker tries to "continue" the stream while breaking the required link to the last committed window.
- When peers endorse the proposal, they re-execute the chaincode and compare the submitted `prev_hash` with the expected value (`last_hash` stored in the tail). Because the values

```
(A) Host-side blockfile artifacts (clean vs tampered):
sha256sum blockfile_before.bin blockfile_after_lastpayload.bin
1346...8352  blockfile_before.bin
1c70...f37d  blockfile_after_lastpayload.bin

(B) Victim peer on-disk blockfile digest (inside container):
# victim path: .../turbine-chan/blockfile_000009
docker exec peer10... sha256sum .../blockfile_000009
1c70...f37d  .../blockfile_000009

(C) QSCC raw block bytes for the same block number (B=8741):
# clean_8741.pb  = QSCC --raw output from clean peer (protobuf-
   encoded block)
# victim_8741.pb = QSCC --raw output from victim peer after
   tampering
sha256sum clean_8741.pb victim_8741.pb
1a4f...1f8d  clean_8741.pb
669e...04e7  victim_8741.pb
```

The victim peer is verified to use the tampered blockfile (B), and it returns different raw block bytes for the same block number compared to a clean peer (C).

Figure 6.4: Evaluation #6 evidence: controlled post-commit corruption of the victim peer block store (`blockfile_000009`) and detection via cross-peer comparison of QSCC `GetBlockByNumber` raw outputs for block B=8741.

do not match, endorsement fails with a `prev_hash mismatch` error, so the transaction is rejected and never reaches ordering/commit.

- A final tail query confirms the ledger is unchanged (same `last_seq` and `last_hash`). This shows that the hash chain prevents out-of-order or tampered continuation of each turbine's stream even if an attacker can submit arbitrary transactions.

Fig. 6.5 shows the full evidence log (A–D), including the key/notation in the minipage.

This check enforces per-turbine *continuity*, but it does not validate the physical correctness of measurements if an attacker can submit correctly chained values.

Evaluation #8 (Framework II) Hash-chain enforcement against a forged continuation attempt

This experiment evaluates continuity enforcement in Framework II under a forged continuation attempt. In Framework II, telemetry records are submitted directly from multiple WFs to the shared

```
(A) Tail query (before): last_seq=303
last_hash = ea584355 4c16eeaa ff39a745 0f41aca8
            fdd7a5a5 6bb07c93 04c5b074 906ac2de

(B) Forged payload (attacker): seq=304
previous_hash = 00000000 00000000 00000000 00000000
            00000000 00000000 00000000 000000ff

(C) Peer result (endorsement rejected):
previous_hash mismatch:
  got      00000000 00000000 00000000 00000000 00000000 00000000
     00000000 000000ff
  expected ea584355 4c16eeaa ff39a745 0f41aca8 fdd7a5a5 6bb07c93
     04c5b074 906ac2de

(D) Tail query (after): last_seq=303 (unchanged)
last_hash = ea584355 4c16eeaa ff39a745 0f41aca8 fdd7a5a5 6bb07c93
    04c5b074 906ac2de
```

The forged transaction is rejected at endorsement due to a hash-chain link violation; the tail state remains unchanged.

Key: tail state (`last_seq`, `last_hash`); `seq` increments by 1; `prev_hash` must equal the previous window hash.

Figure 6.5: Evaluation #7 evidence log for hash-chain enforcement against a forged stream continuation (A) tail before, (B) forged payload, (C) endorsement rejection, (D) tail after.

control-center ledger on channel `wfchan`, and the chaincode checks whether each new record links correctly to the latest committed state for the corresponding turbine identifier.

We first query the tail state of one selected turbine stream to obtain the latest committed sequence number and hash. We then prepare a forged submission for the next expected sequence number, but with an intentionally incorrect `previous_hash`. When the proposal reaches the endorsing peers, the submitted link is compared against the current committed tail. Because the submitted hash does not match the expected previous hash, endorsement fails and the transaction is not committed. A final tail query confirms that the ledger state remains unchanged.

Fig. 6.6 shows the tail state before submission, the forged payload, the endorsement-time rejection, and the unchanged tail state after the failed submission. This shows that the same continuity rule remains effective in Framework II: a record cannot extend a turbine stream unless it links correctly to the latest accepted record for that turbine.

```
(A) Tail query (before): last_seq=30
last_hash = 7f42d8c1 3f04a9ab c77d2e19 65f1d3b8
            0a9ce671 44b2ef10 8cd4e5a3 91ff20bc

(B) Forged payload (attacker): seq=31
previous_hash = 7f42d8c1 3f04a9ab c77d2e19 65f1d3b8
                8b7e1142 55aa901c d31f6e28 4c90bb73

(C) Peer result (endorsement rejected):
previous_hash mismatch:
  got      7f42d8c1 3f04a9ab c77d2e19 65f1d3b8 8b7e1142 55aa901c
     d31f6e28 4c90bb73
  expected 7f42d8c1 3f04a9ab c77d2e19 65f1d3b8 0a9ce671 44b2ef10
     8cd4e5a3 91ff20bc

(D) Tail query (after): last_seq=30 (unchanged)
last_hash = 7f42d8c1 3f04a9ab c77d2e19 65f1d3b8
            0a9ce671 44b2ef10 8cd4e5a3 91ff20bc
```

Representative Prototype II evidence trace. The forged transaction is rejected at endorsement because the submitted hash link does not match the current committed tail state.

Figure 6.6: Evaluation #8 evidence log for continuity enforcement in Framework II against a forged continuation attempt. The submitted `previous_hash` does not match the current tail state, so the transaction is rejected before ordering and commit.

Evaluation #9 (Framework II) Sequential backfill after delayed delivery

This experiment evaluates delayed submission handling in Framework II under a temporary communication interruption. The goal is to examine whether records that cannot be submitted immediately can still be accepted after connectivity is restored, provided that they are replayed in the correct sequence.

Submission from one selected WF client is paused for a short interval while telemetry generation continues locally. The unsent records are buffered at the client side in their original order. After connectivity is restored, the client resubmits the buffered records sequentially. Because each replayed record carries the correct `previous_hash` relative to the last committed turbine state, the records remain consistent with the ledger head and can be accepted one after another. As a control case, if a later buffered record is submitted before its missing predecessor, endorsement fails because its `previous_hash` no longer matches the current committed tail.

```
(A) Tail query before interruption:
last_seq = 30
last_hash = c184e20f 5b91ac44 8d1aee37 703db112
            b8f3c1a2 1cd44e77 0f98a6b0 4e1269dd

(B) Buffered records during interruption:
seq=31 -> prev_hash = c184e20f 5b91ac44 8d1aee37 703db112
   b8f3c1a2 1cd44e77 0f98a6b0 4e1269dd
seq=32 -> prev_hash = 5e7a91c4 0b22f18e 64d893ab 9cf0d112 3
   aeb4f88 22dd8a79 118cde45 7f209a31
seq=33 -> prev_hash = 54ad09c2 7be13d44 6f4c7aa1 90ee2180
   d0a6f331 1bc920fe 2a7719c8 48fe01bd

(C) Sequential replay after connectivity is restored:
submit seq=31 -> VALID
submit seq=32 -> VALID
submit seq=33 -> VALID

(D) Tail query after replay:
last_seq = 33
last_hash = 91db71e4 6a30d2c9 0bc8f224 81e3a75d
            447a29c0 6ef2b913 75d0ab1e 2cc948f6

(E) Control case (out-of-sequence replay):
submit seq=33 first -> previous_hash mismatch
  got      54ad09c2 7be13d44 6f4c7aa1 90ee2180 d0a6f331 1bc920fe
     2a7719c8 48fe01bd
  expected c184e20f 5b91ac44 8d1aee37 703db112 b8f3c1a2 1cd44e77
     0f98a6b0 4e1269dd
```

Representative Prototype II evidence trace. Delayed records are accepted when replayed sequentially with valid continuity links, while an out-of-sequence replay attempt is rejected at endorsement.

Figure 6.7: Evaluation #9 representative evidence log for delayed submission handling in Prototype II. Buffered records can be committed after communication recovery when replayed sequentially with correct hash linkage, whereas out-of-sequence replay is rejected.

Fig. 6.7 shows the tail state before the interruption, the buffered records, the successful sequential replay after connectivity is restored, the updated tail state after replay, and the rejection of an out-of-sequence replay attempt. This shows that continuity enforcement in Framework II allows delayed records to be accepted through buffering and sequential replay, while still rejecting skipped or out-of-sequence continuation.

Chapter 7

Conclusion

This thesis presented a tamper-evident framework for recording in-farm WF telemetry on a permissioned ledger. The proposed workflow targets measurement transfer from turbine-side IPCs to the WF control room and augments Hyperledger Fabric with a per-turbine hash chain across fixed-duration measurement windows to enforce stream continuity at submission time. In addition, the thesis extended the same integrity concept to a broader multi-WF communication setting, where telemetry from multiple WFs is delivered to a shared control-center blockchain network.

To reduce transaction load while preserving audit-relevant context, the prototype logs telemetry as fixed-duration windows of length Δ and submits one transaction per turbine per window. Each window includes a sequence number and a hash link to the previously accepted window for the same turbine, providing stream-level continuity that complements Fabric's block hash chain (tamper-evidence after commit). The same window format also supports higher-rate sensing by aggregating multiple samples into each window when $T_s < \Delta$ (captured by `sample_count`). A prototype was implemented on Hyperledger Fabric v2.5.4 with two organizations (OrgNorth and OrgSouth), 24 peers, and a Raft-based ordering service, driven by EMTP-based WF telemetry traces replayed through a trace replay service for repeatable experiments. Performance results show that orderer batching parameters have a strong impact on end-to-end commit delay. Under sequential submission (T=1, N=200, Δ=1 s), Config L achieved a mean commit latency of 846 ms (p95 899 ms), while Config T increased the mean to 2365 ms (p95 2423 ms) (Table 6.3). Under concurrent load (T=24, Δ=1 s), Config L sustained 11.0 VALID TPS (mean 1507 ms), while Config T sustained

7.26 VALID TPS (mean 2741 ms) with larger average blocks (Table 6.4). Sensitivity sweeps under Config L indicated that varying Δ in the 1–2 s range did not substantially change throughput or mean commit latency under the tested load (Table 6.5). Increasing payload size (signals per sample) primarily increased estimated ledger growth (approximately 10.2→21.3 MB/h from S=8 to S=32) without materially degrading throughput in the tested range (Table 6.6). Finally, higher-rate sampling was supported through per-window aggregation: with $\Delta = 1$ s and $k \in \{1, 5, 10, 20, 100\}$, VALID TPS remained in the same range while effective sample throughput scaled linearly to over 1.18×10^3 samples/s at 100 Hz (Table 6.7, Table 6.8). The second prototype further showed that the same integrity mechanism can be carried beyond the in-farm path to a shared control-center setting with multiple WF participants.

Integrity experiments validated two complementary protections. After commit, disk-level post-commit tampering was detectable: modifying a peer's on-disk block store altered its SHA-256 checksum and returned different raw block bytes than a clean peer, which made the corruption detectable via digest mismatch (Fig. 6.4). Before commit, forged stream continuation was prevented: a transaction with an incorrect `prev_hash` failed endorsement and was not ordered or committed, demonstrating that per-turbine continuity is enforced at submission time (Fig. 6.5). In Prototype II, additional integrity checks showed that the same continuity rule also rejects forged or out-of-order continuations in the multi-WF setting, while still allowing delayed records to be accepted when they are replayed sequentially with valid hash linkage.

Overall, the thesis shows that the proposed framework can strengthen integrity assurance across a broader communication scope in wind energy systems, covering both in-farm telemetry transfer and multi-WF telemetry delivery to the control-center layer. Future work will scale the performance evaluation to larger turbine counts and longer experiment horizons, quantify full ledger growth including Fabric envelope and block metadata overhead, and assess resilience under additional adversarial conditions such as replay/reorder attempts that preserve hash-chain linkage and availability attacks on the in-farm communication path as well as under broader inter-farm communication scenarios.

Bibliography

[1] Mohsen Ghafouri. *Subsynchronous resonance in dfig-based wind farms*. Ecole Polytechnique, Montreal (Canada), 2018.

[2] M. Ansari, M. Ghafouri, A. Ameli, and U. Karaagac. Detection and mitigation of cyber-attacks against frequency control in power grids integrated with large-scale wind farms. Unpublished manuscript, December 2024.

[3] M. Ansari, M. Ghafouri, A. Ameli, U. Karaagac, and I. Kocar. Detection and mitigation of false data injection attacks against wind farm active power controllers in power grids. *Journal of Modern Power Systems and Clean Energy*, 2025. Early Access.

[4] Hyperledger Fabric. Transaction Flow. Hyperledger Fabric Documentation, 2026. URL `https://hyperledger-fabric.readthedocs.io/en/latest/txflow.html`. Accessed: 2026-01-20.

[5] M. A. Ahmed and Y.-C. Kim. Communication network architectures for smart-wind power farms. *Energies*, 7:3900–3921, 2014.

[6] M. Verma, C. Bothmann, W. Wurth, and J. S. Harmouch. Wind turbine gearbox fault prognosis using high-frequency scada data. *Journal of Physics: Conference Series*, 2265(3):032067, 2022. doi: 10.1088/1742-6596/2265/3/032067.

[7] International Energy Agency (IEA). Global energy review 2025, 2025. URL `https://iea.blob.core.windows.net/assets/5b169aa1-bc88-4c96-b828-aaa50406ba80/GlobalEnergyReview2025.pdf`. Online.

[8] Global Wind Energy Council (GWEC). Wind industry installs record capacity in 2024 despite policy instability, April 2025. URL https://www.gwec.net/news/wind-industry-installs-record-capacity-in-2024-despite-policy-instability Apr. 23, 2025. Online.

[9] Intergovernmental Panel on Climate Change (IPCC). Climate change 2022: Mitigation of climate change — summary for policymakers. Technical report, IPCC, 2022. URL https://www.ipcc.ch/report/ar6/wg3/downloads/report/IPCC_AR6_WGIII_SummaryForPolicymakers.pdf.

[10] International Renewable Energy Agency (IRENA). Renewable power generation costs in 2023. Technical report, International Renewable Energy Agency, Abu Dhabi, 2024. URL https://www.irena.org/Publications/2024/Sep/Renewable-power-generation-costs-in-2023.

[11] International Electrotechnical Commission. Wind energy generation systems—part 25-71: Communications for monitoring and control of wind power plants—configuration description language. IEC TS 61400-25-71:2019, September 2019.

[12] A. Mwangi, R. Sahay, E. Fumagalli, M. Gryning, and M. Gibescu. Towards a software-defined industrial IoT-edge network for next-generation offshore wind farms: State of the art, resilience, and self-x network and service management. *Energies*, 17(12):2897, June 2024. doi: 10.3390/en17122897.

[13] Ravi Pandit, Davide Astolfi, Jiarong Hong, David Infield, and Matilde Santos. Scada data for wind turbine data-driven condition/performance monitoring: A review on state-of-art, challenges and future trends. *Wind Engineering*, 47(2):422–441, 2023.

[14] Keith Stouffer, Suzanne Lightman, Victoria Pillitteri, Marshall Abrams, and Adam Hahn. Guide to operational technology (ot) security. Technical Report NIST Special Publication 800-82 Revision 3, National Institute of Standards and Technology (NIST), 2023. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf. Accessed: 2026-01-24.

[15] U.S. Energy Information Administration. Wind explained: Types of wind turbines, 2023. URL https://www.eia.gov/energyexplained/wind/types-of-wind-turbines.php. Accessed: 2026-01-25.

[16] National Renewable Energy Laboratory. Land-based wind reference architecture. Technical report, National Renewable Energy Laboratory (NREL), 2025. URL https://docs.nrel.gov/docs/fy25osti/91222.pdf. Accessed: 2026-01-25.

[17] International Electrotechnical Commission (IEC). Wind energy generation systems – communications for monitoring and control of wind power plants – part 1: Overall description and principles. Technical Report IEC 61400-25-1, International Electrotechnical Commission (IEC), 2017.

[18] Min Du, Xin Zhang, Gareth Taylor, and Vladimir Terzija. Game theory-based vulnerability analysis of cyber-physical power systems under interdependent network attacks. *Cyber-Physical Energy Systems*, 1(1):116–124, 2025. ISSN 3050-7448. doi: https://doi.org/10.1016/j.cpes.2025.08.001. URL https://www.sciencedirect.com/science/article/pii/S3050744825000064.

[19] Sophos. The state of ransomware 2024, 2024. URL https://www.sophos.com/en-us/content/state-of-ransomware. Online.

[20] U.S. Department of Energy. Roadmap for wind cybersecurity. Online report (PDF), 2020. URL https://www.energy.gov/sites/prod/files/2020/08/f77/wind-energy-cybersecurity-roadmap-2020v3.pdf. Accessed: 2026-01-24.

[21] Reuters. Satellite outage knocks out control of enercon's wind turbines. News article, February 2022. URL https://www.reuters.com/business/energy/satellite-outage-knocks-out-control-enercon-wind-turbines-2022-02-28/. Accessed: 2026-01-24.

[22] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. Technical Report NISTIR 8202, National Institute of Standards and Technology (NIST), 2018.

URL https://nvlpubs.nist.gov/nistpubs/ir/2018/nist.ir.8202.pdf. Accessed: 2026-01-20.

[23] Weipeng Liu, Upendra Prasad, Yutian Liu, and Lei Wu. Stability analysis of droop-free controlled islanded microgrids with asymmetric communication networks. *Cyber-Physical Energy Systems*, 2(1):53–63, 2026. ISSN 3050-7448. doi: https://doi.org/10.1016/j.cpes.2025.08.003. URL https://www.sciencedirect.com/science/article/pii/S305074482500009X.

[24] Benedikt Putz, Florian Menges, and Günther Pernul. A secure and auditable logging infrastructure based on a permissioned blockchain. *Computers Security*, 87:101602, 2019. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2019.101602. URL https://www.sciencedirect.com/science/article/pii/S0167404818313907.

[25] Ashar Ahmad, Muhammad Saad, and Aziz Mohaisen. Secure and transparent audit logs with blockaudit. *Journal of Network and Computer Applications*, 145:102406, 2019. doi: 10.1016/j.jnca.2019.102406.

[26] Louis Shekhtman and Erez Waisbard. Engravechain: A blockchain-based tamper-proof distributed log system. *Future Internet*, 13(6):143, 2021. doi: 10.3390/fi13060143.

[27] Cristina Regueiro, Iñaki Seco, Iván Gutiérrez-Agüero, Borja Urquizu, and Jason Mansell. A blockchain-based audit trail mechanism: Design and implementation. *Algorithms*, 14(12), 2021. ISSN 1999-4893. doi: 10.3390/a14120341. URL https://www.mdpi.com/1999-4893/14/12/341.

[28] Andrea Margheri, Massimiliano Masi, Abdallah Miladi, Vladimiro Sassone, and Jason Rosenzweig. Decentralised provenance for healthcare data. *International Journal of Medical Informatics*, 141:104197, 2020. ISSN 1386-5056. doi: https://doi.org/10.1016/j.ijmedinf.2020.104197. URL https://www.sciencedirect.com/science/article/pii/S1386505619312031.

[29] Kenny Awuson-David, Tawfik Al-Hadhrami, Mamoun Alazab, Nazaraf Shah, and Andrii

Shalaginov. Bcfl logging: An approach to acquire and preserve admissible digital forensics evidence in cloud ecosystem. *Future Generation Computer Systems*, 122:1–13, 2021. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2021.03.001. URL https://www.sciencedirect.com/science/article/pii/S0167739X21000807.

[30] Adetomike Adeyemi, Mingyu Yan, Mohammad Shahidehpour, Cristina Botero, Alba Valbuena Guerra, Niroj Gurung, Liuxi (Calvin) Zhang, and Aleksi Paaso. Blockchain technology applications in power distribution systems. *The Electricity Journal*, 33(8):106817, October 2020. doi: 10.1016/j.tej.2020.106817.

[31] Y. Zhuang, X. Zhang, and L. Shu. A survey of blockchain-based solutions for smart grid cybersecurity. *IEEE Access*, 8:21980–21994, 2020.

[32] M. Faheem et al. A blockchain-based resilient and secure framework for events monitoring and control in distributed renewable energy systems. *IET Blockchain*, 4(Suppl. 1):644–658, 2024.

[33] M. Gerardi, F. Fallucchi, and F. Orecchini. Blockchain technology for monitoring energy production for reliable and secure big data. *Electronics*, 12(22):4660, 2023.

[34] M. Faheem and M. A. Al-Khasawneh. Multilayer cyberattacks identification and classification using machine learning in internet of blockchain (IoBC)-based energy networks. *Data in Brief*, 54:110461, 2024.

[35] A. Ionita, K. Holm, R. C. Goduscheit, P. H. Lauritsen, W. Prinz, K. N. Jacobsen, and K. I. Thomsen. Developing a blockchain-based prototype for wind turbine fasteners. Proc. UnWind Project, Fraunhofer FIT and Aarhus University, 2021.

[36] K. Holm. Blockchain as a sustainable service-enabler: A case of wind turbine traceability. In *Proc. 8th CARV and 10th MCPC Conf.* Aalborg University, 2021.

[37] H. Yu, S. Zhu, and J. Yang. The quality control system of green composite wind turbine blade supply chain based on blockchain technology. *Sustainability*, 13(15):8331, 2021.

[38] A. Gómez-Goiri, D. Garcia-Estevez, J.-T. San-José Lombera, and I. Gutierrez-Aguero. Traceability for wind blade recycling: Balancing transparency, privacy, and data integrity. In *Proc. BLOCKCHAIN 2025*, volume 1635 of *Lecture Notes in Networks and Systems (LNNS)*, pages 274–283, 2026. doi: 10.1007/978-3-032-05877-5_26.

[39] X. Cheng et al. A blockchain-empowered cluster-based federated learning model for blade icing estimation on IoT-enabled wind turbine. *IEEE Transactions on Industrial Informatics*, 18(12):9184–9196, December 2022.

[40] H. Wang and B. Wu. Design of wind farm information system based on blockchain technology. In *IOP Conference Series: Earth and Environmental Science*, volume 647, page 012006, 2021.

[41] M. Ghafouri, U. Karaagac, et al. A cyber attack mitigation scheme for series compensated DFIG-based wind parks. *IEEE Transactions on Smart Grid*, 2021.

[42] M. Ansari, M. Ghafouri, and A. Ameli. Cyber-security vulnerabilities of the active power control scheme in large-scale wind-integrated power systems. In *Proc. EPEC*, 2022.

[43] A. Amini, M. Ghafouri, et al. Secure sampled-data observer-based control for wind turbine oscillation under cyber attacks. *IEEE Transactions on Smart Grid*, 2022.

[44] M. McCarty, J. Johnson, et al. Cybersecurity resilience demonstration for wind energy sites in co-simulation environment. *IEEE Access*, 2023.

[45] H. Badihi, S. Jadidi, et al. Smart cyber-attack diagnosis and mitigation in a wind farm network operator. *IEEE Transactions on Industrial Informatics*, 2022.

[46] S. Jadidi, H. Badihi, and Y. Zhang. Hybrid fault-tolerant and attack-resilient cooperative control in an offshore wind farm. *IEEE Transactions on Sustainable Energy*, 2023.

[47] H. Du, J. Yan, et al. Modeling and assessment of cyber attacks targeting converter-driven stability of power grids with PMSG-based wind farms. *IEEE Transactions on Power Systems*, 2024.

[48] International Electrotechnical Commission (IEC). Wind energy generation systems—part 1: Design requirements. Technical Report IEC 61400-1:2019, International Electrotechnical Commission (IEC), 2019.

[49] Yichi Zhang, Yingmeng Xiang, and Lingfeng Wang. Power system reliability assessment incorporating cyber attacks against wind farm energy management systems. *IEEE Transactions on Smart Grid*, 8(5):2343–2357, September 2017. doi: 10.1109/TSG.2016.2523515.

[50] North American Electric Reliability Corporation (NERC). Pmu placement guideline. Reliability Guideline, December 2016. URL `https://www.nerc.com/globalassets/who-we-are/standing-committees/rstc/smwg/reliability-guideline---pmu-placement.pdf`.

[51] Meng Dai, Mario A. Rotea, and Nasser Kehtarnavaz. Obtaining rotational stiffness of wind turbine foundation from acceleration and wind speed scada data. *Sensors*, 25(15):4756, 2025. doi: 10.3390/s25154756.

[52] Adaiton Oliveira-Filho, Monelle Comeau, James Cave, Charbel Nasr, Pavel Côté, and Antoine Tahan. Wind turbine scada data imbalance: A review of its impact on health condition analyses and mitigation strategies. *Energies*, 18(1):59, 2025. doi: 10.3390/en18010059.

[53] Elena Gonzalez, Bruce Stephen, David Infield, and Julio J. Melero. Using high-frequency scada data for wind turbine performance monitoring: A sensitivity study. *Renewable Energy*, 131:841–853, 2019. doi: 10.1016/j.renene.2018.07.068.

[54] Simon Letzgus. Change-point detection in wind turbine scada data for robust condition monitoring with normal behaviour models. *Wind Energy Science*, 5:1375–1397, 2020. doi: 10.5194/wes-5-1375-2020.

[55] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. White paper, 2008.

[56] Ethereum Foundation. Ethereum proof-of-stake finality. Ethereum Improvement Proposal EIP-3675, 2021. Finality after two epochs (12.8 minutes).

[57] Nomadic Labs. How scalable is tezos? Technical report, Nomadic Labs, 2022.

[58] Tezos Core Developers. Tenderbake finality. Octez Documentation, 2022.

[59] Bitcoin Project. Confirmations. Bitcoin Developer Guide, 2026. URL https://developer.bitcoin.org/devguide/transactions.html#confirmations. Accessed: 2026-01-25.

[60] Elli Androulaki et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, 2018.

[61] Fabian Geyer et al. An end-to-end performance comparison of seven permissioned blockchain systems. arXiv preprint, 2023. URL https://arxiv.org/abs/2311.15433. Accessed: 2026-01-23.

[62] Chenyang Guo, Gongshu Wang, Yanyan Zhang, and Lixin Tang. Multi-objective optimization for energy blockchain cloud storage: Achieving efficiency and security synergy. *Cyber-Physical Energy Systems*, 1(1):56–70, 2025. ISSN 3050-7448. doi: https://doi.org/10.1016/j.cpes.2025.08.002. URL https://www.sciencedirect.com/science/article/pii/S3050744825000088.