International Information Technology University

UDC 004.273:004.652 On manuscript right

Imankulova Binara Bakytzhanovna

Design and implementation of a provenance-aware LoRaWAN IoT framework for smart city air-quality data management

8D06102 – Computer and Software Engineering

Dissertation for the degree of
doctor of philosophy (PhD)

Scientific consultants
Doctor of Technical Sciences,
Professor Uskenbayeva R.K.
Candidate of Technical Sciences,
Associate Professor,
Amanzholova S.T.

Foreign consultant
Doctor of Natural Sciences,
Professor Dietmar Beyer

Republic of Kazakhstan
Almaty, 2026

CONTENTS

NORMATIVE REFERENCES 3
SYMBOLS AND ABBREVIATIONS 4
INTRODUCTION 5
1 RESEARCH BACKGROUND AND LITERATURE REVIEW 12
1.1 Air-quality monitoring in smart cities and LPWAN constraints 12
1.2 Trustworthiness of sensor data and the concept of data provenance 17
1.3 Methodology of the systematic literature review 20
1.4 Provenance capture and representation in IoT systems 24
1.5 Integrity of provenance and log records 28
1.6 Data loss, missingness and completeness in LoRaWAN data pipelines 31
1.7 Comparative analysis, research gaps and framework requirements 34
2 DESIGN OF THE PROVENANCE-AWARE IOT FRAMEWORK 37
2.1 Design principles and architecture 37
2.2 Integration with the LoRaWAN communication stack 41
2.3 Threat model and trust assumptions 44
2.4 Formal model of the framework 52
2.5 Provenance storage and verification 65
3 IMPLEMENTATION OF THE PROVENANCE-AWARE IOT FRAMEWORK 73
3.1 Architecture of the implemented platform 73
3.2 LoRaWAN deployment and data acquisition setup 75
3.3 MQTT ingestion and provenance capture 81
3.4 Provenance-aware storage and database schema 87
3.5 Integrity verification and audit workflow 91
3.6 Provenance queries and graph-view reconstruction 94
3.7 Missing-data attribution procedure 94
4 EXPERIMENTAL EVALUATION OF THE PROPOSED FRAMEWORK 97
4.1 Experimental setup, dataset, and evaluation metrics 97
4.2 Data completeness and missingness analysis 102
4.3 LoRaWAN link quality, ADR behavior, and reception diversity 104
4.4 Storage growth and replay-based platform profiling 112
4.5 Integrity verification and provenance traceability 114
4.6 Simulation-based evaluation of missing-data attribution 120
4.7 Discussion and limitations 125
CONCLUSION 129
REFERENCES 132
APPENDIX A Search queries in Scopus and Web of Science databases 144
APPENDIX B Sensor specifications 147
APPENDIX C System implementation and verification outputs 148
APPENDIX D Simulation code and reproduction of the attribution experiment 151
APPENDIX E Act of implementation 154

NORMATIVE REFERENCES

This thesis uses references to the following standards and regulatory documents:

Instructions for the preparation of a thesis and an abstract, Higher Attestation Commission of the Ministry of Education and Science of the Republic of Kazakhstan, dated September 28, 2004, No. 377-3.

GOST 7.32-2017. System of standards on information, librarianship and publishing. Report on research work. Structure and rules of presentation.

GOST 7.1-2003. System of standards on information, librarianship and publishing. Bibliographic record. Bibliographic description. General requirements and rules.

ST RK 34.005-2002. Information Technology. Basic terms and definitions.

ST RK 34.015-2002. Information Technology. Set of standards for automated systems. Terms of reference for creating an information system.

ISO/IEC 24039:2022. Information technology. Smart city digital platform reference architecture. Data and service.

ISO 37120:2018. Sustainable cities and communities. Indicators for city services and quality of life.

SYMBOLS AND ABBREVIATIONS

ABP	Activation by Personalization
ADR	Adaptive Data Rate
AES	Advanced Encryption Standard
CO	Carbon Monoxide
DBMS	Database Management System
DevEUI	Device Extended Unique Identifier
DR	Data Rate
DoS	Denial of Service
ER	Entity-Relationship
FCnt	Frame Counter
GDPR	General Data Protection Regulation
GPS	Global Positioning System
HMAC	Hash-based Message Authentication Code
IDS	Intrusion Detection System
IoT	Internet of Things
ISM	Industrial, Scientific and Medical
JSON	JavaScript Object Notation
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LPWAN	Low-Power Wide-Area Network
LTE-M	Long-Term Evolution for Machines
MIC	Message Integrity Code
MQTT	Message Queuing Telemetry Transport
NB-IoT	Narrowband Internet of Things
OTAA	Over-the-Air Activation
PM	Particulate Matter
PROV-O	PROV Ontology
PROV-DM	Provenance Data Model
RH	Relative Humidity
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SNR	Signal-to-Noise Ratio
UART	Universal Asynchronous Receiver-Transmitter
W3C	World Wide Web Consortium

INTRODUCTION

General characteristics of the work.

The dissertation presents the design and the implementation of a provenance-aware LoRaWAN Internet of Things (IoT) framework. The targeted application area was chosen as the management of air-quality data in smart cities. Provenance, integrity, and completeness are three main dimensions that the framework addresses trust in monitoring data. Provenance here describes how each stored measurement was received, decoded, validated and then stored. Integrity is the ability to detect subsequent changes to the recorded history as well as completeness reflects whether all expected measurements were received and, if not, how many are missing.

The framework captures provenance on the platform side, after uplink reception. It requires no changes to the LoRaWAN protocol and adds no provenance data to the sensing-node payload. Provenance records follow W3C PROV semantics and they are stored in per-device hash-linked chains with chain-head checkpoints. It means that the design provides tamper evidence under a stated threat model for the stored history. The same records are used to build PROV-style graph views for backward and forward tracing. For each missing measurement, the framework identifies the stage where the loss occurred: the device, upstream communication, platform intake, or processing. The cause of each data loss is identified by means of the frame counters, the reception times, the state of the records and the event log of the network server. When the available data are not sufficient, the cause is classified as undetermined.

Relevance of the work.

Internet of Things (IoT) has become an important component of the modern urban infrastructure across a wide range of services such as in smart cities. Where smart city services have relied on measurements of sensing nodes which transferred through gateways, communication networks and also platform-level processing systems. Current forecasts have estimated the number of connected devices worldwide will reach 32.1 billion by 2030 [1]. The growing number of connected devices has increased both the volume and the heterogeneity of data across a wide range of urban applications such as in environmental monitoring, traffic management as well as public safety [2], [3].

Sensor measurements have supported a wide range of activities such as in monitoring, reporting, operational control and decision-making. When each measurement goes through a chain of some stages, for example, it could be sensing, or wireless transmission, ingestion, processing, storage and publication, and all this before becoming part of the operational data set. Metadata can be added, the data format can be changed, or a new record can be created at each stage and during the platform-side processing measurements may be lost, duplicated, replayed, incorrectly decoded, or modified [4], [5], [6].

Malicious activity isn't the only source of such problems; for example, packet loss, duplicate reception by multiple gateways, decoding failures, as well as storage errors could impact to the final dataset. Air-quality monitoring has been especially sensitive to such conditions. Since missing or incorrectly processed values may have

influencing the long-term statistics, the environmental indicators. It could be difficult to determine how a stored value was obtained (transformed or written) to the database if the processing history is not recorded.

On the one hand, data provenance is used as a record of the history of each measurement. More specifically, provenance describes the source of a measurement, the conditions of reception, the processing operations and the resulting stored record. Typically, provenance information can support a number of functions such as traceability, auditability and integrity verification (i.e., the capacity to detect later changes on the stored data). On the other hand, the above functions are of particular interest when environmental data are used for public reporting, scientific analysis and long-term urban planning. For example, a single missing or modified value may affect the long-term statistics that are published for a whole city district (e.g., the annual mean concentration of PM2.5).

Existing provenance approaches have provided different forms of protection and representation of data, based mainly on first, blockchain systems and second, cryptographic mechanisms. On the one hand, blockchain-based systems can preserve tamper-resistant records, but they often introduce additional overhead in terms of storage, also latency and consensus [7], [8], [9]. On the other hand, cryptographic mechanisms are able to detect the modification of records, although they may require additional computation, as well as key-management procedures or changes in the protocol [10], [11]. Despite their advantages, both approaches pose requirements that are difficult to accommodate in low-power sensing nodes and LPWAN deployments (e.g., limited memory and processing capacity, duty cycle restrictions, etc.).

Most often, LoRaWAN has been used for distributed environmental monitoring across a wide range of deployments such as in smart cities, since it provides long-range communication with a low energy consumption. Furthermore, the protocol provides the confidentiality of the payload and the integrity check of each message during transmission (i.e., by means of AES-128 encryption and a Message Integrity Code (MIC), respectively). However, it does not preserve the processing history of a measurement after uplink reception. Stored and exported records may therefore be changed without a corresponding provenance record. Recent studies show that practical approaches combining LoRaWAN-compatible provenance capture, tamper-evident storage, and traceability remain limited [12].

Last, but not least, missing data represent a further problem that existing approaches have rarely addressed. In long-term LoRaWAN deployments, a substantial share of the expected measurements is never stored, due to several reasons such as first, devices that stop transmitting, second, frames lost on the radio path and third, messages lost between the network server and the platform, or failed during decoding. Data completeness is typically reported as one percent, indicating how much data is missing but not where exactly the loss occurred. However, without such information, operators are not able to decide whether to repair the devices, or extend the coverage of gateways or correct the platform. Similarly, analysts cannot judge whether the gaps in the data are random or systematic.

The systematic literature review presented in Chapter 1 has identified four gaps in the existing provenance solutions for IoT and smart city systems: first, the

provenance capture requires changes in the end devices, or covers only part of the processing pipeline; second, the standard provenance models do not include any evidence of reception; third, the integrity claims are not tied to an explicit threat model; and fourth, there is no attribution of missing data at the level of each stage. In function of the above gaps, we define five requirements for a provenance-aware LoRaWAN platform, namely, the capture of provenance across all stages without changes in the end devices, a standard provenance model that includes the evidence of reception, the tamper evidence for the stored provenance under a stated threat model, a bounded overhead, and the attribution of missing data at the level of each stage. In this dissertation we develop a framework that meets such requirements, putting the focus on LoRaWAN deployments for air quality monitoring in smart cities.

The aim and objectives of the research. The aim of this dissertation is to develop a platform-side provenance-aware framework for traceable and tamper-evident management of air-quality data in LoRaWAN-based smart-city monitoring systems.

To achieve this aim, the dissertation addresses four **objectives**:

1. to analyse existing IoT data provenance approaches and identify their limitations in smart city and LPWAN deployments;
2. to design and formalize a lightweight provenance, integrity, and missing-data attribution model for LoRaWAN-based monitoring;
3. to develop and implement a platform-side prototype that captures, stores, verifies provenance records and reconstructs provenance graph views;
4. to evaluate the prototype in a LoRaWAN monitoring deployment and in simulation in terms of data completeness and attribution of missing data, platform overhead, integrity-verification performance, and provenance traceability.

The object of the research is the lifecycle of IoT measurement data in LoRaWAN-based smart city monitoring systems.

The subject of the research is the set of methods and mechanisms used to capture provenance, protect provenance integrity, and support traceability across the platform-side measurement lifecycle.

Methods of research. The research has combined four types of methods, namely analytical, design, empirical and simulation methods. The analytical stage included a systematic literature review conducted according to PRISMA 2020, which examined IoT data provenance, integrity protection, data completeness and LPWAN communications, and defined the requirements for the proposed framework. Then, the design stage translated such requirements into a platform-side architecture and a formal model of the generation of provenance, the verification of integrity and the attribution of missing data. Moreover, a functional prototype was implemented in order to capture provenance after the reception of each uplink, to store it in hash-linked chains per device, to verify the integrity of each chain and to reconstruct graph views for traceability queries.

Afterwards, the empirical stage evaluated the prototype by means of a 12-month LoRaWAN deployment for air quality monitoring, in terms of data completeness, ingestion performance of the backend, storage overhead, verification time of the

provenance chains, tamper detection and traceability. The attribution of missing data was evaluated by means of computer simulations. More specifically, the radio layer was simulated with the ns-3 network simulator and its LoRaWAN module, and failures of devices, gateways and platform with known causes were added, in order to compare the attributed causes with the true ones.

The scientific novelty

A platform-side provenance model for LoRaWAN monitoring is proposed, in which gateway reception evidence is an explicit W3C PROV entity and no changes to devices or the protocol are required, as confirmed by backward and forward tracing of stored measurements to their reception evidence.

A tamper-evident storage scheme adapted to the LoRaWAN uplink lifecycle is introduced, combining per-device hash chains and chain-head checkpoints, which detected all five tested types of record change and verified 403,900 records in 5.87 s.

A method for stage-level attribution of missing measurements is proposed, which, unlike aggregate completeness metrics, identifies whether data were lost at the device, upstream, at platform intake, or during processing, with 3 errors in 270,107 attributed measurements in ns-3 simulation.

Main thesis statements submitted for defense:

1) The architecture of a provenance-aware LoRaWAN framework on the platform side, aimed at air quality monitoring, which captures provenance, verifies its integrity and attributes missing data without any modification of the LoRaWAN protocol or of the payload of the sensing nodes.

2) The provenance model of the lifecycle of LoRaWAN uplinks, which represents the reception, ingestion and storage stages by means of W3C PROV entities, activities and agents, including the evidence of reception at the gateways, and supports both backward and forward traceability of the stored measurements.

3) The method for tamper-evident storage of provenance, based on hash-linked chains per device and checkpoints of the chain heads, which detects the modification, deletion, insertion and reordering of records, as well as the truncation of the chain tail, under the stated trust assumptions.

4) The method for the attribution of missing measurements at the level of each stage, which takes advantage of the evidence recorded by the platform and the network server in order to assign each missing measurement to one of the stages (i.e., device, upstream, platform intake or processing), or to classify it as undetermined.

Scientific results

1. A systematic literature review was conducted according to PRISMA 2020. Among 1,201 records retrieved from Scopus, Web of Science and additional sources, 90 studies were included in the review. According to the results, the existing approaches have rarely combined platform-side capture, suitability for LoRaWAN, tamper-evident storage, traceability and attribution of missing data.

2. A provenance-aware LoRaWAN framework aimed at air quality monitoring was designed and implemented. More specifically, the framework captures the evidence of reception, processing and storage after the reception of each uplink, without any modification of the protocol or of the payload of the sensing nodes.

3. A formal model of provenance and integrity was developed. The model defines stable identifiers for devices, uplinks, gateway receptions, measurements and provenance records, and represents the lifecycle on the platform side by means of three activities (i.e., RX, INGEST and STORE). In particular, the deployment produced 135,000 RX, 135,000 INGEST and 133,900 STORE records.

4. A method for tamper-evident storage, based on hash-linked chains per device and checkpoints of the chain heads, was developed. The verification of 403,900 records took 5.87 s (i.e., 0.0145 ms per record), and all five tested types of change were detected. Among them, the truncation of the chain tail was detected by means of the comparison of checkpoints in 0.94 ms.

5. A method for the on-demand reconstruction of graph views in the PROV style was developed. The method supports both backward tracing, from a stored measurement to its source uplink, and forward tracing, from an uplink to its stored records, without maintaining a permanent global graph.

6. The framework was evaluated by means of a 12-month deployment with three sensing nodes and two gateways, which produced 135,000 logical uplinks, 133,900 stored measurements and 403,900 provenance records. The effective completeness reached 84.9%, and the database occupied 164.08 MB, of which 114.14 MB corresponded to the provenance records.

7. A paired benchmark of 5,000 logical uplinks over 10 runs was performed. According to the results, the capture of provenance increased the median processing latency from 1.565 ms to 2.189 ms per uplink (i.e., by 0.62 ms).

8. The method for the attribution of missing measurements at the level of each stage was evaluated by means of simulations, with a radio layer based on ns-3 and calibrated to the deployment. Over 20 runs of one year each, only 3 out of 270,107 attributed measurements were assigned to the wrong stage. Moreover, the coverage reached 66.4% (93.7% without the re-joins of devices), and the accuracy decreased to 96.5% when the event log of the network server was not available.

The theoretical and practical significance of the work.

The theoretical significance of the dissertation lies in the proposed provenance model for monitoring systems based on LPWAN technologies. The model represents the lifecycle of IoT measurements on the platform side, from the reception of the uplink to the storage in the database, and combines the capture of provenance, the verification of its integrity and the attribution of missing data. Moreover, the proposed model can be used as a basis for further research regarding verifiable data lineage and data completeness in IoT environments with constrained resources.

The practical significance of the work lies in the provenance-aware LoRaWAN prototype implemented for air quality monitoring in smart cities. The prototype operates on the platform side, without any modification of the LoRaWAN protocol or of the payload of the sensing nodes, and allows the analysis of missing measurements, decoding failures and violations of the integrity of the provenance chains. In particular, the results show that the event log of the network server must be retained independently of the platform collector, since otherwise the failures of the platform cannot be distinguished from the losses on the radio path.

Connection of this work with other research projects. The research presented in this dissertation was started within the framework of the project *"Software and hardware complex for the analysis and monitoring of climatic and environmental changes of the environment*" (grant No. AP05134597-OT-20 of the Ministry of Education and Science of the Republic of Kazakhstan, 2018 - 2020). Furthermore, the continuation of the research will be carried out within the framework of the project for young scientists "Zhas Galym" (grant No. AP32523239, 2026 - 2028).

Approbation of work. The main statements and scientific results of the dissertation have been presented and discussed at the following conferences:

1. seminars of the Department of Computer Engineering at the International Information Technology University, Almaty, Kazakhstan;
2. the 5th International Conference on Smart Information Systems and Technologies (SIST 2025), Astana, Kazakhstan;
3. the X International Scientific Conference "Mathematics, Mechanics and Information Technologies" (ICMMIT 2025), al-Farabi Kazakh National University, Almaty, Kazakhstan;
4. a seminar at Astana IT University, Astana, Kazakhstan (August 28, 2026).
5. the International Conference on Digital Technologies and Applications (ICDTA 2026), Vaughn College of Aeronautics and Technology, New York, USA (October 9-10, 2026).

The results of the dissertation have been implemented at SDU University (Kaskelen, Kazakhstan), as confirmed by the Act of Implementation presented in Appendix E.

Scientific publications. The results of the dissertation have been published in 6 scientific works, namely 3 papers in journals recommended by the Committee for Quality Assurance in Science and Higher Education of the Ministry of Science and Higher Education of the Republic of Kazakhstan (MSHE RK), 2 papers in journals indexed in Scopus and Web of Science (WoS), and 1 paper in the proceedings of an international conference.

1. **B. Imankulova**, W. M. S. Yafooz, K. Alshammari, B. Alshemali, A. E. Yahya and A. Abd Almisreb. Tracing Data Origins in Smart Cities: An IoT Perspective // IEEE Access. 2025. Vol. 13, pp. 130201-130214. DOI: 10.1109/ACCESS.2025.3591277 [WoS Q2, Scopus percentile: 91, Cite Score(2025): 9.3, available: https://ieeexplore.ieee.org/document/11087496]
2. Mukash Zh., Bakranova D., Serikkanov A., Bakranov N., **Imankulova B.** Energy-Aware AI-Enabled Intelligent Sensing Systems for IoT and Edge–Cloud Architectures. Sensors 2026, Volume 26, Issue 21, 6927, 10.3390/s26216927 [WoS Q2, Scopus percentile: 83, Cite Score (2025): 9.4]
3. **B. Imankulova**, A. Sagymbekova. Privacy Challenges in Environmental IoT Systems: A Case Study on Air Quality Monitoring, 2025 IEEE 5th International Conference on Smart Information Systems and Technologies (SIST), Astana, Kazakhstan, 2025, pp. 1-6, doi: 10.1109/SIST61657.2025.11139338 [Scopus]
4. **Imankulova B.B**. Privacy challenges and framework design for IoT devices: A comprehensive approach. Bulletin of AUPET. 2025. Vol. 68, Issue 1. p. 256–267. DOI: https://doi.org/10.51775/2790-0886_2025_68_1_256 [KOKHBO]

5. **Imankulova, B. B**., Kozina, L. A., Duzbayev, N. T., Amanzholova, S. T. Transmission, storage and preprocessing of data received from sensors using LoRa technology. Bulletin of AUPET. 2024. Vol. 64, Issue 1. p.16-24. DOI: https://doi.org/10.51775/2790-0886_2024_64_1_16 [KOKHBO]

6. **Imankulova B.B.**, Amanzholova S.T., Duisebekova K.S. Software and technical aspects of data collection from end nodes: problems and prospects. Bulletin of AUPET. 2020. Vol. 51, Issue 4. p.164-172. DOI: https://doi.org/10.51775/1999-9801_2020_51_4_164 [KOKHBO]

Chapter overview

In Chapter 1 we review the state of the art regarding IoT data provenance for smart city systems. The chapter examines the concepts of provenance, the requirements of smart city data, the representation models, the integrity mechanisms and data completeness, and presents a systematic literature review conducted according to PRISMA 2020. As a result of the review, four research gaps and five requirements for a provenance-aware LoRaWAN platform are derived.

In Chapter 2 we present the design of the proposed framework. The chapter describes the design principles and the layered architecture, the integration with the LoRaWAN communication stack, as well as the threat model and the precise meaning of the security claim. Afterwards, the formal model of the framework is presented, namely the system components and identifiers, the provenance records and graph, and the models for the generation of provenance, the verification of integrity and the attribution of missing data. At the end of the chapter we describe the storage and verification of provenance.

In Chapter 3 we describe the implementation of the framework, including the platform architecture, the field deployment, the ingestion via MQTT and the capture of provenance, the storage schema, the procedure of integrity verification, the provenance queries and the procedure of attribution of missing data.

In Chapter 4 we evaluate the framework by means of a 12-month deployment, in terms of data completeness, link quality and Adaptive Data Rate (ADR) behaviour, storage and processing overhead, integrity verification and traceability. Furthermore, the attribution of missing data is evaluated against known causes by means of simulations with a LoRaWAN radio layer based on ns-3. At the end of the chapter a comparison with the closest studies and a discussion of the limitations were added.

Finally, the Conclusion summarizes the main findings and contributions of the dissertation, also outlines the limitations of the proposed framework, identifies the directions for future research.

Structure and volume of the thesis

The dissertation consists of 155 pages, including 34 figures, 22 tables, a list of 171 references and 4 appendices.

1 RESEARCH BACKGROUND AND LITERATURE REVIEW

1.1 Air-quality monitoring in smart cities and LPWAN constraints

Today, in large cities air pollution has become one of the major problem and it has adverse effects on human health [13]. Current studies in Kazakhstan related to the air quality research, especially including big cities such as Almaty, and Astana showed the presence of elevated levels of fine particulate matter. The increase in concentration occurs mainly during the heating season due to coal combustion, as well as due to traffic and unfavorable weather conditions [14], [15], [16]. In particular, the environmental conditions of Central and Western Kazakhstan were studied using complex methods and data obtained from various sources [17], [18]. Current research shows a link between elevated PM2.5 levels and respiratory and cardiovascular effects [19]. Therefore, further air-quality monitoring should be expanded for the purpose of the implementation of the effective environmental protection policy [20], [21]. At the same time, air pollution forecasting methods have been developed based on ensemble machine learning models and real-time monitoring systems [22], [23]. In Central Asia, issues such as resilience to climate change and the use of Internet of Things technologies in environmental monitoring are becoming relevant [24].

Recently, the Internet of Things (IoT) has become one of the technological elements of smart city systems. Urban infrastructure can include sensors to measure environmental conditions and monitor various operations in different parts of the city [25]. Thus, the collected data can be used for environmental monitoring, transportation management, utility system control, and decision-making in urban management. In smart city ecosystems, real-time data is used to manage transportation [26], [27], energy [28], [29], healthcare [30], environmental monitoring [31], security and privacy [32], [33] and other urban services [34]. Modern IoT platforms are used to monitor device status, remotely manage infrastructure, and control connected devices [35,36].

Reference stations provide accurate measurements, but they are very expensive and are few in each city. Sensor networks are a complementary technology and provide more detailed data in space and time [31], [37]. They also allow for monitoring pollution at the street and district levels. Such networks are only useful if their data can be trusted. Inexpensive sensors have a higher risk of drift and failure than reference devices. Trust in such data for that reason requires both valid measurements and traceable data handling; this dissertation addresses the latter.

The perception layer forms the physical foundation of the IoT system, and consists of sensors, actuators and embedded devices. Most often, the devices of this layer are used to collect environmental data and to interact with the physical infrastructure (e.g., air quality sensors, traffic monitoring units, smart meters, CCTV cameras, etc.). Above the perception layer, the network layer is responsible for the transmission of the collected data from the sensors towards the processing and storage components. Different communication technologies can be used for this purpose, such as LoRaWAN, 5G, Wi-Fi and Bluetooth Low Energy (BLE). In this

sense, the choice of the appropriate technology depends on the required communication range, the data rate, the power consumption and also the deployment conditions.

The data processing layer is in charge of the storage, filtering, transformation and analysis of the collected data. Typically, the tools associated with this layer include cloud platforms, edge computing systems, databases as well as with data analytics tools. As a result, the raw data are transformed into structured also useful information. Then, the application layer providing specific services, allows users to interact with the system. In particular, such services are related to traffic and waste management, environmental monitoring, emergency response, etc. In summary, the above layers support the coordinated collection, transmission, processing and use of data in smart city systems.

LPWAN technologies for urban sensing. Deploying large numbers of battery-powered sensors throughout a city requires a wireless connection with long range, low cost, and low power consumption per device. For this purpose, LPWAN (Low Power Wide Area Networks) networks were developed [38]. Among them, the most widely used are NB-IoT, LoRaWAN, and Sigfox [39], [40]. For municipal and research deployments LoRaWAN could be attractive with its two properties. First, it operates in unlicensed spectrum, and second, it allows an organisation to run its own network infrastructure [41], [42]. Table 1.1 compares the main wireless communication technologies based on the reviewed studies.

Table 1.1 - Comparison of wireless communication technologies for air-quality monitoring

Criterion	Wi-Fi	Zigbee	NB-IoT or LTE-M	LoRaWAN
1	2	3	4	5
Communication range	Short to medium (10 - 100 m)	Short (10 - 100 m)	Wide (cellular coverage)	Long-range outdoor coverage (2 -15+ km)
Energy consumption	High (hours - days)	Low (month - years)	Medium (days - months)	Very low (years)
Data rate	Medium or high (tens-hundreds Mbps)	Small local packets (up to 250 kbps)	Small or medium (tens of kbps to 1 Mbps)	Low (0.3 - 50 kbps)
Infrastructure control	Local, but limited range	Requires dense local mesh	Operator-dependent (SIM card, subscription)	Can use private local server
Suitability for distributed outdoor sensors	Limited	Limited	Good, but operator-based	High

Continuation of Table 1.1

Criterion	Wi-Fi	Zigbee	NB-IoT or LTE-M	LoRaWAN
1	2	3	4	5
Relevance for provenance study	Less focused on LPWAN constraints	Local-network oriented	Cellular dependency and cost	Strong fit for LPWAN provenance

As shown in Table 1.1, each communication technology offers a range of advantages for specific deployment conditions. Regarding the air quality monitoring scenario under consideration, LoRaWAN strikes the most appropriate balance, since the scenario incorporates geographically distributed outdoor sensor nodes, small volumes of data and a requirement for a long battery life. More specifically, the long communication range and the low power consumption of LoRaWAN reduce the need for a dense communication infrastructure and for frequent maintenance of the sensor devices. Furthermore, unlike cellular technologies, LoRaWAN can operate through a privately managed gateway and a local network server, which allows for more control over the reception, processing and storage of data.

LoRa (Long Range) is a radio modulation technology for Low Power Wide Area Network (LPWAN) developed by Semtech [43]. LoRa uses Chirp Spread Spectrum (CSS) technology. It operates in unlicensed sub-GHz bands (433, 868, and 915 MHz).

The range and throughput of LoRa depend on key parameters such as Spreading Factor (SF), Coding Rate (CR), Transmission Power (TP), Bandwidth (BW), and Carrier Frequency (CF).

One of the key parameters of LoRa modulation is the Spreading Factor (SF), which determines the duration of a single symbol. The SF value from 7 to 12 in LoRaWAN, and the symbol duration is calculated using the following formula [44]:

$$T_{sym} = \frac{2^{SF}}{BW} \tag{1.1}$$

where SF is the spreading factor and BW is the bandwidth.

Increasing the spreading factor increases the number of chips used to represent a single symbol. For example, with SF7, one symbol consists of 128 chips, while with SF11, this number reaches 2048. Consequently, increasing SF leads to a decrease in the data rate and an increase in packet transmission time. For a 125 kHz channel and coding rate 4/5, the transmission time of a short LoRaWAN frame (about 36 bytes) increases from about 0.08 s at SF7 to about 2 s at SF12.

The coding rate (CR) is the ratio of information bits to the total number of transmitted bits when using forward error correction (FEC). For LoRa modulation, the CR value is calculated as $4/(4+n)$, where n takes values from 1 to 4, corresponding to coding rates from 4/5 to 4/8 [45]. For example, to correct errors at CR = 4/8, four extra bits are transmitted for every four information bits. Increasing

the number of redundant bits improves the immunity of data transmission to interference but increases the packet transmission time.

On the LoRa radio module, the transmission power is adjustable from -4 dBm to 20 dBm (in 1 dBm increments). The end device has a default transmission power of 14 dBm [46]. Increasing power also increases the signal-to-noise ratio, but also increases energy consumption [47].

The LoRa carrier frequency (CF) is the center frequency of the transmission range. The radio frequency range is an unlicensed sub-GHz frequency. The range extends from 137 MHz to 1020 MHz (or from 860 MHz to 1020 MHz depending on the specific transceiver chip, such as the Semtech SX1272 or SX1276) in 61 Hz increments [45]. For LoRa modulation, the bandwidth, for example in Europe, can be set at 125 kHz or 250 kHz. A smaller bandwidth requires more time to transmit a message. A larger bandwidth increases the data rate.

LoRaWAN is a communication protocol that operates on the Medium Access Control (MAC) layer and uses LoRa modulation at the physical layer. It defines the rules for data transmission, message structure, and communication between end devices and the network. The initial LoRaWAN specification, version 1.0, was published in January 2015 [46]. The technology stack is shown in Figure 1.1.

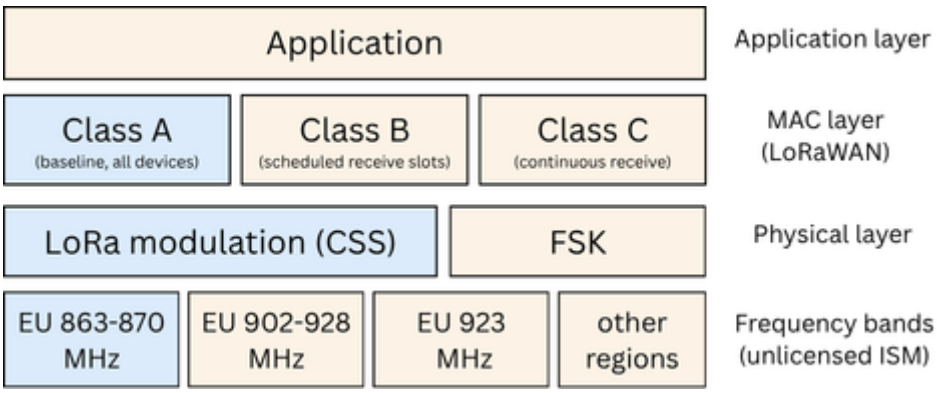


Figure 1.1 - LoRaWAN technology stack base on [48]

As shown in Figure 1.1, LoRa modulation forms the physical layer, and the LoRaWAN protocol defines a MAC layer with three device classes. Class A devices open only two short receive windows after each uplink and that being so consumes the least amount of power; Class B devices open scheduled receive slots additionally; and Class C devices are constantly listening. Sensor nodes operate as Class A devices in the EU868 band.

In the European frequency range EU863-870, the data transfer rate is determined by the Data Rate (DR) parameter, which specifies a combination of the spreading factor (SF) and the bandwidth (BW). For modes DR0-DR5, a frequency band of 125 kHz is used, with the spreading factor varying from SF12 to SF7. Mode DR6 uses SF7 with a bandwidth of 250 kHz. The data transfer rate for modes DR0-DR6 ranges from 0.25 to 11 kbps. In addition, a DR7 mode with FSK modulation is provided, providing a data rate of 50 kbps [44].

Each LoRaWAN uplink, in addition to the encrypted application payload (FRMPayload), contains a frame header with the device address (DevAddr), frame

control flags, and a 16- or 32-bit uplink frame counter (FCnt), followed by the port number (FPort) and the message integrity code (MIC) [46]. The network server uses the MIC and FCnt to authenticate frames and reject replays. The frame counter is incremented by one with each new uplink and reset to zero when the device reconnects to the network via the OTAA procedure. These fields are used by the proposed framework in three ways: the device ID defines the chain of provenance for each device, the frame counter contributes to the uplink ID and allows for attribution of missing measurements, and the MIC limits the integrity protection provided by LoRaWAN until provenance capture begins on the platform side.

A LoRaWAN network has a star-of-stars topology [49]:

- end devices transmit uplink frames by LoRa radio;
- any gateway within range receives the frame and forwards it to the network server, also it sends reception metadata, such as signal strength and signal-to-noise ratio;
- the network server removes duplicates received by several gateways, checks the integrity and frame counter of each frame, and passes the payload to an application server or an integration, typically over MQTT;
- the application platform decodes the payload, validates the values, and stores the measurements.

Thus, one measurement passes through at least five different software components or running different software before it reaches the database.

The properties that make LoRaWAN energy efficient also limit the capabilities of the device and radio channel. In most regional frequency plans, transmissions are limited by a duty cycle, for example, 1% in most EU868 sub-bands. Frame transmission time increases rapidly with the spreading factor, so each additional byte of payload can reduce the number of messages a device can send and increase its power consumption [49]. The maximum application payload depends on the data rate and ranges from 51 to 222 bytes in the EU868 plan [50]. There is a limit on metadata beyond the measured values. Links are typically not acknowledged, and the device is not notified of frame loss. The same frame can be received by multiple gateways. This can help improve reliability, but creates duplicates that must be removed, and reception data from each gateway may be lost after deduplication.

LoRaWAN provides message integrity and encryption between the device and the network, as well as application servers [51], [52]. These mechanisms protect data in transit. They do not document what happens to the data after it leaves the network server, nor do they protect records stored on the application platform.

Long-term evaluations of LoRaWAN deployment in cities show that reception quality and loss vary significantly depending on location, distance, and gateway placement [53]. Practical deployment in smart cities faces additional challenges of coverage, interference, and network management [54].

Together, these properties define three challenges for the management of LoRaWAN monitoring data:

- in LoRaWAN monitoring gaps in measurement data are common, and their causes are not visible from the data itself;

- additional provenance metadata or cryptographic signatures require extra payload space, airtime, and energy;
- the information about what happened to a measurement is distributed over the gateways, the network server, the integration layer, and the application platform. Some of this information may be lost during standard operations, such as deduplication.

1.2 Trustworthiness of sensor data and the concept of data provenance

The air quality value on the dashboard is the endpoint of a chain of operations. That is, measurement by the device, radio transmission, reception by gateways, processing by the network server, delivery to the application platform, decoding, verification, and storage. Before such a value is used for decision-making, for example, to alert a public health emergency or assess an emission source, the user may need to answer three questions:

1. Where does the value come from, and what happened to it on the way? This question concerns the origin of the measurement and the operations applied to it. It is answered by the provenance of the data.
2. Has the record of the value been changed after it was stored? This question concerns the integrity of the stored records, including the provenance records themselves.
3. Why is a value missing? This question concerns the completeness of the data. In a network without guaranteed delivery, missing values are a normal condition, not an exception.

The three questions correspond to three dimensions of trustworthiness of sensor data. Each requires precise definition; they are related but not interchangeable.

The concept of provenance is broader than origin. Origin identifies the device and the place where a measurement was produced. Provenance also documents the activities that transformed or moved the data, the agents responsible for them, and the order in which they occurred [55]. Thus, origin is one aspect of provenance, not its synonym.

Record integrity is different from measurement validity. Integrity, in the sense have been used in this dissertation, means that changes to stored records can be detected. It does not guarantee that the physical measurement was correct when it was taken. A record may be perfectly intact and still describe a value distorted by sensor drift, poor calibration, or local interference. Measurement validity is a separate data-quality dimension, addressed by calibration and quality-control methods [56], [57]. It is beyond the scope of this dissertation.

In the case of conpleteness it requires an explanation, not only a count. The proportion of missing values can be measured relative to the expected reporting schedule. However, for an operator, the cause of the loss is as important as its size. For example, a different approach is required if values are lost in a radio channel than if values are lost in data ingestion software.

Of the three dimensions, provenance is the one that connects the other two. A provenance record documents what happened to a measurement, and its integrity

determines whether that documentation can be trusted. The absence of expected provenance records is the starting point for explaining missing data. The remainder of this section introduces the concepts and models of data provenance that the rest of the chapter builds on.

Data provenance: definitions and types. Data provenance, also called lineage or pedigree, is information about the origin of a data item and the process by which it was produced [58]. This concept has been developed in three research communities:

- In database research distinguishes three types of provenance for query results [59]: why-provenance (which source records justify the result), where-provenance (from which locations the values were copied), and how-provenance (how the source records were combined).
- In scientific workflows use provenance to make computational results reproducible and to explain how a data product was derived from its inputs [58].
- As for operating systems and storage, they automatically collect provenance, as a by-product of file and process operations [60].

In IoT systems, data provenance describes the origin of sensor data, which devices and components processed it, how it was processed, and how it was transmitted between system components [61], [62]. A provenance record contains metadata about data collection, processing operations, transformations, as well as exchanges between components [63]. This information supports several uses: tracing a value back to its source, identifying inconsistencies, assessing whether data were processed according to defined procedures, and reconstructing events after an incident [12], [64]. Figure 1.2 shows the IoT data lifecycle with integrated provenance information.

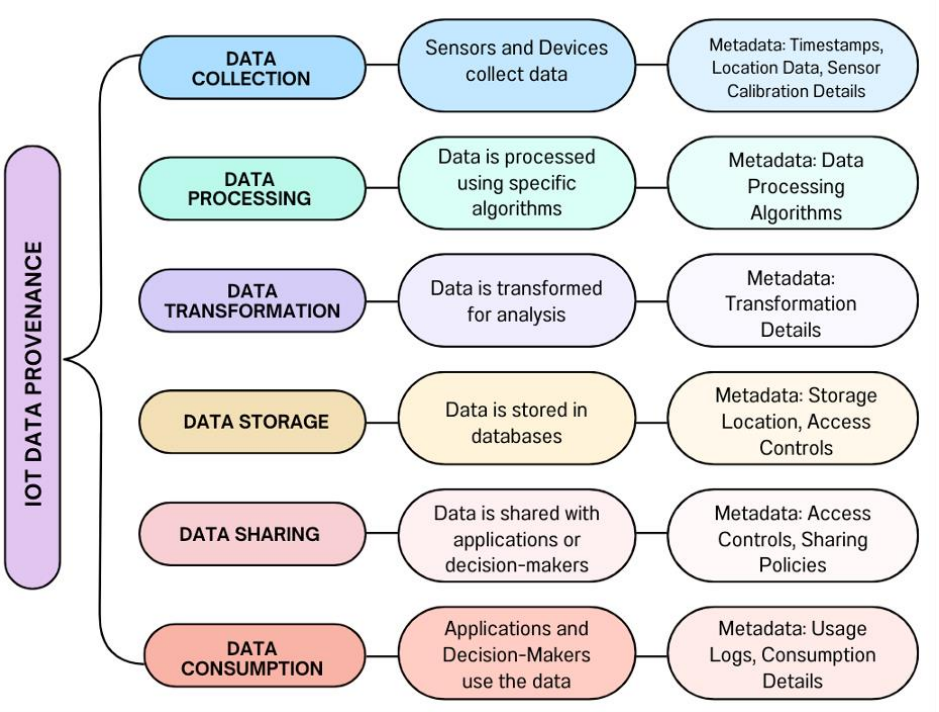


Figure 1.2 - Conceptual diagram of IoT data provenance [12]

Figure 1.2 shows the IoT data lifecycle with provenance information captured, stored, retrieved, and protected at each stage.

Provenance models: from OPM to W3C PROV. Provenance are interoperability when represented in a common model. The Open Provenance Model (OPM) was the first community model. It represented provenance as a graph of artifacts, processes, and agents connected by causal dependencies [65]. OPM has been replaced by the W3C PROV family of specifications, which are now reference standards [55], [66].

- The PROV Data Model (PROV-DM) is built around three core types, namely entities, activities and agents (Figure 1.3). An entity represents a data item or a physical thing, such as a raw radio frame, a decoded measurement or a stored record. Moreover, an activity is something that occurs over a period of time and acts upon entities (e.g., reception, decoding, validation or storage). Last, an agent is a device, a software component, an organisation or a person that bears responsibility for an activity or an entity, such as a gateway, the network server or the platform collector in the case of LoRaWAN.

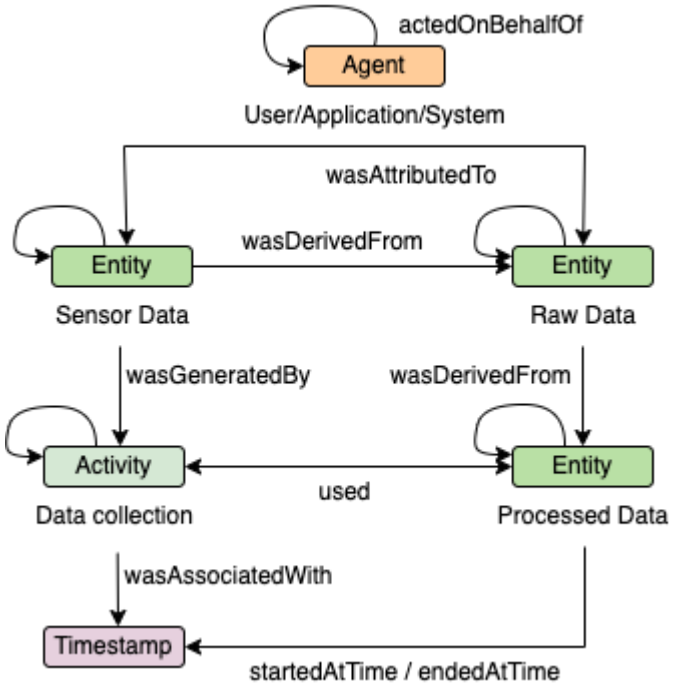


Figure 1.3 - Structure of a provenance data model [12]

Figure 1.3 shows the general provenance model: entities, activities, and agents are connected by relations that describe how data are generated, transformed, transferred, and used.

PROV-DM defines a small set of relations between these types:

- *wasGeneratedBy* links an entity to the activity that produced it;
- *used* links an activity to the entities it consumed;
- *wasDerivedFrom* links one entity to another from which it was derived;
- *wasAssociatedWith* and wasAttributedTo link activities and entities to responsible agents;
- *wasInformedBy* links one activity to another that supplied it with an entity.

Each activity may carry start and end times. These times record when data passed through each stage of processing. PROV-DM is expressed in several serialisations, including the PROV-O ontology for RDF-based systems [66]. Bundles allow provenance to be described about provenance itself.

A PROV record of a measurement is a small directed graph. Read backwards, it answers our first question, from which transmission and device the measurement was derived. Read forwards, it shows which records were derived from a given transmission. The standard fixes the vocabulary, but not the granularity of the record or the point in the data path where it is captured. These choices are left to each system, and Section 1.4 shows that they vary widely in IoT.

Previous research has identified several unresolved issues related to data provenance in the IoT [12,61,62], namely:

- Capture (ingestion). Data is transformed into multiple entities as it moves through the system. So, the system must decide which events to record and when, and the overhead must be kept acceptable for devices and networks with limited resources.
- Storage. Each change in the data creates a new provenance record, so the archive can grow faster than the data it describes. Therefore, the storage must retain the necessary history without an excessive volume.
- Retrieval. Users need both simple look-ups and traversals of lineage relationships (i.e., from a stored value back to its source, and vice versa), which requires suitable mechanisms for indexing and querying.
- Security and privacy. Provenance records may reveal device identifiers and the network structure, even when the sensor values are encrypted. Therefore, the records must be protected against unauthorised access and modification, since forged provenance can disguise manipulated data.

Provenance records are used to describe the origin of data. If these records can be altered without detection, they do not provide trust in the data. Ensuring the integrity of provenance records is an important requirement. Provenance allows tracking only the data that has entered the system. Missing data can only be identified by comparing the received data with the data expected from the reporting schedule.

The systematic literature review described in the following section examines how existing studies address these three aspects: provenance capture and representation (Section 1.4), the integrity of provenance records (Section 1.5), and data loss and completeness (Section 1.6). The results of the remaining gaps and the resulting requirements are summarized in Section 1.7.

1.3 Methodology of the systematic literature review

This review updates and extends our earlier systematic review of data provenance in IoT-based smart city systems [12], which covered publications from 2018 to 2024 and included 41 studies. The update was required for two reasons. First, the field has developed rapidly since 2024, and a substantial share of relevant studies was published in 2025–2026. Second, the framework proposed in this dissertation combines provenance capture with tamper-evident storage and the attribution of

missing data in LoRaWAN networks. The search strategy was rebuilt and extended to adjacent research areas. The review was conducted and reported in accordance with the PRISMA 2020 statement [67].

The review addressed four questions:

- RQ-L1. How is data provenance captured, represented, and stored in IoT, sensor network, and smart city systems?
- RQ-L2. Which mechanisms are used to make provenance and log records tamper-evident, and what are their performance costs?
- RQ-L3. What is known about packet loss and transmission conditions in real-world LoRa/LoRaWAN deployments, and how is missing or incorrect data identified in the literature?
- RQ-L4. What gaps remain where provenance, integrity verification, and completeness must work together in LPWAN-based environmental monitoring?

The search was conducted in Scopus and the Web of Science Core Collection. The databases index the main peer-reviewed journals and conference proceedings in computer science, the Internet of Things, and sensor networks. The search was organized into seven thematic blocks (Table 1.2). The core of the review is Block B1, which focuses on data provenance in the Internet of Things and related environments. The search queries from the previous review [12] were expanded to include terms in two ways: using synonyms for data provenance capture and using related computing environments, namely, cyber-physical systems and edge and fog computing. The Scopus query for Block B1 was as follows:

TITLE-ABS-KEY ("data provenance" OR "provenance data" OR "data lineage" OR "provenance tracking" OR "provenance capture" OR "provenance model" OR "secure provenance" OR "provenance-based" OR "provenance-aware") AND TITLE-ABS-KEY ("Internet of Things" OR "IoT" OR "sensor network*" OR "smart cit*" OR "cyber-physical" OR "edge computing" OR "fog computing") AND PUBYEAR > 2017 AND PUBYEAR < 2027 AND DOCTYPE (ar OR cp OR re) AND LANGUAGE (english).*

An equivalent query was used in Web of Science, with the field tag TS, *PY=(2018–2026), DT=(Article OR Conference Proceedings OR Review), and LA=(English)*. Blocks B2–B7 covered related areas necessary for positioning the proposed framework. Full search strings for all blocks are provided in Appendix A.

Table 1.2 – Search blocks of the systematic review

Block	Focus	Period	Records (Scopus)	Records (WoS)
1	2	3	4	5
B1	Data provenance in IoT, sensor network, smart city, cyber-physical, edge and fog systems	2018-2026	313	220
B2	Secure and tamper-evident logging (hash chains, MACs, forward security, authenticated data structures)	no lower year limit	68	42
B3	Ledger and verifiable databases and their	2015-	80	71

Continuation of Table 1.2

Block	Focus	Period	Records (Scopus)	Records (WoS)
1	2	3	4	5
	performance	2026		
B4	Secure provenance and provenance integrity	2018-2026	54	38
B5	W3C PROV and provenance models for sensor data	no lower year limit	43	34
B6	Packet loss and transmission conditions in LoRa or LoRaWAN deployments	2018-2026	13	8
B7	Detection and diagnosis of missing and faulty sensor data	2018-2026	74	143
Total			645	556

The initial query for Block B3 returned a large number of records on database systems unrelated to IoT or auditing (Scopus 149, Web of Science 113). Therefore, the same context filter (*audit*, provenance, "data integrity," IoT, "Internet of Things," sensor*, performance, benchmark*)* was applied to both databases, reducing the number of results to 80 and 71 records, respectively. For blocks B2 and B5, there was no publication year restriction, as the fundamental work on secure logging and data provenance models was conducted before 2018. Research published before 2018 that was retrieved through these queries and met the selection criteria was included in the review.

Eligibility criteria. The study included peer-reviewed journal articles, conference papers, and reviews written in English. Articles met the criteria for the relevant section and did one of the following: propose and evaluate a method, describe an implemented system, or systematically address the section's topic. During the title and abstract analysis stage, articles were excluded for the following reasons:

E1 - inappropriate document type (editorial, keynote address, seminar summary, or retracted article);

E2 - full text not in English (no records were excluded for this reason, because the language filter was applied in the search queries);

E3 - application domain outside the scope of the review, without transferable results (for example, clinical systems, finance, or supply chains without sensor data);

E4 - the search terms matched, but the record did not address the block criterion;

E5 - no method or evaluation (conceptual position only).

At the full-text stage, the reasons for exclusion were:

FT1 - no implemented system or evaluation;

FT2 - the block criterion was not addressed in substance;

FT3 - the setting was not transferable to IoT or sensor data platforms;

FT4 - earlier or duplicate version of an included study;

FT5 - insufficient technical detail for comparison.

Selection process. Exported records from both databases were merged and duplicates were removed in Zotero.

Title and abstract screening were performed with the support of a large language model (NotebookLM, Google; Claude, Anthropic). For each record, the model proposed a decision and a justification against the block criterion, and the final decision was made by the author. Large language models have been shown to achieve high sensitivity in abstract screening for medical systematic reviews, although their precision varies between reviews and domain-specific validation is required [68]; the model was therefore used to support, not to replace, the author's decisions. Full texts were assessed by the author, all extracted values were verified by the author.

Reports that could not be retrieved due to access restrictions were marked as unretrieved and were not considered excluded. If a study was published in multiple versions (e.g., a conference paper and its extended version in a journal, or the same study in two publications), only the most complete version was included, and the remaining versions were excluded with the code FT4. Regulatory standards and technical documentation not considered in the PRISMA algorithm were used as reference sources. These include the W3C PROV-DM and PROV-O recommendations, IETF RFC 6962 and RFC 9162, the LoRaWAN specification, and vendor documentation.

Data extraction and synthesis. For each included study, the following data were extracted with a structured form:

- study type and application domain;
- evaluation setting (real deployment, testbed or prototype, or simulation) and network technology;
- point of provenance capture (device, gateway, broker, platform, or cloud), provenance model, granularity of a provenance record, and storage of provenance;
- integrity mechanism, trust anchor, and external anchoring or third-party verification;
- stated threat model, attackers considered, and tampering cases evaluated;
- handling of missing or lost data and whether changes to the devices are required;
- evaluation metrics and main quantitative results;
- limitations stated by the authors.

Results of the selection. The selection process is shown in Figure 1.4. The database searches identified 1,201 records (Scopus 645, Web of Science 556). After the removal of 481 duplicates, 720 records were screened by title and abstract, and 502 of them were excluded. Of the 218 reports sought for retrieval, 66 could not be obtained because of access restrictions. 66 papers out of 152 full-text reports on the evaluation of the selection criteria for compliance were excluded. A total of 90 studies were identified as a result of the database search.

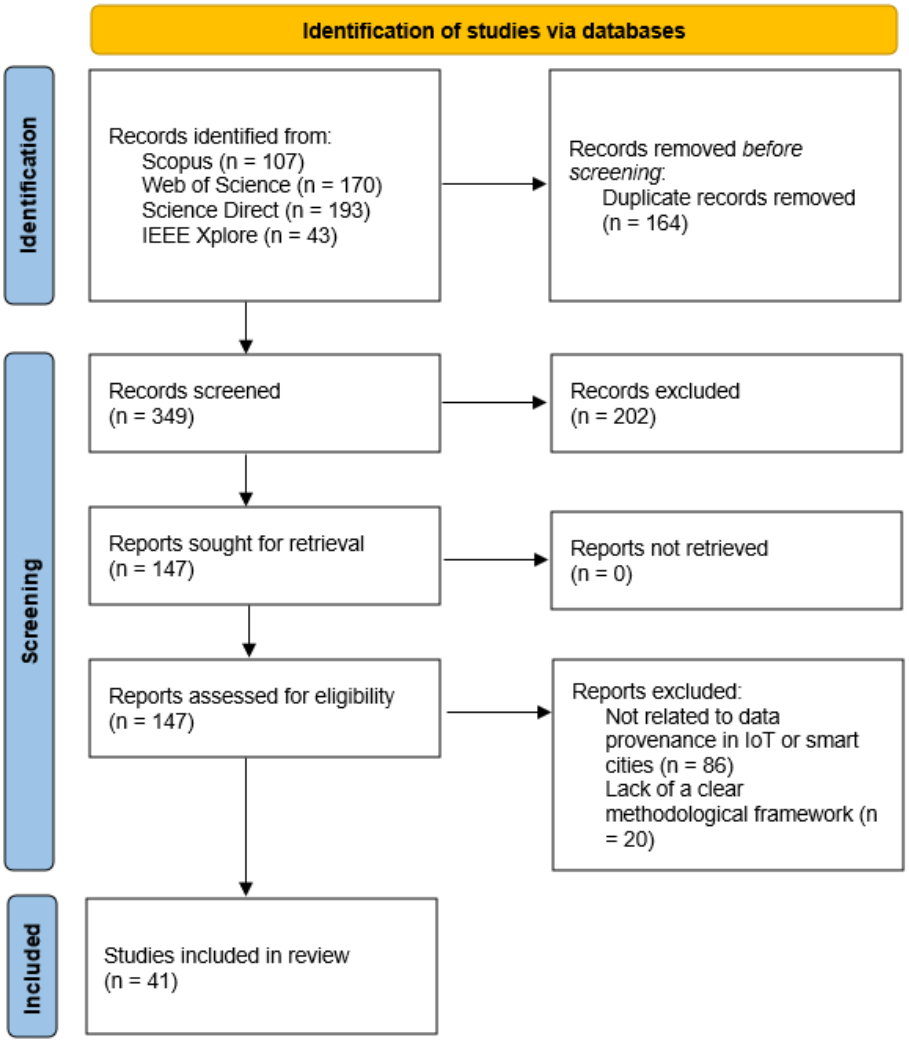


Figure 1.4 – PRISMA 2020 flow diagram of the study selection

The review has some limitations, such as sixty-six reports identified through the database searches being unavailable due to access restrictions. Therefore, some relevant data, including several closely related studies published in 2026, may be missing from this review.

1.4 Provenance capture and representation in IoT systems

This section addresses the first review question (RQ-L1): how is data provenance information captured, represented, and stored in Internet of Things (IoT), sensor networks, and smart city systems? The review collects a number of studies that propose, implement, or evaluate data provenance tracking mechanisms for IoT or sensor data. The studies are compared across four parameters: the point in the data flow where provenance is captured, the representation model, the granularity of the provenance record, and the storage of provenance information.

Data provenance can be recorded at various points along the data path, from the sensor to the database. The data collection point determines two things: which processing stages can be used to describe the data's provenance, as a record can only document what happens at or before its creation, and whether devices require

modification to obtain it. The studies reviewed are divided into five groups based on data collection points (Table 1.3).

Capture on devices and forwarding nodes. In multi-hop sensor networks, provenance is usually recorded as the path of a packet from its source to the base station. Each forwarding node adds a compact encoding of its identity, and the base station reconstructs the path. Research in this group focuses on reducing the number of bytes added to each packet:

- arithmetic path encoding requires 4 - 10 bytes per packet [69];
- incremental refinement based on the learned topology requires about 2 bytes [70];
- index-based compression decodes paths with virtually no errors and identifies malicious forwarders with 78 - 92% accuracy [11];
- zero-based watermarking adds 24 bytes per hop and also checks data integrity [71];
- Bloom filter encoding requires 16-30 bytes per packet and can detect packet loss on individual links [72], the same idea was applied to IoT networks with a 4 KB filter [73];
- other path-based schemes are described in [74], [75], [76].

The second line of work establishes the provenance of measurements based on the physical properties of the device and radio channel. Wireless channel fingerprints and physically unclonable functions are used to associate data with a specific device and location [10,77,78]. Using a 64-byte fingerprint, false positives and missed detections were reduced to zero. A similar approach identifies the transmitting device based on changes in its internal clock with an accuracy of over 99% [79]. These schemes verify the origin of the measurement, but not what happens to it after reception [79]. Software PUFs combined with zero-knowledge proofs provide origin verification with privacy [80], [81] and an FPGA implementation of the same model performs authentication in about 40 μs on the server and 80 μs on the device. For synchrophasor networks, a PUF-based provenance scheme intended for the quantum era adds 256/n bits per packet over a window of n packets and about 0.1 ms of execution [82]. In most cases, the study group requires changes to the device firmware or packet formats. Most of them assume a multi-hop topology, in which the forwarding path is the primary uncertainty about the origin of the data.

Capture at gateways and edge nodes. Several studies move provenance capture to devices that aggregate data from constrained nodes:

- gateways that collect sensor batches and submit them to a blockchain through smart contracts, reaching about 3,000 transactions per second with a mean latency of 400 ms in a 100-node test [8];
- HyperProv, which records checksums, storage pointers, and operation histories in Hyperledger Fabric running on edge hardware, processing about 400 transactions per minute on a Raspberry Pi with a power draw of 2.7-3.6 W [83];
- a comparable Fabric deployment on ARM64 gateways, which reached about 12 transactions per second with a mean latency of 2.3 s [84], [85];
- a fog architecture that stores W3C PROV documents in a private blockchain and reduces storage time from 2.5 - 3 s in the cloud to 0.5 - 1 s at the fog layer [86];

- a management hub that logs data access and consent operations on behalf of sensor nodes [87];
- a cold-chain monitoring gateway that anchors hourly statistical summaries rather than individual readings, which needs only 38 blockchain transactions per day [88].

In the case of streaming data, fine-grained provenance identification links each output tuple to the source tuples that produced it. GeneaLog does this with a throughput reduction below 10% and an added latency below 46 ms [89]. Twins switches provenance recording on and off at the edge according to need [90]. An adaptive control method reduces the CPU load of provenance from 6.5 - 9.0 times the baseline for full provenance to 1.7 - 4.0 times [91].

Capture at the message broker. Data capture is performed at the message broker level. Data provenance can also be captured in middleware where messaging is expected to occur without changing devices or applications. An MQTT broker with provenance tracking capability records which subscribers received each message and publishes these records as W3C PROV streams, which adds message delivery times by 40 - 50 ms [92]. Another scheme records energy trading messages at the broker level before committing them to the blockchain [93].

Platform- or server-based data capture. Platform-based approaches [94] integrate an IoT platform, server, or analysis environment, rather than the devices themselves:

- Smart home platforms, where the provenance of API calls, control messages, and state changes are recorded as a W3C PROV graph. ProvThings reports a 5% increase in device latency, a 20.6% increase in event processing latency, and approximately 260 KB of provenance information per day [95]. ForenThings reports full provenance coverage for forensic reconstruction at approximately 22.7 MB per 1000 events [96];
- Environmental and urban data, where PROV-based provenance documentation describes the acquisition parameters, processing steps, and algorithms underlying environmental data products [97], data integration workflows for smart cities [98], and sensor data management [99], [100];
- Real-time device behavior verification [101], [102], network forensics reconstructed from captured packets [103], and auditable network event diagnostics [104];
- Device identification at the receiver from clock frequency variations in the received signal with greater than 99% accuracy and without device changes [79].

Capture across several layers (end-to-end). The largest group of studies [105], [106] capture data provenance at multiple points, from devices to the cloud. Examples include:

- last-mile food delivery [107];
- propagation of health data with security metadata [108];
- scientific workflows on the edge-to-cloud continuum, with a capture overhead below 1.6% [109];
- linking sensor readings to software versions in a deployment pipeline [110];
- building digital twins [111];

- certification of wind-resource measurements [112];
- blue carbon accounting with LoRaWAN sensors [113].

The last study is the only provenance study in the review that uses LoRaWAN. Its provenance is established on the device side: each sensor signs its measurements with ECDSA before transmission.

Table 1.3 provides a brief description of the five groups. For each data collection point, the number of studies, representative examples thereof, the type of provenance record, the declared overhead costs, and the need for modifications to the end devices are listed. Since a single study may record provenance at multiple points, the total amount of data exceeds the number of studies.

Table 1.3 - Provenance capture points in the reviewed studies

Capture point	Studies	Representative studies	Typical provenance record	Reported overhead	Device changes
1	2	3	4	5	6
Devices and forwarding nodes	12	[69], [70], [73], [80]	Packet path; origin fingerprint	2 - 30 B per packet or hop	Required
Gateway or edge	14	[83], [84], [88]	Data item; transaction	≈ 12 tx/s on edge hardware	Usually required
Message broker	2	[92], [93]	Message delivery	+ 40-50 ms per message	Not required
Platform, server or cloud	18	[95], [97], [96], [79]	API calls, events, data products	+ 5-21% latency; 0.26 - 22.7 MB per day or per 1,000 events	Mostly not required
Several layers (end-to-end)	34	[89], [107], [109], [113]	Data object across its life cycle	< 10% throughput loss (streaming); otherwise depends on the integrity mechanism	Usually required

Table 1.3 shows the tradeoff between capture point and device-side overhead. Approaches that capture data origins at devices and forwarding nodes describe the path of each packet. However, they add 2-30 bytes per packet or hop and also require changes to device firmware. Gateway- and edge-level approaches, most of which are based on distributed ledgers, avoid this payload overhead but still require registry clients or device signatures. Thus, they achieve only about 12 transactions per second on edge hardware. The largest group, which records data provenance at multiple levels, offers the most comprehensive description of the data lifecycle. However, its overhead depends primarily on the integrity mechanism and is difficult to compare across studies.

Two important observations were made regarding LoRaWAN-based monitoring. First, schemes based on routes that dominate the device-level group are of limited value in LoRaWAN. Each uplink reaches the gateways in a single hop, and the route itself contains little information. Second, none of the platform-based approaches consider gateway reception data, for example, the set of receiving gateways and their radio metadata, as part of the provenance record. Furthermore, none of them use the recorded provenance to explain missing measurements.

1.5 Integrity of provenance and log records

This section addresses the second review question (RQ-L2), which asks what mechanisms are in place to ensure protection against unauthorized modification of provenance records and logs.

Threat models and trust assumptions. The 60 articles reviewed differentiate adversaries into two groups:

- Research on IoT data provenance primarily considers external adversaries. Specifically, compromised devices or forwarding nodes that inject, modify, or replay data en route to the platform [74], [78], [71]. The platform or server that stores data provenance information is typically trusted, either explicitly [95], [108], or implicitly.

- Only a few IoT studies consider an insider controlling storage or holding keys. Examples include rational domain administrators hiding unfavorable network provenance information [104], a malicious asset owner [112], a malicious insider in an environmental monitoring system [113], and an insider holding keys in an audit trail architecture [114].

- Research in secure logging and database management assumes the opposite. The logger, database administrator, or cloud service provider are untrustworthy, and records must remain verifiable even if these parties attempt to modify them [115], [116], [117], [118], [119].

This distinction determines which mechanisms are adequate. A mechanism that protects data from external attackers on the network does not necessarily make the stored history unalterable for the operator.

Families of integrity mechanisms. Five families of mechanisms were identified (Table 1.4). Most studies combine several of them (56 out of 90 were classified as combinations).

Hash chains, MAC codes, and forward-secure logging. The oldest family links each record to its predecessor via a hash or message authentication code, so that a modified, deleted, or reordered record breaks the chain. An unkeyed hash chain can be recomputed by anyone who controls the storage. Keyed variants prevent this, and copy-protected schemes additionally change the key after each record, so that an attacker who compromises the logger cannot overwrite earlier records:

- Copy-protected sequential aggregate signatures maintain a constant size of the verification data and require approximately 5.5 ms per signature [120].

- Tree-searchable key generators reduce key change times to 0.2 - 0.8 μs [121].

In the Internet of Things, the same principle is used in MAC-protected authentication [72], [70], [122] and in a hash chaining scheme for resource-constrained sensors. This scheme verifies a signature in 0.031 ms, but verification time increases after long outages with a large number of lost packets [123].

The study closest to platform-side IoT data logging seals each telemetry record with HMAC at the edge gateway and periodically anchors the Merkle packet root data to a four-validator consortium ledger [114]. It reports a commit time of 2.5 ms on LAN and 58 ms on WAN, and a verification time of 0.9 ms for cold writes and 15 μs for re-verification.

Authenticated data structures. In Merkle trees and history trees, the verifier checks whether a record is included in the log, or a later version of the log extends an earlier one using short proofs. For example, a tamper-resistant history tree processes 1,750 - 10,500 events per second, with proofs of about 3 KB for a log of 80 million events [115]; a parallel authenticated tree based on chameleon hashes achieves about 280,000 writes per second, which is more than eight times faster than a Merkle tree [124]; Lock techniques bring Merkle trees closer to the throughput of conventional index structures [125]; Tunable transparency logs demonstrate how such structures can be optimized for various read and audit workloads [126].

Ledger and verifiable databases. These databases have the built-in ability to append only verifiable records as a centralized database feature. SQL Ledger stores transaction blocks with hash chains in Azure SQL Database and periodically exports database digests to immutable storage. It supports over 70,000 transactions per second with an additional latency of 12-42 μs per modified row, but fully validating 100 million transactions takes approximately 55 minutes [116].

Systems like LedgerDB achieve over 200,000 writes per second and anchor their state to a trusted timestamp source [117]. Hardware-assisted systems use trusted execution environments to protect against privileged host escaping, resulting in throughput increases of approximately 22% for VeDB [118] and SecuDB [127]; VeriDB adds 1.5-2.2 μs per read or write operation [128]. A systematic benchmark shows that the benchmarked databases vary by orders of magnitude in throughput and storage capacity depending on their design [129], [130]. A review of ledger databases shows throughputs ranging from approximately 600 transactions per second for Hyperledger Fabric to 100,000 for LedgerDB [131]. Immutability provided only by database privileges can be bypassed by an administrator with elevated rights [132].

Blockchain-based provenance verification. Blockchain is the most common data integrity mechanism in IoT research, and its cost varies greatly depending on the platform [133–136]. Ethereum-based frameworks report 1.0 - 1.6 transactions per second with a latency of 14 - 36 seconds [137], or about 4 seconds per transaction at a cost of $9 - 130 per month [138]. Permissioned Hyperledger Fabric deployments range from approximately 12 transactions per second on edge hardware [84,132] to several thousand in server environments [8,132], [139]; In a LoRaWAN-based environmental monitoring system, Fabric achieved 145 transactions per second with a latency of 62 ms [113].

Some solutions reduce these costs by anchoring periodic digests instead of individual records. Sending a Merkle root daily to a public blockchain costs approximately $443 per year [140], while hourly digests require only 38 transactions per day [88,132]. Epoch-based tasks of 104 bytes replace 9.1 MiB of anchoring per record, costing approximately $0.19 per measurement station per year [112], and selective anchoring of high-risk events further reduces the overhead [141].

Hardware-Based Trust and Watermarking. Physically unclonable functions bind data to a specific device without storing keys [77], [80], [82]. Secure boot and secure key storage on low-cost microcontrollers serve the same purpose [132,138]. Watermarks embed integrity information in the data itself, either at each stage [71] or in database tuples [142]. All these approaches operate at the device level and require changes to its hardware or software.

Table 1.4 provides a brief description of the five families. The manipulations they protect against, the party to trust, the cost of representing them, as well as their suitability for an LPWAN monitoring platform.

Table 1.4 - Families of integrity mechanisms for provenance records and logs

Family	Protects against	Trusted party	Representative cost	Fit for an LPWAN platform
1	2	3	4	5
Hash chain, MAC, forward-secure log	Modification, deletion, reordering of stored records; with keys or key evolution, also rewriting by an intruder	Key holder (verifier or auditor)	µs-ms per record; constant-size verification data [120,121]	High: no device changes, low cost
Authenticated data structures (Merkle, history trees)	Modification and forking of a log; efficient inclusion and consistency proofs	Auditors receiving commitments	10^3 - 10^5 entries/s; KB-size proofs [115,124]	High, combined with periodic commitments
Ledger or verifiable databases	Modification by database administrators or cloud providers	Digest storage, timestamp authority, or TEE	10^4 - 10^5 tx/s; ≈ 22% overhead with TEE [116,118]	Medium: depends on a specific database product
Blockchain (per record or anchored digests)	Modification by any single party	Consortium or public network	1- 6,000 tx/s; seconds of latency on public chains [112,137]	Low per record; medium with periodic anchoring

Continuation of Table 1.4

Family	Protects against	Trusted party	Representative cost	Fit for an LPWAN platform
1	2	3	4	5
Hardware roots of trust / watermarking	Device impersonation; modification in transit	Device hardware	µs per step, but device changes [78,80]	Low: requires device changes

As shown in Table 1.4, only hash chains and authenticated data structures combine low cost with no device modifications. They are the most suitable basis for a platform-based mechanism. Their weakness is that the proof remains under the control of the platform operator. That is, unless keyed hashing or periodic external commitments are added. Blockchains and ledger databases transfer trust to an external network or product, but at a significantly higher cost, while hardware roots of trust require changes to the devices themselves.

External anchoring and third-party verification. Thirty-two of the sixty-eight studies verify their records externally. They use a public or consortium blockchain [140], [114], a trusted timestamping authority [117], [118], immutable cloud storage for database digests [116], or commitments sent to independent auditors [115]. Among fifty IoT studies, only twenty-one provide such verification, and most of them use blockchain. In the remaining IoT studies, integrity proofs are stored on the same platform as the records. The proof reveals accidental corruption and changes made by outsiders, but not consistent reworking by the platform operator.

Frequency of linkage is a tradeoff. Anchoring each record provides the best protection at the highest cost. Periodic data updates reduce the cost by orders of magnitude, but records made since the last backup remain unprotected until the next one [112,114].

1.6 Data loss, missingness and completeness in LoRaWAN data pipelines

This section addresses the third research question (RQ-L3), which examines the state of the art in data loss in LoRa and LoRaWAN deployments and how missing or incorrect sensor data is diagnosed. Data provenance is an important issue, as a provenance record describes what happened to a received measurement. If a measurement is not received, the record is missing, and the reason for its absence must be determined. This section first describes where data loss can occur in the LoRaWAN data pipeline. It then summarizes empirical data on data loss, discusses data completeness as a data quality parameter, and introduces approaches to detecting and attributing data loss based on data provenance.

Where data can be lost in a LoRaWAN pipeline.

A measurement from a LoRaWAN end device goes through several stages before it is stored: radio transmission, reception by one or more gateways, forwarding to a network server, deduplication and verification on the network server, delivery to

the application platform (usually via MQTT), payload decoding, verification, and storage [41], [49]. A measurement can be lost at each stage for various reasons:

- Radio transmission: signal loss, interference, collisions with other transmissions, and duty cycle limitations imposed by regulatory agencies can prevent a frame from reaching any gateway. LoRaWAN channels are typically unacknowledged, so the device is not informed of the loss.
- Gateway and backhaul: a gateway failure or a backhaul failure discards all frames received only by that gateway.
- Network server: frames may be rejected as duplicates due to an invalid message integrity code or an unexpected frame counter value, for example, after a device reboot.
- Platform data reception: messages may be lost between the network server and the application, for example, when the subscriber is unavailable or a database transaction fails.
- Processing: received data may fail decoding or verification and never become a stored measurement.

The device itself may also fail to transmit data due to a power failure or reboot. The LoRaWAN frame counter (FCnt) is incremented for each transmitted uplink [49]. Thus, a gap in the received data counters indicates transmitted but not received frames, but only during a single activation session: reconnecting resets the counter.

Empirical evidence on loss in LoRa and LoRaWAN deployments. The review found only four studies reporting measured losses in real-world LoRa or LoRaWAN deployments. A synchronous LoRa mesh network for monitoring underground drainage systems showed packet error rates of 1.3% and 2.2% during 23- and 45-day field tests over distances of 170 - 1830 m [143]; forest tests showed packet loss of 0 - 0.5% over distances of 50 - 200 m with a spreading factor 12 and a single gateway [144]; a campus network for emergency coordination showed a packet delivery rate of 92.4% ± 1.3% for eight nodes over 180 days [145]; A LoRaWAN-based environmental monitoring system demonstrated packet loss of less than 1% in a controlled test [113].

Long-term evaluations of urban LoRaWAN deployments conducted outside the scope of this review confirm the same picture: losses are highly dependent on location, distance, and gateway placement [53].

All of these studies report losses as a single aggregate metric, packet error rate, or delivery ratio. None of them decompose missing data by the stage at which it was lost. None of them separates radio channel losses from losses on the network server or application platform. With the exception of one 180-day deployment [145], observation periods range from one day to several weeks.

Completeness as a data-quality dimension. In the data quality literature, completeness is one of the primary measurements of sensor data quality. It is typically measured as the proportion of expected values present in a dataset over a given period [56], [146]. Systematic mapping of IoT-based air quality monitoring systems reveals completeness among the most frequently assessed measurements, along with accuracy and timeliness [147]. Probabilistic approaches assess completeness when the expected number of observations is itself uncertain [146].

Such approaches measure the amount of missing data but do not explain the reason for their absence. They work with a finite data set and don't use data from the data processing pipeline. A completeness value of, say, 85% doesn't tell the operator whether the missing 15% should be addressed by moving the gateway, repairing the device, or updating the data ingestion software.

Provenance-based detection and attribution of loss. Most of the 90 studies do not explain why data is missing. Forty studies do not address the missing data problem at all, and six do not report how they deal with it. Thirty-nine studies only detect missing data without determining the cause; three of them also attempt to reconstruct or fill in the missing values. Only five studies determine the cause.

All five studies address the same issue: malicious nodes that lose packets in multi-hop networks. A Bloom filter for path origination detects packet loss and locates the faulty link 75-100% of the time, but cannot distinguish a single lost packet from multiple consecutive ones [72]. Index-based origination compression identifies malicious forwarders with 78-92% accuracy, but with a false positive rate of 13-40% [11]. A provenance-based trust model for UAV networks detects nodes that lose all or part of their traffic 80-96% of the time [148]. Two other studies use provenance graphs for forensic analysis of lost packets [103] and for diagnosing network errors in administrative domains [104].

These studies always look for a malicious node along the forwarding path. None of them separate radio losses from those caused by the node or platform. Neither of them is applicable to LoRaWAN, since in its single-hop star topology, the forwarding path is not the primary source of uncertainty.

Two other studies indirectly address the issue of missing data. A management architecture for IoT platforms detects component failures within 0.8 s, but it was only tested with simulated LoRaWAN traffic and does not explain missing measurements [149]. In a hash-chain-based signature scheme for resource-constrained sensors, verification becomes slower than using ECDSA after more than 2400 consecutive lost packets [123]. Fault diagnosis studies from Block B7 were largely excluded at the full-text stage, as they detect sensor faults or anomalies in measured values and do not link missing data to data processing pipeline stages.

Three observations regarding data loss in LPWAN monitoring are supported by the reviewed literature, and these observations are also identified in research studies:

1. Loss is measured but not decomposed. LoRa and LoRaWAN studies report overall packet error rates or delivery rates. They do not specify at which stage of the pipeline the data loss occurred, and most cover short observation periods.

2. Completeness is assessed at the dataset level. Data quality assessment approaches quantify how much data is missing, but they do not use pipeline data to explain the causes.

3. Origin-based attribution targets malicious nodes rather than pipeline stages. Existing attribution methods identify compromised forwarding nodes in multi-hop networks. None of the reviewed studies use origin records or LoRaWAN frame counters to distinguish between radio link loss and platform loss or from periods during which the device did not transmit data. Neither study specifies the conditions under which such a distinction is possible.

1.7 Comparative analysis, research gaps and framework requirements

This section addresses the fourth research question (RQ-L4), namely, the remaining gaps in data provenance, integrity, and completeness in LPWAN-based environmental monitoring. First, a comparison of studies most relevant to the topic of this dissertation is provided. The research gaps identified in previous sections are then articulated, and the requirements for a framework addressing these gaps are outlined.

Ten studies were selected for detailed comparison in Table 1.5. Each met at least one of the following conditions: it records the provenance of data from platform-side sensors; it combines provenance tracking with tamper evidence for IoT monitoring; it uses LoRaWAN; and it determines the cause of data loss using provenance tracking. The comparison covers the properties mentioned in the sections above. The table lists only those features or methods applied or evaluated in each study, as stated in the full text.

Table 1.5 – Comparison of the closest studies

Study	LoRa WAN	No device changes	W3C PROV	Integrity (storage / external)	Insider in threat model	Deletion, reordering, truncation tested	Missing data
1	2	3	4	5	6	7	8
[97]	–	✓	✓	– / –	–	–	–
[95]	–	✓	✓	– / –	–	partial	–
[92]	–	✓	✓	– / –	–	–	–
[107]	–	–	✓	blockchain / –	–	–	–
[108]	–	–	✓ (extended)	✓ / ✓	–	–	–
[103]	–	✓	✓	– / –	–	–	attribution (malicious drops)
[72]	–	–	–	in-packet MAC / –	–	partial (no truncation)	attribution (malicious drops)
[88]	–	–	–	hourly digests / ✓	–	–	detection
[114]	–	✓	–	HMAC + Merkle / ✓	✓	✓	detection
[113]	✓	–	–	signatures, hash chain / ✓	✓	–	detection

Continuation of Table 1.5

Study	LoRa WAN	No device changes	W3C PROV	Integrity (storage / external)	Insider in threat model	Deletion, reordering, truncation tested	Missing data
1	2	3	4	5	6	7	8
This work	✓	✓	✓	keyed hash chain (HMAC) and checkpoints / partial	✓	✓	stage-level attribution

As Table 1.5 shows, no reviewed study covers all the criteria at once. The closest study [114] provides platform-side tamper evidence with external anchoring, but it does not use LoRaWAN or W3C PROV and only detects missing records. The framework proposed in this dissertation addresses all the criteria within one platform.

Research gaps.

1. No platform-side provenance model for the LoRaWAN data path. Existing studies either modify the devices or capture provenance only above the network layer. None represents the full path of a measurement, from reception by the gateways to storage, in a standard model and without device changes (Section 1.4).

2. No lightweight tamper evidence with an explicit threat model. Strong guarantees rely on blockchains or device-side cryptography, which are costly for monitoring platforms. IoT studies rarely state whether the operator is trusted and rarely test deletion, reordering, or truncation (Section 1.5).

3. No attribution of missing measurements to pipeline stages. Loss is reported only as an aggregate rate. No method distinguishes radio loss, platform loss, and periods without transmission, or states when this distinction is possible (Section 1.6).

4. The three aspects are treated separately. Each of the closest studies addresses only one aspect (Table 1.5), and their interaction, for example the effect of lost uplinks on chain verification [123], has not been studied in one framework.

Requirements for a provenance-aware LoRaWAN framework.

These gaps lead to five requirements:

1. Complete data logging at every stage, outside of devices. Data provenance is logged at every stage, from radio reception to storage, on the platform and without changes to the devices or the LoRaWAN protocol.

2. Standard and queryable representation. Data provenance follows a standard model, includes reception data (receiving gateways and communication channel parameters), and supports bidirectional tracing.

3. Evidence of unauthorized access consistent with the stated threat model. Modification, deletion, insertion, reordering, and truncation of data are detected via a reference stored separately from records, and operator trust is explicitly stated.

4. Limited overhead. Storage and latency costs remain consistent with the reporting interval and deployment lifespan.

5. Attribution of missing data at the stage level. Each missing measurement is attributed to the stage at which it was lost or reported as unspecified if evidence does not support a single cause.

Conclusions on Chapter 1.

LoRaWAN is a suitable technology for air quality monitoring in dense urban environments. However, its limitations result in frequent data loss and leave little room for additional metainformation on devices. The reliability of the acquired data depends on three parameters: provenance, integrity of stored records, and completeness.

A systematic review of 90 studies shows that data provenance in the IoT is primarily captured on devices or at a level above the network. In 46 of 66 studies, device provenance requires modification. W3C PROV is used primarily on platforms and brokers. Only one study uses LoRaWAN, and none of them model the measurement path from radio reception to platform-side storage.

Integrity mechanisms range from hash chains and authenticated data structures to ledger databases and blockchains. Strong guarantees in the IoT are primarily based on blockchains or device-side cryptography, which is expensive for monitoring platforms. IoT data provenance studies rarely indicate whether the platform operator is trustworthy and rarely test for deletion, reordering, or truncation.

Losses in LoRa and LoRaWAN deployments are reported as aggregate frequency, while completeness is assessed at the dataset level. Existing provenance-based attribution identifies malicious nodes in multi-node networks. No method links missing measurements to the stage of the data processing pipeline where they were lost. The most closely related studies address only one of these aspects. Five requirements have been identified for a LoRaWAN framework with data provenance.

2 DESIGN OF THE PROVENANCE-AWARE IOT FRAMEWORK

2.1 Design principles and architecture

The design of the provenance-aware IoT framework is aimed at supporting the trust in smart city data and also is based on several explicit principles that match realistic deployment conditions. The system is designed for the long-term monitoring of air quality over LoRaWAN networks, in which the limitations of the network are manifold, as discussed in previous chapters: energy consumption, small payload size, unstable communications, etc. Such limitations have a direct impact on the completeness and reliability of the collected data. Having this said, if provenance is treated as an optional add-on, it will most likely come at the price of losing important lineage information precisely when the data are incomplete or also noisy.

Principle 1: Provenance as a first-class feature. The goal of the framework is not to add provenance as an extra function after the development of the system, but to consider it as an integral feature of the IoT system from the beginning. The data go through a full lifecycle, from the raw measurement on the device to its transmission, processing, aggregation as well as publication. As soon as a measurement enters the system, its history (i.e., its provenance) is collected. At each processing stage, direct links are kept between the data, the running processes, also the participants of the system. In relation to existing standards, the design is aligned with the W3C PROV model [55] thanks to which the compatibility with external tools for provenance analysis is also facilitated.

Principle 2: Lightweight edge overhead and progressive enrichment. As we know, IoT sensor nodes operate under tight resource constraints. Under these conditions, devices must minimize computational costs, transmission time, and metadata size in LoRaWAN deployments to preserve battery life and meet duty cycle constraints [42,49], [38]. Accordingly, the framework introduces no device-side provenance overhead and relies on progressive platform-side enrichment. The sensing nodes transmit only the compact measurement payload and the device-status fields required by the monitoring application. Provenance records are generated after uplink reception. The network server and platform enrich the received event with reception timestamps, gateway observations, radio metadata, decoding results, and storage outcomes.

Principle 3: Traceability across the platform-side data lifecycle. The framework preserves links between the received uplink, gateway reception evidence, decoded measurement, validation outcome, and stored database record. Data may be subject to various transformations, such as filtering, resampling, calibration, anomaly rejection, or aggregation in time or space, all of which will create a record in the chain of provenance. Such records identify the transformation step and may include the responsible software component or service, input entities, and also the resulting output entities [55]. The system records validation outcomes and processing status for each decoded measurement. Auditors and analysts will be able to track the entire history of changes and can also check them for compliance with the rules [58].

Principle 4: Modularity and separation of roles. The system is based on a multi-level architecture with clear areas of responsibility. There are four levels: devices, communication gateways, platforms, also applications. Data is transmitted from the bottom up: each level accepts information (including the history of its origin), supplements or verifies it, then passes it on. The layers interact with each other by means of standardized APIs and also with message formats.

Based on these principles, the proposed framework is composed of four main layers: the device layer, communication and gateway layer, platform layer, and application layer. Going into detail, the perception layer includes the sensing nodes which interact with the physical environment and collect air-quality measurements. The collected measurements are transmitted as standard LoRaWAN uplinks without adding provenance information to the end-device payload.

The communication and gateway layer includes the LoRaWAN gateways and network server. As such, this layer is responsible for receiving and routing uplink messages to the application server. Network-level metadata, including reception timestamps, gateway identifiers, RSSI, and SNR, are also provided together with the received uplink data. Within the platform layer, the application server, provenance service, and storage backends are integrated to process and store the received monitoring data. The main work of this layer is composed of three parts, one is the decoding and validation of LoRaWAN uplinks, the second is the capture and storage of provenance records, and the third is the execution of provenance verification and query operations. This way, provenance-aware data storage is established without increasing the communication and computation overhead on the sensing nodes. The application layer provides dashboards and analytical workflows which utilize the provenance-aware Smart City datasets. As a result, the monitoring results can be interpreted together with information about their origin, processing history, and storage operations.

Across these layers, the framework implements the concept of a provenance chain. Each collected measurement is represented as an entity with a unique identifier, and each processing step as an activity that produces a new entity from one or more previous entities. Agents represent devices, gateways, platform services, and operators associated with corresponding activities and entities. This way, the data path will be tracked from receiving a LoRaWAN message to storing the measurements, and then using them in urban applications. Table 2.1 summarizes the main design principles of the proposed framework. Moreover, the system layers mainly affected by each design principle are identified in the table.

Table 2.1 - Mapping of design principles to architectural implications

Design principle	What it enforces in the framework	Primary layer impact
1	2	3
Provenance as a first-class feature	Provenance fields are captured together with measurement data and persist across the pipeline through explicit Entity - Activity - Agent links	All layers

Continuation of Table 2.1

Design principle	What exactly is provided within this system?	Primary layer impact
1	2	3
Minimal edge overhead	Device-side metadata remains compact; enrichment and validation are shifted to gateways and platform services	Device, Communication, Platform
Platform-side traceability and provenance integrity	Each platform-side provenance event creates a record; hash-linked records support integrity verification and audit-oriented traceability from uplink reception to storage.	Platform, Application
Modularity and separation of roles	Layer-specific responsibilities and stable interfaces reduce provenance loss when components change	Communication, Platform

As shown in Table 2.1 framework will contain main design principles and answers to the question “What exactly is provided within this system?”.

Figure 2.1 illustrates where the provenance information is created, enriched, and stored across the layered Smart City pipeline.

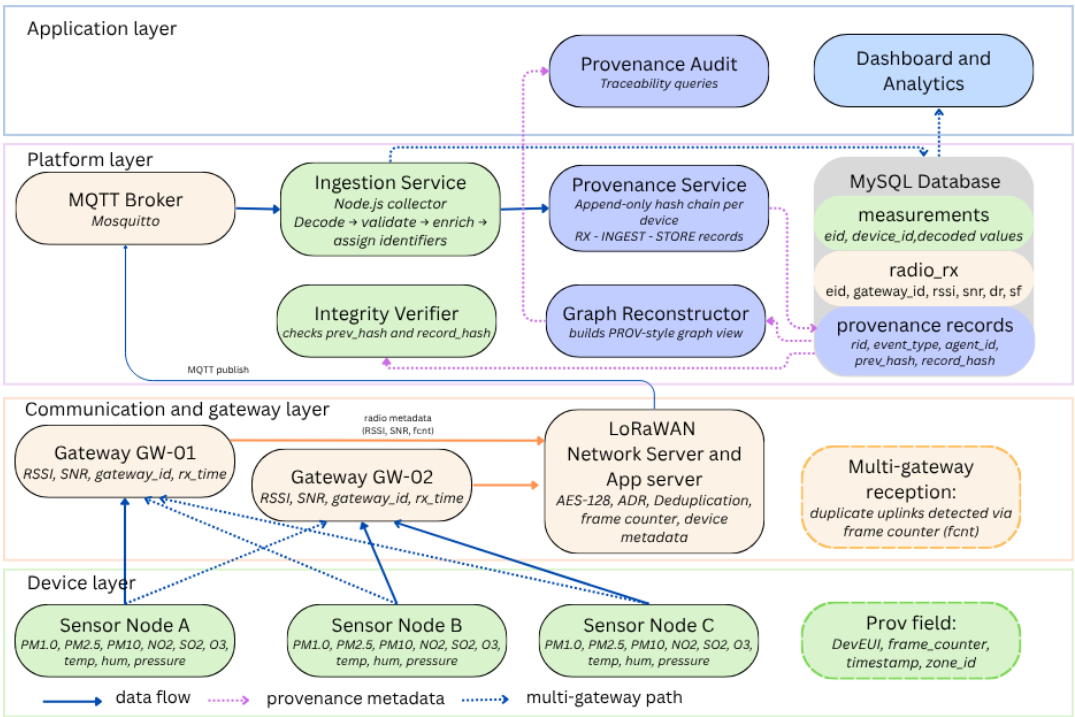


Figure 2.1 - High-level layered architecture of the provenance-aware LoRaWAN IoT framework with provenance capture, integrity verification, and graph reconstruction

The proposed framework follows a layered structure composed of the device (perception) layer, communication and gateway layer, platform layer, and application layer. The device layer produces monitoring measurements and provides the initial

observation attributes, with device identifiers, payload values, and device-side timestamps if available. The communication and gateway layer forwards the LoRaWAN traffic and enriches the data flow with network-level metadata, such as gateway identifiers, reception timestamps, RSSI, SNR, and frame counter information.

Within the platform layer, provenance-aware storage, provenance services, integrity verification, and provenance query functions are implemented. The application layer provides dashboards, audit views, and analytical workflows which consume the provenance-aware Smart City data. As a result, Figure 2.1 presents the conceptual architectural design of the proposed framework. The next section formalizes these architectural elements by defining the main system components, data entities, provenance records, and integrity relationships utilized in the proposed framework.

Figure 2.2 illustrates the logical provenance flow and storage interactions within the proposed framework. Unlike the layered architectural view presented in Figure 2.1, the logical model focuses on the generation of provenance records, enrichment of metadata, storage interactions, and lineage relationships between measurements, communication events, and provenance records.

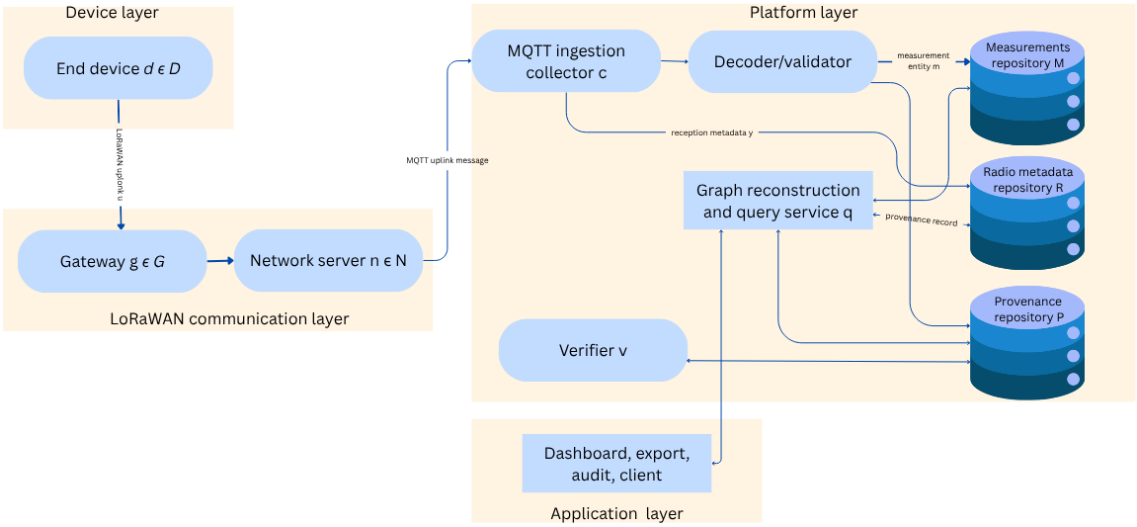


Figure 2.2 - Logical provenance flow and storage interactions within the proposed provenance-aware LoRaWAN framework

The proposed framework separates the measurement storage, radio metadata storage, and provenance storage. It helps to preserve traceability across the different stages of the monitoring pipeline. The sensor observations generated by the end devices are transmitted through the LoRaWAN communication infrastructure. After that, they are further processed by the MQTT ingestion collector and decoder. During operation, reception metadata, verification results, and the history of its origin are added to each measurement.

At the platform level, this service collects a graph of relationships and allows us to track the provenance of data, even if they are stored in different databases. In parallel, the verifier subsystem performs the integrity validation of provenance

records and append-only provenance chains. The separation of storage and verification functions improves the modularity of the framework. Each function can therefore be managed and updated independently.

The logical model connects measurements, communication events, as well as the history of data provenance. These links are needed to verify the integrity and build a dependency graph. They are used to build a formal model, which we will discuss in detail in the following sections.

2.2 Integration with the LoRaWAN communication stack

Figure 2.3 shows the standard LoRaWAN communication architecture, which includes endpoints, gateways, a Network server, a connection server (Join Server for the OTAA procedure), and an Application Server. This architecture serves as the communication foundation for the proposed system. The framework being developed is integrated into LoRaWAN without changing the protocol, message format, or cryptographic mechanisms. The LoRaWAN specification defines MAC-level logic, device classes, frame structure, MIC mechanisms, and replay protection based on a frame counter [49]. The provenance data is collected externally, on the platform side, and they use the information already available in the standard LoRaWAN deployment.

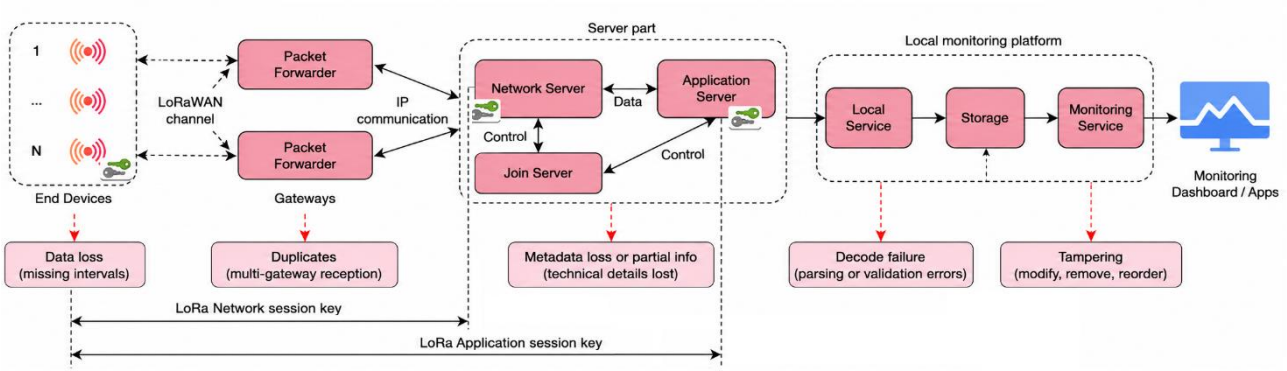


Figure 2.3 - LoRaWAN communication architecture

Figure 2.3 shows the standard LoRaWAN architecture and the main risks related to provenance during the transmission, processing, also storage of data. Lorawan's history does not retain a full history of data processing after it has been received, but basic communication security is provided.

The processing of data after the ingestion also plays a significant role in the quality of the stored measurements. Along the way to the database, the data may be affected by transmission losses, duplicate receptions, loss of metadata, processing errors or tampering, and LoRaWAN does not keep any record of such steps. Despite this, the history of the data can be recovered without changes in the end devices or in the gateways. The proposed framework captures provenance information, stores hash-linked records and verifies their integrity, so each measurement can be traced along its whole lifecycle.

The integration strategy is aimed at smart city sensing with constrained resources and follows a non-invasive principle. The end devices remain lightweight and transmit compact sensor payloads according to the configured sampling policy. In terms of our proposal, this translates as no provenance content being embedded into the LoRaWAN application payload. This solution maintains payload capacity at various data rates without requiring additional time to track data origin. As a result, the collection of provenance data is mainly moved to the server side, where computing power, storage resources as well as access control mechanisms are sufficient.

The gateways remain unchanged, and act as standard forwarders. The gateways receive the uplink frames and relay them to the network server through the usual backhaul, so the framework does not require them to store provenance data or to execute any provenance logic. The network server performs the standard LoRaWAN functions, such as the handling of sessions, the tracking of frame counters and the integrity checks at protocol level. The important point for the integration is that the network server provides an application interface, which exposes both the decoded payload and the radio metadata of each uplink. Based on such information, the platform is able to build provenance records for traceability, integrity checks and also reliability analysis.

In the proposed framework, the platform subscribes to the LoRaWAN uplink events through an integration interface based on MQTT. The network server publishes uplink event messages that include the device identifier and a timestamp, together with the reception context, like the gateway identifier, the RSSI, the SNR and as well as the data rate parameters. When available, the event stream also provides references to the message counter and the outcome of the validation regarding duplicates or counter consistency. Such fields are treated as metadata relevant for provenance, since they affect the interpretation of data quality and provide evidence about the communication conditions during a whole year of monitoring.

Network-server event log. Network-server event log. The framework requires the network server to retain an event log E_N that records, for every frame it accepts, the device identifier, the frame counter and the reception time at the network server. The log must be written by the network server itself (e.g., through one of its built-in database integrations [150]), and not through the same MQTT subscription used by the provenance collector. Otherwise, an outage of the collector would remove both the provenance records and the evidence of their absence. The log is not part of the provenance chain, and is not used to verify individual records. Instead, it serves two purposes: it allows distinguishing the losses at platform intake (class P0) from the upstream losses (class U1) in the attribution model (Section 2.4.5), and it provides an independent reference to check the set of received uplinks.

A dedicated platform component, the provenance collector, consumes uplink event messages from MQTT and converts them into platform-level entities as well as provenance records. After receiving an uplink event, the collector decodes the payload and creates a measurement entity identifier for the decoded observation. It then appends the provenance records for the reception and ingestion stages. The

ingestion record links the logical uplink entity to the stored measurement entity and binds the main metadata fields, such as device identity, timestamps, gateway information and also link indicators. Traceability is therefore established from the first platform-managed representation of the data, and the evidence needed for later auditing is preserved. In the implemented prototype, the logical uplink entity contains stable identifiers and the recorded reception context, rather than a stored copy of the complete MQTT message.

The collector processes each uplink as a single atomic database transaction, in the sense of the ACID properties [151]. If decoding and verification succeed, the RX record, the INGEST record with status OK, the measurement record as well as the STORE record are written together. If decoding or verification fails, an RX record and an INGEST record with status FAIL are written, so the platform keeps the evidence of the failure (class P1). If the store operation fails, the transaction is rolled back and no record of the uplink remains on the platform. In that case, the uplink can only be detected through the network server event log (class P0). As a result, the provenance repository never contains STORE records without the corresponding RX and INGEST records, nor partially processed uplinks. The implementation is given in Algorithm 3.1.

Further processing within the platform follows the same lineage rules (see Section 2.4). In the conceptual model, each step that changes the representation or the value of the data (e.g., filtering, calibration, aggregation or resampling) is registered as an explicit TRANSFORM action with a new entity identifier, while publication to the dashboard is registered as a PUBLISH event. In this way, raw measurements are never overwritten by derived values, and lineage can extend from the final visualizations back to the upstream context. The implemented prototype registers RX, INGEST and STORE events, while TRANSFORM and PUBLISH remain extension points (Section 2.5).

Provenance chains are maintained per device, in order to preserve the record order and to allow an independent verification of each sensing stream. Such design matches the behavior of LoRaWAN monitoring, since each device produces a sequence of uplinks over time. At the platform layer, provenance records are stored in an append-only repository and linked by cryptographic references. At the same time, the platform supports cross-device queries through entity identifiers and indexed metadata. Smart city monitoring requires this capability, because analysts may need to compare patterns across zones, devices and also time periods.

Although the platform stores provenance information as a chain of records for independent verification, its underlying provenance structure corresponds to a graph structure based on the PROV model. Using references to the input and output entities of each provenance event, the platform reconstructs a graph representation from the stored records. The reconstructed graph supports both backward tracing, from any output entity to its sources, as well as a forward tracing, from any measurement entity to all derived products. The graph representation is therefore used for provenance queries and visual explanation.

The proposed structure adds provenance to the LoRaWAN network without modifying it. Devices send the same compact payloads, and gateways forward frames

as usual. All provenance information is taken from what the network server already publishes via MQTT, by including device identifiers, frame counters, reception metadata as well as platform processing results. Such structure is suitable for long-term air quality monitoring, since radio conditions change over time, and the transmitted data can only be trusted if their origin, processing and also storage can be tracked and verified.

2.3 Threat model and trust assumptions

The present section analyzes the goals and capabilities of potential attackers in monitoring systems based on LoRaWAN. The trusted components and the trust boundaries of the proposed framework are defined, and the attack scenarios that target both measurement data and provenance records are described. For each threat, the detection and protection mechanisms are considered, based on standard LoRaWAN security, keyed hash chains (HMAC-SHA-256), consistency checks of the data history and integrity verification procedures. The associated residual risks are also assessed.

In this dissertation, the platform is described as secure only in the narrow sense of detecting unauthorized changes (tamper evidence). According to the threat model described below, if an attacker modifies, deletes, inserts, rearranges, or also truncates stored data provenance records after they have been received, this change is detected upon the next check. The framework does not prevent such changes and does not add confidentiality or availability beyond what LoRaWAN already provides. Furthermore, the framework cannot verify the physical correctness of measured values, since a correct record does not always mean a correct measurement.

A0 – network adversary, who can eavesdrop, jam, drop, delay, or replay frames on the radio channel, at gateways, or on the backhaul, corresponding to the network adversary of [152]. Protection is provided by LoRaWAN (MIC, frame counters, session keys). The framework does not prevent the resulting losses, but it attributes them to the upstream stage (class U1).

A1 – database-level writer without access to the verifier, the checkpoints, or the chain key, for example a compromised database account or an erroneous maintenance script. This corresponds to the untrusted-storage adversary considered in secure audit logging [153], [115]. With the keyed chain (Section 2.4.4), modification, deletion, insertion, and reordering of records are detected anywhere in the chain, and tail truncation is detected relative to the latest checkpoint. With an unkeyed SHA-256 chain, as used in the original deployment, a consistent rewrite of the records appended after the latest checkpoint would not be detected.

A2 – trusted operator, who is assumed to be honest. The verifier detects accidental changes by the operator in the same way as changes by A1.

A3 – fully malicious administrator, who controls the repository, the verifier, the chain key, the checkpoints, and the network-server event log. This adversary can construct a false but internally consistent history and is outside the scope of the framework.

Attackers may attempt to replace or falsify measurements, or hide incidents, insert fake data, compromise cryptographic keys or platform credentials, or disable the platform through denial-of-service (DoS) attacks. The considered attack scenarios are:

- On the air: listening, replay attacks, packet injections, frequency jamming.
- At the hardware level: physical hacking of sensors, modification of their firmware and key extraction.
- On the network path: interception of data on gateways and servers in order to modify or delete messages.
- Inside the platform: insider access to the database and attempts to rewrite the provenance logs.
- Related risks: MitM (man-in-the-middle) attacks through DevEUI spoofing, as well as device compromise at the delivery stage.

In the proposed architecture, the local monitoring platform, the data history storage subsystem, database and integrity verification services, as well as the internal time source are considered trusted components. It is assumed that they are deployed in a secure environment with administrative access control. Device initialization and configuration processes also rely on trusted procedures such as secure personalization or the standard OTAA activation procedure. The LoRaWAN root keys (AppKey and NwkKey) are managed within standard network security practices and also are outside the direct control of the provenance tracking system [154], [155].

Regarding the trust model, the LoRaWAN radio channel, the gateways and also the external network infrastructure belong to an untrusted zone. Packets can be intercepted, dropped, delayed, lost or also replayed, and the gateways themselves can be compromised. LoRaWAN provides protection at protocol level for the uplink communication, by means of device authentication, MIC validation, frame counters and the management of session keys. However, such mechanisms do not record the history of the uplink after its reception on the platform side (i.e., decoding, validation, transformation and storage in the database), and do not provide any tamper evidence for the records modified after the ingestion. In order to extend the integrity monitoring beyond the LoRaWAN reception, the framework adds mechanisms for the capture and verification of provenance on the platform side. More specifically, the framework uses separate hash-linked provenance chains per device, together with the corresponding procedures for integrity verification. The provenance layer is designed to detect the specified modifications after the ingestion that produce inconsistent record hashes or chain links. Note that the provenance layer does not verify the physical correctness of the sensor readings, and does not provide any protection when the trusted platform and its verification state are fully compromised.

Regarding the security model, we assume that an attacker does not obtain administrative control over the provenance repository, does not know the chain key, and is not able to replace the trusted checkpoints of the chain heads, to compromise the verification service or to manipulate the trusted time source of the platform. Within such trust boundaries, a modification of the hashed content of a record is detected when the verification finds a mismatch between the stored and the recomputed hash of the record, or an inconsistency in the link to its predecessor.

More specifically, a modification of the hashed content causes the recomputed SHA-256 value to differ from the stored record_hash, unless the affected verification state is also replaced in a consistent way. Similarly, the deletion, insertion or reordering of records, without a consistent recomputation of the affected part of the chain downstream, produces a mismatch between prev_hash and the record_hash of the expected predecessor. However, the truncation of the chain tail does not necessarily break the remaining internal hash links. For this reason, tail truncation is detected by means of the comparison of the current chain head and the number of records with a previously stored trusted checkpoint.

The integrity guarantees of the framework are defined under a trusted-platform assumption. Under this assumption, the verification mechanism is designed to detect the following specified post-ingestion changes to stored provenance records:

- direct modification of hashed record content;
- intermediate-record deletion, insertion, or reordering that disrupts the expected chain linkage;
- tail truncation relative to a previously stored trusted chain-head checkpoint.

The threat model does not cover a fully malicious administrator of the platform, the compromise of the verification service or of the LoRaWAN root keys, nor the physical compromise of the sensing nodes. The proposed mechanism focuses on the integrity and traceability of the history of data on the platform side, once the uplink has been received.

The main integrity mechanism on the platform side is a set of provenance chains per device, linked by means of keyed SHA-256 hashes (HMAC-SHA-256) [156]. The representation of provenance follows the Entity-Activity-Agent concepts and some selected relations of the W3C PROV model [55]. The model uses the PROV relations *wasGeneratedBy, used, wasAssociatedWith, and wasDerivedFrom* to reconstruct dependency relationships between the stored entities, activities, as well as agents [66].

The main assets of the proposed architecture include the measurement *payload (d)* and associated *metadata (m)*, *provenance records* (r_i), links to previous *prev_hash* hashes, and hashes of the *record_hash* records themselves. Moreover, the assets include the provenance chains of each device (C_d), the reconstructed graph of provenance (G_p), the results of checks and saved measurement logs. The LoRaWAN cryptographic keys are considered as security tools at network level, managed within the standard LoRaWAN security model. In contrast, the proposed framework only focuses on the integrity of the history of data, once it is received by the local monitoring platform.

The threat scenarios described below show how the capabilities of the intruder and the attacked assets are related to the protection and detection tools of the framework. The residual risk is assessed in a qualitative way, within the stated threat model. A low residual risk means that the threat under consideration is mostly addressed by the implemented mechanisms, under the assumption of a trusted platform. A medium residual risk means that the mitigation or detection of the threat also depends on the controls at communication layer, on the timing of the verification or on operational safeguards. Finally, a high residual risk corresponds to a threat that

can bypass the provenance mechanism, or that lies outside its scope of protection. Such classification is used for the architectural analysis only, and does not represent a quantitative estimation of probability.

- *Data-content modification.* An attacker may attempt to distort the payload or the metadata while they are transmitted over the air, forwarded through a gateway or stored on a server. During the wireless transmission, the MIC validation and the frame counters of LoRaWAN support the verification of message integrity and the protection against replay at network level. Once the provenance records have been stored, the modifications that make their hashed content or the links to their predecessors inconsistent can be detected by means of the recomputation of *record_hash* and the check of the relation between each *prev_hash* and the *record_hash* of the expected predecessor. Such mechanism addresses the integrity of the recorded history after the ingestion only. The physical compromise of a sensing node and the falsification of measurements before the ingestion remain outside the scope of protection of the framework.
- *Data injection and spoofing*. An attacker may try to inject fabricated packets, or to pretend to be a legitimate device. The main protection against unauthorized LoRaWAN messages is provided by the protocol itself, by means of the device and session authentication, the MIC validation, the frame counter checks and the security mechanisms of the network server. At platform level, stable identifiers like *device_id* and *uplink_key* are combined with the timestamps and the frame counter information, in order to support the detection of duplicates and the analysis of the order of events. However, if a legitimate node is physically compromised, it can still generate readings that are false but cryptographically valid. Such scenario is outside the stated threat model of the proposed provenance mechanism.
- *Deletion or omission of provenance records*. The deletion of an intermediate provenance record, without the recomputation of the affected records downstream, breaks the expected continuity of the hash-linked chain. The *prev_hash* value of the next available record does not match the *record_hash* of its current predecessor. As a result, the missing or inconsistent links between entities may also become visible during the reconstruction of the lineage graph and also the execution of audit queries. The change is detected once the affected segment of the chain is inspected, so the time needed for its detection depends on the audit schedule.
- *Reordering or insertion of records*. The reordering of existing records, or the insertion of an orphan record without a consistent update of the affected hash links, produces inconsistencies in the expected relations between each record and its predecessor. As a result, the recalculated hashes and also the links to the previous records no longer match. The sequential verification of the chain reveals such interference in the structure of the logs or in the chronology of events, as long as the platform is trusted and also the stored verification state remains intact.

- *Replay attacks*. For replayed LoRaWAN uplinks, the main protection at protocol level is the validation of frame counters at the network server. On the platform side, uplink_key, the timestamps and the frame counter information are used to identify repeated events and to support consistency checks. In this sense, the protection against replay is mainly a function of the LoRaWAN communication layer, rather than a property of the provenance hash chain.
- *Gateway or communication-channel compromise*. A compromised gateway may delay, reset, duplicate, or incorrectly forward packets. A gateway that does not possess the required session keys cannot modify an uplink payload and produce a corresponding valid MIC without such modification being rejected during network-server integrity validation. Packets dropped before reaching the network server appear at the platform as missing expected uplinks and reduce measurement completeness; no platform-side provenance record can be created for an uplink that was never received. At the same time, DoS attacks, jamming of the radio signal and deliberate blocking of packets are among the threats to accessibility and provenance control mechanisms are not eliminated. In this regard, the residual risk to data integrity is assessed as low or moderate, while the risk to system availability remains high.
- *Platform or database compromise*. If a platform or database is compromised, an attacker may try to change saved measurements, logs of provenance, or hashes. Isolated modifications that are not followed by consistent recomputation of the affected chain state are detected as hash mismatches or broken predecessor links during verification. However, an attacker with administrative control over both the provenance repository and the verification state may construct a false but internally consistent chain. Such a compromise violates the trusted-platform assumption and is outside the protection scope of the framework.
- *Insider threats*. Insider threats develop in a similar way, when privileged users intentionally delete or edit their data history. The system recognizes interference only if the insider does not have the resources to completely and consistently replace all states of the chain. If he controls the storage, the verification logic, and the audit results, he will be able to bypass the security barriers. Therefore, the framework only partially reduces the risks of insider attacks, provided that rights are strictly delimited, requiring the mandatory implementation of external organizational control.

The proposed model does not protect against DoS attacks, radio interference, physical hacking of the sensors or leakage of confidential data. The complete compromise of the trusted platform, as well as the compromise of the cryptographic assumptions on which SHA-256 integrity verification depends, are outside the evaluated threat model. Rather than being a complete security solution, the framework has a defined scope: it provides provenance traceability and tamper evidence for the processing history on the platform side, after the uplink is received. In this context, the system is not responsible for network availability, channel encryption or the physical validity of sensor measurements.

To reduce the operational risks, the monitoring system needs a secure registration of the LoRaWAN devices (preferably via OTAA) [49,51] and a strict separation of access rights to the server platform, the database, the log storage and the verification module. During normal operation, provenance records are kept with append-only semantics, while regular backups act as an additional recovery mechanism. The audit subsystem records the anomalies found during verification and validation, such as record_hash mismatches, broken prev_hash links, missing expected records, duplicate events and also discontinuities in timestamps or frame counters. Such anomalies require further audit, since they may result from data loss, unauthorized modification, communication failure or equipment malfunction.

When an integrity incident is found, the operator may apply the following recovery procedure: isolate the problematic nodes, block or reissue the keys of compromised devices, retrieve clean data from trusted backups and run again the verification of the whole hash chain. Table 2.2 separates the threat scenarios addressed by the proposed mechanisms from those that remain outside the protection scope of the framework.

Table 2.2 - Threat scenarios with affected assets and mitigation mechanisms of the proposed framework

Threat scenario	Affected asset	Protection and detection mechanism	Main layer	Residual risk
1	2	3	4	5
Eavesdropping on wireless traffic	Payload and metadata in transit	The standard LoRaWAN encryption protects the confidentiality of the payload during the wireless communication. Once the data leave the LoRaWAN security boundary, their confidentiality is outside the scope of the provenance mechanism.	Communication	Medium
Payload or metadata modification	Payload, identifiers, timestamps, radio metadata	The network server validates the LoRaWAN MIC of each received uplink, and rejects the frames that fail such check. After the ingestion, the inconsistent modifications	Communication, platform	Low to medium

Continuation of Table 2.2

Threat scenario	Affected asset	Protection and detection mechanism	Main layer	Residual risk
1	2	3	4	5
		of provenance records are detected by recomputing record_hash and checking the prev_hash links.		
Injection of false messages	Payload, metadata, device identity	LoRaWAN authentication, MIC validation, and frame counters protect against unauthorised frames. Platform-side identifiers support duplicate and consistency checks, but a compromised legitimate node may still generate false measurements.	Communication, platform	Medium
Replay attack	Payload and provenance records	LoRaWAN frame counters and platform-side duplicate detection using uplink_key, frame counters, timestamps, and entity identifiers.	Communication, platform	Medium
Deletion or omission of provenance records	Provenance records and chain links	Intermediate-record deletion without consistent downstream recomputation produces broken prev_hash linkage and is detected during chain verification. Tail truncation is treated separately and is detected by comparing the current chain state with a previously stored trusted chain-head checkpoint.	Platform	Medium
Reordering or insertion of records	Provenance records and chain order	Sequential verification of each device chain detects reordering or insertion when hashes or predecessor links become inconsistent.	Platform	Low

Continuation of Table 2.2

Threat scenario	Affected asset	Protection and detection mechanism	Main layer	Residual risk
1	2	3	4	5
Unauthorized access or insider modification	Provenance records, hashes, timestamps	Role-based access control and restricted interfaces are required to limit unauthorised platform access. Hash-chain verification provides evidence of record modification when the attacker cannot consistently replace both the affected provenance records and the trusted verification state.	Platform	Medium to high
LoRaWAN key or platform credential compromise	LoRaWAN keys, platform credentials, verification service access	Secure credential management and restricted platform access are required. The integrity properties of the framework no longer apply in full if trusted credentials, the provenance repository, or verification components are compromised.	Device, platform	High
Fully malicious platform operator	Provenance storage, verifier, audit outputs	Not addressed by the proposed framework; outside the trusted-platform assumption.	Platform	High

As shown in Table 2.2, the framework mainly covers the integrity and traceability of provenance once the data reach the trusted local platform. Record modification, deletion, insertion, or reordering that produces inconsistent hashes or chain links is detected by recalculating the keyed record hashes and checking the *prev_hash/record_hash* relationships. Tail truncation is detected separately by comparison with the stored chain-head checkpoint. Replay protection and communication confidentiality are primarily provided by LoRaWAN, while DoS conditions and complete compromise of the local platform remain outside the provenance mechanism. The reported integrity properties therefore apply under the stated trusted-platform assumption.

The main residual risks are related to the compromise of system components, database accounts or verification services. An attacker with full control over the provenance repository and the verification state may build a false, but internally consistent, hash chain. Under the assumption of a trusted platform, however, the unauthorized changes that leave the stored record hashes or chain links inconsistent are detected during the integrity verification.

In this sense, the threat model is aligned with the protection and detection mechanisms implemented by the framework. LoRaWAN provides authentication and message integrity at protocol level, while the proposed framework records and verifies the history of processing on the platform side. Such combination supports the traceability from the received uplinks to the stored measurement records.

The confidentiality of the transmitted payloads is provided by LoRaWAN, and is not extended by the framework. Availability is not protected either. The framework does not prevent the loss of data, but it makes the losses visible and, when the recorded evidence permits it, attributes them to a stage of the pipeline by means of the attribution model. In summary, within the stated threat model, the framework provides tamper evidence and traceability for the history of processing on the platform side, together with the attribution of missing data at the level of each stage, without claiming a complete end-to-end security.

2.4 Formal model of the framework

The functions of tracking, recovery of the data history and integrity verification are based on a formal model, which combines a structured description of the provenance of data with a lightweight cryptographic protection. The model does not change the standard LoRaWAN protocol, since the data history is collected and processed on the platform side, right after the reception and decoding of the uplink messages.

2.4.1 System components and identifiers

The main architectural components of the monitoring and also provenance framework are represented by the following tuple:

$$S = (D, G, N, P, V, Q) \tag{2.1}$$

where

$D = \{d_1, d_2, \dots d_n\}$ denotes the set of sensing devices;

$G = \{g_1, g_2, \dots g_m\}$ denotes the set of LoRaWAN gateways;

N denotes the LoRaWAN network and messaging infrastructure;

P denotes the provenance storage and management subsystem;

V denotes the integrity verification subsystem;

Q denotes the provenance query and graph reconstruction subsystem.

Each sensing device transmits its measurement results periodically through the LoRaWAN network. The platform receives and processes such data, and records their provenance on the platform side once the uplink has been received.

Each logical uplink is identified by a key that remains stable across the repeated deliveries of the same event from the network server, and that is unique across the re-activations of the device:

$$uplink_key_i = \begin{cases} dedup_id_i, \\ H\big(DevEUI_d \parallel epoch_{d,i} \parallel fcnt_i \parallel payload_hash_i\big), \end{cases} \quad (2.2)$$

where $dedup_id_i$ is the identifier that the network server assigns to the deduplicated uplink event (the deduplicationId in ChirpStack v4), $fcnt_i$ is the uplink frame counter, which the device increments for every new uplink, restarts after a join [49], $payload_hash_i$ is the hash of the payload, ‖ denotes concatenation, and $H(\cdot)$ is SHA-256. The session epoch $epoch_{d,i}$ identifies the activation session of device d: it is incremented when the platform processes a successful join event of the device, or when it observes a decrease of the frame counter, it indicates a re-join whose join event was not received. Without the epoch, two uplinks from different OTAA sessions with equal frame counters and identical payloads would obtain the same key after the frame counter is reset. The reception timestamp is stored as metadata but is not included in the key, because gateway and platform clocks may not be synchronised and repeated deliveries of the same event must map to the same key.

A measurement entity is formally represented as:

$$m_j = (eid_j, device_id_j, t_j, val_j, meta_j) \quad (2.3)$$

where, the eid_j is the unique entity identifier, $device_{id\,j}$ is the sensing device identifier, t_j is the measurement timestamp, val_j is the measured sensor value set, $meta_j$ contains the associated metadata.

The framework supports environmental variables like PM_1, $PM_{2.5}$, PM_{10}, gaseous pollutants, temperature, as well as humidity. The representation of provenance is aligned with the Entity-Activity-Agent concepts of the W3C PROV model [55], with its OWL encoding PROV-O [66].

2.4.2 Provenance records and graph

Entities are the physical or digital elements of the system, like sensor readings, MQTT uplink messages, decoded data, database entries and the results of integrity checks. *Activities* are the operations performed on them, like decoding, the addition of metadata, validation, storage and integrity verification. *Agents* are the components that perform such operations, namely the sensors themselves, the gateways, the data reception services, the databases and the verification modules. All together, entities, activities and agents form the conceptual model on which the whole history of the origin of data in the architecture is based.

In order to describe the history of data, the system uses a minimal set of relations from the W3C PROV standard [55]. The *wasGeneratedBy* relation shows a which process created an object and also used shows which data were used by such process. The process to a specific component assigns *wasAssociatedWith* relation (i.e., an agent), and the *wasDerivedFrom* parameter specifies the source data from which a new object was derived. Thus, the relationships form a continuous chain, allowing the history of any published value to be traced back to its original source.

The provenance model represents the processing of each LoRaWAN uplink as a sequence of entities, activities and agents, linked by relations in the PROV style. More specifically, the model captures the transformation of the logical uplink entity, from its reception in the network to its final representation in the database, and preserves the intermediate evidence of reception and measurement. Figure 2.4 shows the resulting provenance structure and the relations established between the main components of the lifecycle.

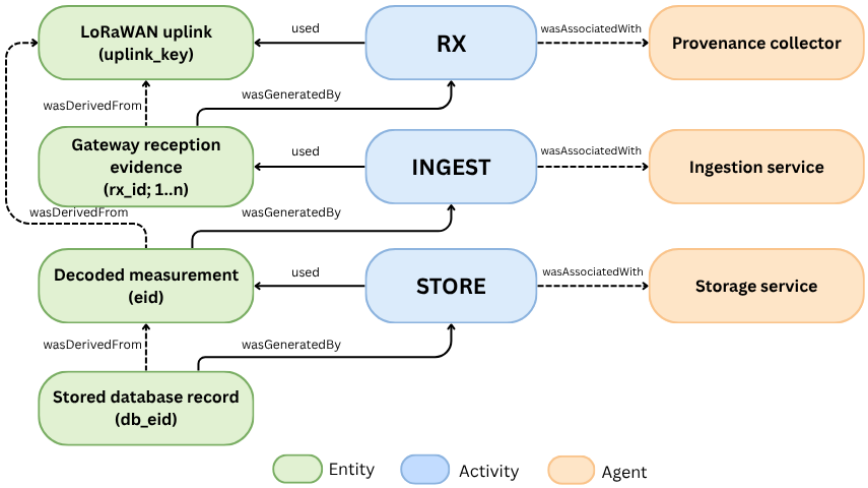


Figure 2.4 – PROV-style provenance model of the LoRaWAN uplink processing lifecycle

As shown in Figure 2.4, the lifecycle is represented by three main activities, namely RX, INGEST and STORE. The RX activity processes the received LoRaWAN uplink and produces the evidence of gateway reception. For the same logical uplink, such evidence may contain several reception records. The INGEST activity uses the reception evidence to derive the decoded measurement entity, whereas the STORE activity persists the decoded measurement as a database record. The corresponding used, wasGeneratedBy and wasDerivedFrom relations preserve the dependency between consecutive entities of the lifecycle.

Each activity is also associated with the platform component in charge of its execution. In particular, the provenance collector is associated with RX, the ingestion service with INGEST and the storage service with STORE. As a result, the representation supports backward tracing, from a stored database record to the identifier of the logical LoRaWAN uplink and its reception evidence, as well as forward tracing, from an uplink to the resulting stored measurement.

Let $C_d = (r_{d,1}, r_{d,2}, \dots r_{d,n})$ denote the ordered provenance chain of device d, where n_d is the number of provenance records in the chain. The k -th provenance record of device d combines the PROV notions of entity, activity, also agent with the predecessor-hash fields used in hash-chained audit logs [153] is represented as

$$r_{d,k} = \left\langle \begin{matrix} rid_{d,k}, device_id_d, event_type_{d,k}, timestamp_{d,k}, \\ input_eids_{d,k}, output_eid_{d,k}, agent_id_{d,k}, \\ status_{d,k}, meta_{d,k}, prev_hash_{d,k}, record_hash_{d,k} \end{matrix} \right\rangle \quad (2.4)$$

where $rid_{d,k}$ is the unique provenance-record identifier; $device_id_d$ identifies the sensing device; $event_type_{d,k}$ denotes the provenance-event type; $timestamp_{d,k}$ records the time of the event; $input_eids_{d,k}$ and $output_eid_{d,k}$ define entity-level dependencies; $agent_id_{d,k}$ identifies the responsible platform component; $status_{d,k}$ records the processing outcome; $meta_{d,k}$ contains additional event metadata; and $prev_hash_{d,k}$ and $record_hash_{d,k}$ provide the cryptographic linkage between consecutive records.

In the implemented framework, the event type is defined as:

$$event_type_{d,k} \in \{RX, INGEST, STORE\} \quad (2.5)$$

The provenance record structure presented in Table 2.3 provides the operational basis for append-only provenance storage and graph-view reconstruction in the proposed framework.

Table 2.3 - Provenance record fields in the proposed provenance chain model

Field	Description	Layer and source of creation	Status
1	2	3	4
rid	Unique provenance record identifier	Platform provenance service	Mandatory
device_id	Identifier of the sensing node	Device identifier captured by platform	Mandatory
event_type	Provenance event type: RX, INGEST, STORE	Platform provenance service	Mandatory
timestamp	Time associated with the recorded event	Network server or platform service	Mandatory
agent_id	Identifier of the responsible agent or service	Network server, provenance collector, storage service	Mandatory
input_eids	Identifier(s) of the input entity used by the event	Platform provenance service	Mandatory
output_eid	Identifier of the output entity produced by the event	Platform provenance service	Mandatory
zone	Coarse-grained deployment zone	Platform metadata enrichment	Conditional

Continuation of Table 2.3

Field	Description	Layer and source of creation	Status
1	2	3	4
status	Processing outcome of the event (OK or FAIL); a failed INGEST identifies a decoding or validation failure (class P1)	Platform provenance service	Mandatory
fcnt	LoRaWAN frame counter associated with the received uplink	Network server metadata	Conditional; recorded for RX events
prev_hash	Hash of the previous provenance record in the device chain	Platform provenance service	Mandatory
record_hash	Hash of the current record content linked to prev_hash	Platform provenance service	Mandatory
reason	Reason or diagnostic note for exceptional processing outcomes	Platform provenance service	Optional; nullable

At the storage level, provenance records are maintained as per-device hash-linked chains. For queries and visualization, the platform reconstructs a directed provenance graph from the stored record semantics.

Let $\mathcal{R} = \bigcup_{d \epsilon D} C_d$ denote the set of stored provenance records. The reconstructed provenance graph is defined as

$$\mathcal{g}_p = (V_p, E_p) = \Phi(R) \tag{2.6}$$

where $\Phi(\cdot)$ is a mapping function that converts the semantics of records into nodes and edges of the graph in accordance with the PROV standard [55]. The dependency edges are oriented from source entities and activities towards the entities derived from them.

For any object identifier $e \epsilon V_p$, backward and forward traceability are defined through directed reachability in the reconstructed graph, following the graph-based definition of lineage in provenance research [59]:

$$L^-(e) = \left\{ e' \epsilon V_p \mid e' \rightarrow^+_{\mathcal{g}_p} e \right\}$$
$$L^+(e) = \{ e' \epsilon V_p \mid e \rightarrow^+_{\mathcal{g}_p} e' \}, \tag{2.7}$$

Here, $\rightarrow^+ \mathcal{g}_p$ denotes reachability through one or more directed dependency edges in the reconstructed provenance graph $\mathcal{g}_p$. Consequently, $L^-(e)$ contains all

upstream nodes from which e is directly or indirectly derived, whereas $L^+(e)$ contains all downstream nodes that are directly or indirectly derived from e.

2.4.3 Provenance generation model

Provenance generation is modelled as a discrete-event process. Let $I = \{1,2,\dots,M\}$ denote the set of expected uplink opportunities during the observation period.

Completeness is assessed, as is common in data-quality research, as the ratio of available to expected values [157], [158], [56]; the expected values are given by the transmission schedule.

For deployments in which device d operates for duration T_d with a configured transmission interval Δt_d, the total number of expected uplink opportunities is

$$M = \sum_{d \in D} \left\lfloor \frac{T_d}{\Delta t_d} \right\rfloor \tag{2.8}$$

When all n devices use the same monitoring duration T and transmission interval Δt, this expression reduces to

$$M = n \frac{T}{\Delta t}$$

For each expected uplink opportunity i, let $\gamma_i \in \mathbb{N}_0$ denote the number of gateway reception records associated with the transmission. The indicator of a logical uplink accepted by the network server after multi-gateway deduplication is defined as

$$x_i = \begin{cases} 1, & \gamma_i \geq 1, \\ 0, & \gamma_i = 0. \end{cases} \tag{2.9}$$

Two further indicators describe the platform side. Let $z_i \in \{0,1\}$ indicate that the platform committed reception evidence for the uplink, i.e. its RX and INGEST provenance records, and let $y_i \in \{0,1\}$ indicate that the uplink was successfully decoded, validated, and stored as a measurement record. A measurement cannot be stored without reception evidence, and reception evidence cannot exist for an uplink that the network server did not accept; therefore

$$y_i \leq z_i \leq x_i. \tag{2.10}$$

Under the transaction rule of Section 2.2, a decoding or validation failure yields $z_i = 1$, $y_i = 0$, whereas a storage failure rolls back the whole transaction and yields $z_i = 0$. The total numbers of gateway-level reception records, accepted logical uplinks, platform-received uplinks, and stored measurements are

$$\begin{aligned} N_{radio} = \sum_{i=1}^{M} \gamma_i\,, \quad N_{uplink} = \sum_{i=1}^{M} x_i, \\ N_{rx} = \sum_{i=1}^{M} z_i\,, \quad N_{store} = \sum_{i=1}^{M} y_i. \end{aligned} \tag{2.11}$$

Individual gateway observations are retained in the radio-metadata repository. Every platform-received logical uplink produces one RX and one INGEST provenance record, and a STORE record is created only when the measurement is committed to the database:

$$N_{RX} = N_{INGEST} = N_{rx}, \qquad N_{STORE} = N_{store} \tag{2.12}$$

The number of provenance records associated with opportunity i is $p_i = 2z_i + y_i$, and the total number of provenance records is

$$N_{prov} = \sum_{i=1}^{M} p_i = 2N_{rx} + N_{store}. \tag{2.13}$$

The model separates communication-level availability, platform intake, and platform-side processing success:

$$C_{observed} = \frac{N_{uplink}}{M}, \quad P_{intake} = \frac{N_{rx}}{N_{uplink}}, \quad P_{store|rx} = \frac{N_{store}}{N_{rx}},$$
$$C_{effective} = \frac{N_{store}}{M} = C_{observed}\, P_{intake}\, P_{store|rx} \tag{2.14}$$

The quantity $C_{observed}$ corresponds to the packet delivery ratio commonly used in LoRaWAN performance studies [159]. The quantity N_{uplink} is observable only if the network-server event log introduced in Section 2.2 is retained. Without it, N_{uplink} can only be approximated by N_{rx}, which amounts to assuming $P_{intake} = 1$ and counting platform intake losses as communication losses. The deployment values of these quantities are reported in Chapter 4.

2.4.4 Integrity verification model

The system distinguishes communication-layer integrity from platform-side provenance integrity. LoRaWAN provides protocol-level protection during wireless transmission [52]. After uplink reception, the proposed framework records and verifies the platform-side provenance history. Each provenance record is linked to the previous record through a keyed SHA-256 hash (HMAC-SHA-256) , and it forms a per-device hash chain. The design follows the classic principle of a cryptographically secure, append-only record. Each record is cryptographically linked to previous records, so that retrospective modification, deletion, or reordering becomes detectable [153].

Let $\mathrm{Canon}(r_{d,k})$ denote the deterministic canonical representation of the integrity-relevant fields of provenance record $r_{d,k}$. The fields are serialised as compact JSON with lexicographically sorted keys and fixed separators and encoded in UTF-8, following the approach of the JSON Canonicalization Scheme [160]. The representation includes the predecessor hash $prev_hash_{d,k}$ and excludes the field $record_hash_{d,k}$. This design follows the classical principle of tamper-evident append-only logging, where each record is cryptographically linked to previous

records so that retrospective modification, deletion, or reordering becomes detectable [150].

The construction follows the hash-linking principle introduced for digital time-stamping [161] and applied to audit logs in [153] and [115]. During provenance-record generation, the hash of record $r_{d,k}$ is computed as:

$$h_{d,k} = H\left(h_{d,k-1} \parallel \text{Canon}\left(r_{d,k}\right)\right), \quad h_{d,0} = 0^{256} \tag{2.15}$$

where $H(\cdot)$ is the chain function, instantiated as the keyed function defined at the end of this subsection.

The hash fields stored in each record are assigned as follows:

$$prev_hash_{d,k} = h_{d,k-1}, \quad record_hash_{d,k} = h_{d,k}$$

For the verification of a complete chain of a device, the expected hash of the predecessor is defined as

$$expected_prev_{d,k} = \begin{cases} h_{d,0}, & k = 1 \\ record_hash_{d,k-1}, & k > 1 \end{cases} \tag{2.16}$$

For the verification of a selected segment of the chain that begins with record $r_{d,s}, expected_prev_{d,s}$ the value is supplied to the verifier as a trusted initial boundary. For each following record in the segment, the expected predecessor is the stored record hash of the previous record, in the verified order.

For each stored provenance record, the expected hash is recomputed as:

$$h_{d,k} = H\left(expected_prev_{d,k} \parallel \text{Canon}\left(r_{d,k}\right)\right) \tag{2.17}$$

A provenance record is valid if both its content hash and predecessor link are consistent:

$$Valid\ (r_{d,k} | expected_prev_{d,k}) \Leftrightarrow (\hat{h}_{d,k} = record_hash_{d,k}) \wedge (prev_{hash_{d,k}} = expected_prev_{d,k}) \tag{2.18}$$

The complete provenance chain of device d is valid when all records satisfy this condition:

$$Valid(C_d) = \bigwedge_{k=1}^{n_d} Valid(\, r_{d,k} | expected_prev_{d,k}) \tag{2.19}$$

Since the verifier processes each provenance record only once, the number of verification operations per record and the asymptotic complexity of the verification are

$$K_{verify} = \sum_{d \epsilon D} n_d = N_{prov},\ T_{verify}(N_{prov}) = \Theta(N_{prov}) \tag{2.20}$$

An integrity violation is identified whenever the chain-validity condition is not satisfied:

$$IntegrityViolation(C_d) \Leftrightarrow \neg Valid(C_d) \tag{2.21}$$

Under the stated assumptions of a trusted platform, the modification of any field relevant for integrity changes the recomputed hash of the record. The deletion, insertion or reordering of intermediate records, without a consistent replacement of the protected verification state, breaks the expected relation with the predecessor or produces a mismatch in the record hash. Such inconsistencies are detected when the affected chain, or segment of the chain, is verified.

However, hash linking alone is not able to detect the deletion of records from the end of a chain, since the remaining prefix may stay internally consistent [153]. In order to provide a protected reference to an earlier state of the chain, the platform stores a checkpoint of the chain head in the trusted audit subsystem.

Let t_c denote the checkpoint-creation time, n_c denote the chain length at that time, and h_{d,n_c} denote the corresponding chain-head hash. The checkpoint is represented as

$$CH_{d,t_c} = \langle device_id_d, h_{d,n_c}, n_c, t_c \rangle \tag{2.22}$$

Note that the notation in Formula (2.22) represents a stored checkpoint record, rather than an additional cryptographic function (i.e., no new hash or signature is computed for it). In the implemented prototype, the checkpoint is mainly protected by means of the trusted audit and verification subsystem.

At a later time of verification $t_v \geq t_c$, the current chain is consistent with the checkpoint when the checkpointed prefix is still present, and the record at the checkpointed position keeps the protected hash:

$$\begin{aligned} &\text{CheckpointConsistent}\big(C_d(t_v), CH_{d,t_c}\big) \Leftrightarrow \\ &\quad |C_d(t_v)| \geq n_c \ \wedge\ record_hash_{d,n_c} = h_{d,n_c} \end{aligned} \tag{2.23}$$

If the current chain contains fewer than n_c records, tail truncation relative to the checkpoint is detected. If a record remains at position n_c but its hash differs from the checkpointed hash, modification or reconstruction of the checkpointed prefix is detected.

This mechanism protects only the chain prefix covered by the latest trusted checkpoint. Records appended after the checkpoint are not protected against tail truncation until a new checkpoint is created. A fully malicious administrator who can replace both the provenance chain and the trusted checkpoint remains outside the protection scope of the model.

Keyed chain. Since is public, a chain computed with an unkeyed hash function, $H = \text{SHA-256}$, can be recomputed by any party with write access to the provenance repository. In such case, a consistent rewrite of the records appended after the latest checkpoint remains undetected, and only the checkpointed prefix is protected by Formula (2.23). For this reason, the framework instantiates the chain function with HMAC-SHA-256 [162], [163]:

$$h_{d,k} = \text{HMAC}_K\left(h_{d,k-1} \parallel \text{Canon}\left(r_{d,k}\right)\right), \quad h_{d,0} = 0^{256}, \tag{2.24}$$

where the key K is held by the provenance service and by the verifier, outside the provenance repository, and the verification by means of Formula (2.17) uses the same keyed function. An adversary who can write to the repository, but does not know K is not able to produce valid hashes for modified, inserted or reordered records. As a result, such changes are detected also after the latest checkpoint. However, the detection of tail truncation still requires the checkpoint. The additional cost is a constant number of evaluations of the SHA-256 compression function per record, together with the management of the key. a key identifier recorded in $meta_{d,k}$ supports key rotation. The chains of the original 12-month deployment were computed with unkeyed SHA-256; the keyed chain is evaluated on the replayed deployment dataset in Chapter 4.

2.4.5 Missing-data attribution model

Formula (2.14) quantifies how much data is missing but not where in the pipeline it was lost. Requirement R5 additionally requires each missing expected measurement to be attributed to a pipeline stage, or to be explicitly marked as undetermined when the recorded evidence does not permit attribution. The model below uses only evidence that the framework already records: the frame counters and reception times of observed uplinks, the status of RX and INGEST records, and the network-server event log.

Let device d have nominal transmission interval Δt_d and K_d expected slots $k = 0, \dots, K_d - 1$ during the observation period. Let $F_d = \{(t_j, f_j)\}$ be the time-ordered set of frames of device d accepted by the network server, where t_j is the network-server reception time and f_j is the uplink frame counter (FCnt). When the network-server event log is available, F_d is taken from it; otherwise, only frames that have an RX record are known.

Slot mapping. Device clocks drift and transmission times are jittered, so frames are mapped to slots using a tracked schedule phase φ:

$$k_j = \text{round}\left(\frac{t_j - \varphi_{j-1}}{\Delta t_d}\right), \qquad \varphi_j = \varphi_{j-1} + \alpha\left(t_j - \varphi_{j-1} - k_j \Delta t_d\right), \tag{2.25}$$

where φ_0 is estimated from the first observed frames and $\alpha \in (0,1]$ is a smoothing factor; the update is a form of simple exponential smoothing [164] ($\alpha =$

0.05). The update follows slow clock drift without accumulating the jitter of individual frames.

Platform-stage classes. A slot k to which a frame is mapped receives the label

$$c_k = \begin{cases} \text{STORED}, & \text{RX, INGEST(OK) and STORE records exist,} \\ \text{P1}, & \text{RX and INGEST(FAIL) records exist, no STORE record,} \\ \text{P0}, & \text{frame in the network-server log, no RX record.} \end{cases} \quad (2.26)$$

Class P1 denotes a processing failure (decoding or validation) and class P0 a platform intake loss, for example an interruption of the MQTT integration or the collector, or a rolled-back storage transaction.

Upstream gaps. Let (k_a, f_a) and (k_b, f_b) be two consecutive frames of F_d, and let

$$S = k_b - k_a - 1, \qquad L = f_b - f_a - 1 \quad (f_b > f_a), \quad (2.27)$$

where S is the number of empty slots between them and L is the number of frames that the device transmitted but the network server did not accept. Since the device increments FCnt with every new uplink, L frames were lost upstream and the remaining $S - L$ slots had no transmission. If the frame counter is missing from some observed frames, these frames are mapped to slots by their reception time but do not bound gaps; for consecutive frames with known counters that enclose m frames without a counter, $L = f_b - f_a - 1 - m$, and a gap in which the counter was reset is labelled U3. The empty slots are labelled

$$\begin{cases} \text{all } S \text{ slots U1}, & L \geq S, \\ \text{all } S \text{ slots U2}, & L = 0, \\ \text{MIXED: } L \text{ slots U1 and } S - L \text{ slots U2}, & 0 < L < S, \end{cases} \quad (2.28)$$

where U1 denotes upstream loss (radio loss, collision, gateway or backhaul outage, rejection by the network server) and U2 denotes that the device did not transmit (power failure, sensor fault, device outage). In a MIXED gap, the positions of the U1 and U2 slots are unknown.

Frame-counter reset. If $f_b \leq f_a$, the device has re-joined the network (OTAA) and its counter has restarted from zero. The f_b frames transmitted after the re-join and before (k_b, f_b) were lost upstream, whereas for the earlier part of the gap it cannot be determined whether the device was silent or transmitted under the previous session. Hence

$$n_{U1} = \min(f_b, S), \qquad n_{U3} = S - n_{U1}, \quad (2.29)$$

and the last n_{U1} slots of the gap are labelled U1 and the preceding n_{U3} slots U3 (undetermined).

Boundary gaps. Slots before the first and after the last frame of F_d are labelled U3, because no frame-counter evidence bounds them:

$$c_k = \text{U3}, \qquad k < k_{first} \text{ or } k > k_{last} \quad (2.30)$$

Every expected slot thus receives exactly one label, and the expected slots of device d decompose as

$$K_d = n_{STORED} + n_{P1} + n_{P0} + n_{U1} + n_{U2} + n_{MIXED} + n_{U3} \qquad (2.31)$$

We refer to the share of missing slots attributed to a single stage as the attribution coverage, by analogy with the coverage in classification with a reject option. In our case, the undetermined class U3 plays the role of the rejection [165]:

$$Cov_d = \frac{n_{P1}+n_{P0}+n_{U1}+n_{U2}}{K_d - n_{STORED}} \qquad (2.32)$$

For MIXED gaps, the number of U1 and U2 losses is known, even though their positions are not. As a result, the estimates per class at the level of counts remain available.

The model relies on five assumptions. The nominal interval is known and constant. FCnt increases by one for every new uplink, while the retransmissions reuse the same counter and are deduplicated by the network server. FCnt restarts from zero after a re-join. The jitter of the transmission time is small with respect to . Finally, the event log of the network server contains every accepted frame, and is written independently of the provenance collector. When such assumptions are violated, the share of U3 and MIXED labels mainly increases. Moreover, a large timing jitter can also cause a wrong attribution. The above effects are quantified in Chapter 4.

Figure 2.5 shows the application of the above rules on a sequence of 21 expected slots of one device, with a nominal interval of $\Delta t = 600$ s. The upper row shows the frames recorded in the event log of the network server, the middle row the provenance records committed by the platform, and the lower row the class assigned to each slot.

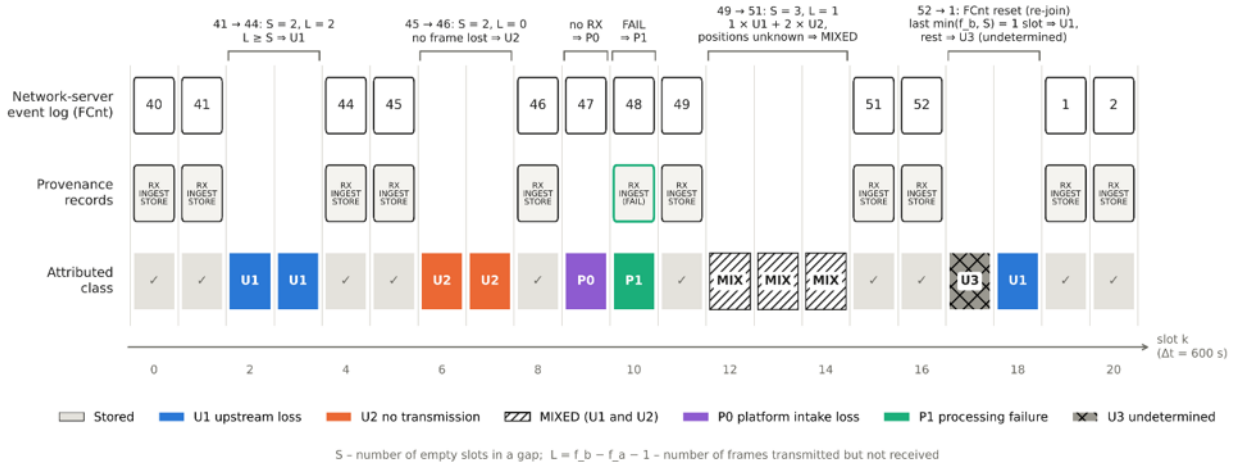


Figure 2.5 - Example of stage-level attribution of missing data for one device over 21 expected slots

Between frames 41 and 44, two slots are empty and two frame counters are missing ($S = 2$, $L = 2$), so both slots are labelled U1. Between frames 45 and 46, the counters are consecutive ($L = 0$), so the device did not transmit in the two empty slots (U2). Frame 47 is present in the network-server log but has no RX record and is labelled P0; frame 48 has RX and INGEST(FAIL) records without a STORE record and is labelled P1. Between frames 49 and 51, one frame was lost in three empty slots ($S = 3$, $L = 1$); the number of slots in each class is known but their positions are not, so the slots are labelled MIXED. After frame 52 the counter restarts at 1, which indicates a re-join: by Formula (2.29), the last slot before frame 1 is labelled U1 and the preceding slot U3. In total, 11 of the 21 expected measurements are missing; 7 of them are attributed to a single stage, 3 are MIXED, and 1 is undetermined, so the coverage given by Formula (2.32) is 7/11.

The network server event log is needed to separate P0 from U1. Without it, the analyst cannot see a frame that was accepted by the network server but not recorded by the platform. In Figure 2.5, for example, frame 47 would appear as a gap between frames 46 and 48 in the RX sequence, and its slot would be labelled U1. With the log, such losses at platform intake are no longer mistaken for radio losses, and they are identified as platform failures.

The formal model defines five correctness properties, on which the implementation and the later evaluation of the system are based. Table 2.4 summarizes such properties, their mathematical basis and the mechanisms that verify or support each of them in the proposed architecture.

Table 2.4 - Correctness properties supported by the formal provenance model

Property	Formal basis	Verified or supported by
1	2	3
Traceability	Backward and forward lineage sets ($L^{-}(e)$) and ($L^{+}(e)$), defined in Formula (2.6) and (2.7)	Provenance graph reconstruction and lineage queries
Consistency	Entity-Activity-Agent relations: wasGeneratedBy, used, wasAssociatedWith, and wasDerivedFrom	PROV-consistent provenance model and record semantics
Tamper evidence	Hash-linked provenance chain, defined in equations (2.16), (2.17), and (2.19)	Chain verification through hash recomputation and prev_hash consistency checks
Linkability	Stable entity identifiers across lifecycle stages, including input_eids and output_eid	Identifier scheme and entity-to-record mapping
Attributability	Three-level observation model, Formulas (2.9)-(2.14), and attribution rules, Formulas (2.25)-(2.32), with an explicit undetermined class	Network-server event log, FCnt sequence, and RX/INGEST status

The above properties link the formal provenance model with the mechanisms implemented in the platform. Traceability is supported by the reconstruction of graph views, while the relations compatible with PROV provide a consistent representation of the dependencies between entities, activities and agents. The hash-linked provenance records provide tamper evidence for the stored history of processing, whereas the stable identifiers of the entities preserve the links along the whole lifecycle of the measurement on the platform side. Moreover, the attribution rules extend such properties to the missing data. So, it means that every expected measurement is either traced to its stored record, assigned to the stage of the pipeline where it was lost, or explicitly classified as undetermined. In summary, the five properties provide the formal basis for the implementation and the experimental evaluation of the proposed provenance-aware LoRaWAN framework.

2.5 Provenance storage and verification

The developed architecture stores the provenance history in the form of an ordered append-only chain. Each entry here marks an important milestone in the data lifecycle: from receiving and validating a packet to converting, archiving, or checking it on a server. According to Section 2.4.4, the records are linked to each other by cryptographic hash links. It provides transparent tamper-evidence and step-by-step integrity verification, as well as helps to completely remove the complex computing load from low-power sensors.

The framework divides the provenance history into two complementary levels. The logical layer is based on the classical W3C PROV (entity-activity-agent) semantics and is responsible for auditing cause-and-effect relationships and end-to-end analytics. The operational layer consists of physical append-only chains in the repository, customized for efficient compression, fast verification, and security auditing.

The distinction between linear storage and graph visualization is the foundation of architecture. Chains are ideal for integrity control (through recalculation of hashes and reconciliation of prev_hash pointers), whereas the logical graph unfolds dynamically and only on request to trace the history or display the structure on the screen. A summary comparison of both representations is shown in Table 2.5.

Table 2.5 - Comparison of chain storage and graph-view provenance representation

Aspect	Chain storage	Graph view
1	2	3
Main purpose	Integrity verification and tamper evidence	Traceability queries and lineage interpretation
Structure	Ordered per-device provenance records	PROV-consistent nodes and edges
Primary unit	Provenance record	Entity, activity, and agent
Query direction	Sequential record-by-record verification	Backward and forward tracing

Continuation of Table 2.5

Aspect	Chain storage	Graph view
1	2	3
Main operation	Hash recomputation and prev_hash consistency checking	Lineage reconstruction through input_eids and output_eid
When used	Audit, verification, and integrity checks	Explanation, provenance queries, and visualisation
Main advantage	Efficient tamper-evident storage	Representation of data lineage

As shown in Table 2.5, the separation of levels allows the framework to keep a lightweight append-only structure without losing the readability of the links. The records are first checked sequentially for tampering, and then a PROV-compatible graph for backward and forward tracing is built from them.

An important architectural nuance is that hash chains are built separately for each device, and cross-platform requests are linked through stable entity IDs. The node-based model preserves the exact order of sensor events and passes verification without problems in case of packet loss, network delays, or communication outages. At the same time, end-to-end analytics between different devices are not disrupted, since logs contain direct links to incoming and outgoing entities. The platform supports both reconstruction of the history of an individual sensing node and cross-device provenance queries based on stable entity identifiers.

Losses do not break the chain: an uplink that never reaches the platform produces no provenance record and therefore no gap in the hash linkage. Missing measurements are identified separately, by comparing the recorded frames with the expected transmission schedule and the frame-counter sequence.

Each data element is represented as an entity with a stable identifier assigned within the platform. These identifiers are used to ensure end-to-end connectivity at all stages. Starting from reception and decoding to verification, storage and reconstruction of the provenance history. The following identifiers are involved in hash chains:

- The Entity Identifier (*eid*) defines a specific data object (raw measurement, uplink payload, decoded application message, database entry or aggregated metric).
- Record Identifier (*rid*) - uniquely identifies the provenance record itself in the chain.
- Agent ID (*agent_id*) - indicates the responsible component or entity (sensor node, gateway, network sector, or platform service).
- Process identifier (*pid*) - if necessary, designates a specific processing module, algorithm version, or service instance.

The model transfers the maintenance of stable identifiers to the platform level, so as not to drive heavy metadata over the airwaves. At the hardware level, the framework makes do with standard compact LoRaWAN identifiers, DevEUI values and frame counters. The permanent identifiers of entities and records are assigned to logs already on the server, immediately after receiving packets. This approach minimizes overhead costs on the sensor side but fully preserves reliable communication and data traceability throughout the processing pipeline.

Each provenance record is a structured object with three types of fields: mandatory, conditional and optional. The mandatory fields provide the basic traceability and integrity, the conditional fields are added only for specific types of events, and the optional fields are used for diagnostics and operational audit. The fields are listed in Table 2.3.

Depending on the type of event and the available metadata, the conditional part can be supplemented with the deployment zone, gateway ID, RSSI and SNR values, frame counter, payload version, decoding status, verification flags or processing logs. In this way, the provenance history remains useful even when the sensor itself provides little data. The format is also open for future extensions, since new metrics can be added without changing the logic of the hash chain. Such property is important for long-term auditing and for changes in the data schemas over time.

The framework defines a set of event types that covers the smart city data pipeline. A minimal set of events is enough for correctness, while richer events allow deeper audits. The conceptual event types are:

- SENSE: a device produces a raw measurement entity.
- TX: a device transmits an entity through the LoRaWAN stack.
- RX: the LoRaWAN infrastructure receives a message and adds the reception metadata.
- INGEST: a platform service ingests a decoded message into the system.
- TRANSFORM: a processing component produces a new entity from one or more entities.
- STORE: the system commits an entity and its provenance to the storage backend.
- PUBLISH: the system publishes an entity to applications or external consumers.

In the implemented framework, provenance is captured on the platform side, where only three event types are recorded: RX, INGEST and STORE. The other event types (SENSE, TX, TRANSFORM and PUBLISH) are kept in the conceptual model for future extensions, and for deployments where provenance can also be captured on the device, processing or application side.

The chain grows as the framework processes the data. Each event that changes the state or the interpretation of the data creates a new provenance record, and each transformation that produces a derived value creates a new output entity identifier. As a result, raw measurements, decoded messages, stored records and derived indicators are never confused, and the lineage can be reconstructed without ambiguity.

The provenance chain supports two query directions corresponding to $L^{-}(e)$and $L^{+}(e)$, as defined in Formula (2.7). Backward tracing starts from an output entity and retrieves the records and source entities that produced it. Forward tracing starts from a raw measurement entity and finds all the derived entities that depend on it. Both directions rely on the explicit input_eids and output_eid references stored in the provenance records.

Queries on the platform are based on indices built over three fields: the entity identifier, the device identifier and the timestamp. With them, an analyst can either retrieve the provenance chain of one device or compare several devices when

studying the smart city system as a whole. Indexing at the perception layer is kept simple, since the nodes only send compact observation metadata, and the full operational provenance chain is built later on the platform.

Figure 2.6 illustrates the per-device hash-linked provenance chain used in the framework. For each processed measurement, the platform records RX, INGEST, and STORE events, linked through predecessor hashes.

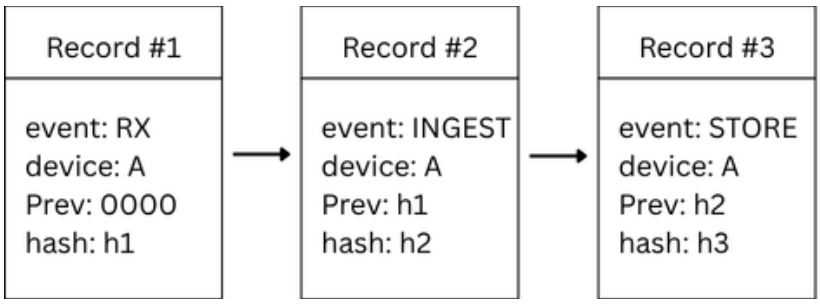


Figure 2.6 - Per-device append-only hash-linked provenance chain for a single sensing node

Figure 2.6 shows that the hash chain grows sequentially as each uplink is processed. The three types of events, RX, INGEST, and STORE, reflect the stages of the data lifecycle on the server side. Those are receiving a radio packet, processing a decoded measurement, as well as storing it in storage. Each record stores the device ID, event type, timestamp, entity references, and hash fields. Since *record_hash* is calculated over the entire content, also including *prev_hash*, it is formed a strict cryptographic relationship between the records. Any change that is not followed by consistent recomputation of the affected downstream hashes produces a detectable mismatch. Detection of tail truncation additionally depends on comparison with a previously stored trusted chain-head checkpoint whose integrity is assumed under the platform trust model.

The framework supports different levels of provenance across layers. At the device layer, provenance remains compact and includes identifiers, timestamps, and a zone identifier. At the stage of communication with the gateway, the radio broadcast parameters (RSSI, SNR, SF coefficient, and gateway ID) are cut off and stored in a separate database of radio data, linked to history through stable entity identifiers. The platform layer stores full provenance records for activities such as reception, ingestion, and storage. As a result, we do not overload peripheral devices with unnecessary traffic, but we keep all the conditions on the server for a detailed search and reliable security audit.

The framework targets low-cost microcontroller sensing nodes and LPWAN connectivity, where memory, payload size, airtime, and battery capacity are constrained. No provenance computation is therefore performed on end devices: provenance capture and integrity protection are implemented on the platform side.

The framework distinguishes three protection targets: the post-ingestion integrity of provenance-linked monitoring records; the integrity of the provenance records themselves, i.e. the ability to detect their modification, deletion, insertion, or reordering; and the integrity of the metadata that the provenance records carry,

namely identifiers, timestamps, frame counters, processing outcomes, and entity references. Gateway-level radio metadata are stored separately and are not covered by the hash chain (see below).

The LoRaWAN protocol secures the radio transport by means of authentication, encryption, the Message Integrity Code (MIC) and frame counters. Once the data enter the platform, however, the protocol neither traces them nor protects the stored logs from later modification. To cover this gap, the proposed framework adds a lightweight protection on the server side, based on append-only storage, access control and record chains linked by keyed SHA-256 hashes (HMAC).

At the core of the architecture are the hash chains kept for each device. Every provenance record stores the hash of the previous record, together with its own hash computed over its deterministic content. As a result, the records depend strictly on each other in sequence, and any tampering becomes visible. No blockchain consensus or distributed ledger technology (DLT) is needed for this purpose.

Hash linking only works if the storage layer remains append-only. For this reason, provenance records are written through a dedicated provenance service, and direct update or delete operations are restricted. Only a dedicated provenance service is allowed to write provenance records, while direct updates and deletions are restricted. Existing records are never overwritten: any correction is added as a new compensating record. The chain therefore stays intact and can always be audited.

The chain-head checkpoints defined by Formula (2.22) provide the reference against which tail truncation is detected. They are stored outside the provenance repository, in a store to which the provenance service and database accounts have no write access; in the prototype, this is the audit subsystem of the platform, and in a production deployment it can be a write-once store or a store under a different administrative domain. Checkpoints are created periodically with interval Δ_c. Records appended after the latest checkpoint are exposed to undetected tail truncation, so Δ_c bounds the exposure window; other changes to these records remain detectable because the chain is keyed. The network-server event log (Section 2.2) provides a further independent reference: if all provenance records of an uplink are deleted and the chain is recomputed consistently, the frame remains in the log without an RX record and is reported as an unexplained platform intake loss (class P0) instead of disappearing without trace.

Tamper evidence for the metadata is obtained by including the fields relevant for integrity in the canonical representation of each record, before the hash is computed. Which fields are included depends on the event type: device identifier, timestamp, zone identifier, frame counter, uplink key, processing outcome and entity references. If any of them is changed, the recomputed hash no longer matches, and the next verification reports it.

Radio metadata at gateway level (gateway identifier, RSSI, SNR, spreading factor and data rate) are left out of the canonical record on purpose. The same logical uplink may be heard by several gateways, and some reception entries arrive only after the RX record has already been committed. In such case, the collector appends them to the radio metadata repository and links them to the existing uplink through the uplink key. Putting them into the RX hash would mean either rewriting a committed

record, or making the hash depend on the order in which receptions arrive. The radio metadata are therefore connected to the provenance chain by the uplink key, but are not tamper-evident. The trade-off is acceptable, since such metadata only describe the link quality and have no effect on the stored measurement values. A separate hash-linked chain for reception records would protect them as well, and is left as a possible extension.

The platform also includes the principle of minimizing metadata so as not to unnecessarily disclose the internal nuances of system deployment. Data packets from the devices themselves weigh a minimum and do not contain unnecessary information about the provenance. The entire detailed operating system is radio communication indicators and internal service identifiers. It is intercepted on the server side, and access to these logs can be tightly closed, leaving it only for audit or integrity verification roles. This compromise allows us to find the perfect balance between traceability, the technical needs of the network and the confidentiality of metadata.

The framework verifies the provenance history both periodically and on demand. On-demand verification is triggered by provenance queries, dataset exports, or audits; periodic verification is triggered by a platform timer. During verification, the system recomputes the record hashes and checks predecessor continuity for each device chain. Because chains are maintained per device, a single sensing stream can be verified independently, with cost linear in the number of its records (Formula (2.20)). The measured verification time is reported in Chapter 4.

The integrity verifier recomputes the keyed (HMAC-SHA-256) record hashes and checks the continuity of prev_hash/record_hash relationships within each per-device provenance chain. Duplicate-event handling and replay-related checks are performed separately during ingestion and LoRaWAN processing. Figure 2.7 summarizes the relationship between the considered threat scenarios, the corresponding detection mechanisms, and the resulting audit workflow.

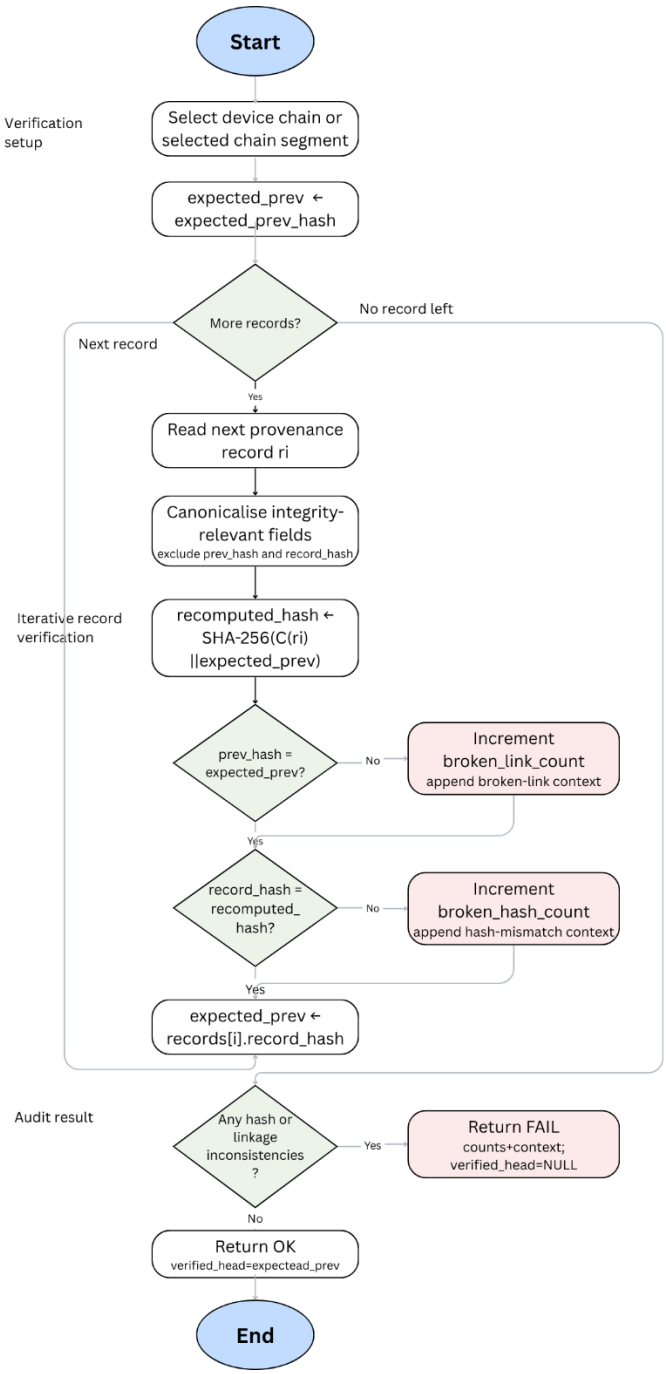


Figure 2.7 – Per-device provenance-chain verification and audit workflow

Figure 2.7 illustrates the verification workflow applied to a complete per-device provenance chain or a selected chain segment. The verifier recomputes each record hash from the canonical non-hash fields and the expected predecessor hash. It then compares the recomputed value with the stored record_hash and checks whether prev_hash corresponds to the expected predecessor. A mismatch is recorded as either a hash-integrity failure or a broken chain link. Duplicate-event and replay checks are performed separately during ingestion, as described in Algorithm 3.1. Provenance graph views are reconstructed on demand after successful verification.

If a nonconformity is detected, the framework marks the affected provenance records and logs the verification result in the audit log. Depending on the deployment policy, the affected records can be excluded from high-trust analytical workflows

until the incident is investigated. These procedures support auditing and diagnosis of provenance integrity; they complement, but do not replace, intrusion detection and access-control mechanisms.

Conclusions for Chapter 2.

In this chapter we presented a layered provenance-aware architecture for air quality monitoring based on LoRaWAN. Provenance is captured on the platform side by means of the standard integration of the network server, so the firmware of the end devices, the payload format and the gateways remain unchanged. The provenance of each uplink is represented with the Entity-Activity-Agent semantics of W3C PROV. More specifically, the evidence of gateway reception is modelled as an entity, and a directed provenance graph that supports backward and forward tracing is reconstructed from the record chains of each device.

The integrity of the stored provenance is provided by hash chains per device, keyed with HMAC-SHA-256, and by checkpoints of the chain heads stored outside the provenance repository. The security claim is defined precisely as tamper evidence under the assumption of a trusted platform, with explicit levels of adversary and threats that remain out of scope.

The formal model gives the provenance volume as $2N_{rx} + N_{store}$ records, with a verification cost that is linear in the number of records and no overhead on the device side. Last, we introduced a model for the attribution of missing data. By means of the frame counters, the reception times, the state of the records and the event log of the network server, the model assigns each missing expected measurement to a stage of the pipeline (i.e., device, upstream, platform intake or processing) or classifies it as undetermined.

3 IMPLEMENTATION OF THE PROVENANCE-AWARE IOT FRAMEWORK

3.1 Architecture of the implemented platform

The architecture of the implemented system follows the standard layered structure of a LoRaWAN deployment [49], combined with a stack for provenance and data management on the platform side. When designing the system, we kept the sensing nodes as simple as possible, so the nodes based on microcontrollers only measure air quality parameters periodically and transmit them as compact hexadecimal uplink payloads. The gateways receive the uplinks and forward them to the LoRaWAN network server through the standard LoRaWAN mechanisms. The network server also plays a significant role in the system, since it manages the LoRaWAN sessions, performs the integrity checks at protocol level and forwards the uplink events to the application platform. However, the network server does not keep any history of how the data are processed after their reception. Despite this, the provenance information can be recorded on the platform side, without additional hardware or changes in the protocol.

The platform receives the uplink events through MQTT and the network server publishes an event for each uplink with the device identifier, the frame counter, the transmission parameters as well as with the reception metadata (e.g., RSSI, SNR, spreading factor and data rate). The platform subscribes to such events and passes each of them to the provenance collector. The received the event, the collector parses the message, decodes the payload, detects duplicates, validates the values and stores the results.

For each logical uplink, the collector assigns a stable entity identifier to the decoded measurement. Two repositories are used for this purpose: the measurement repository for the decoded values, and the radio_rx repository for the radio metadata at gateway level. With the aforementioned identifier, the collector then creates the RX, INGEST and STORE provenance records, and appends them to the append-only chain of the device. Adding such records to the LoRaWAN packets would have a negative impact on the airtime of the nodes. For this reason, all the records are created on the platform once the uplink has been received, so the payload stays compact and no extra airtime is needed. Figure 3.1 shows the data flow and the main components of the implemented platform.

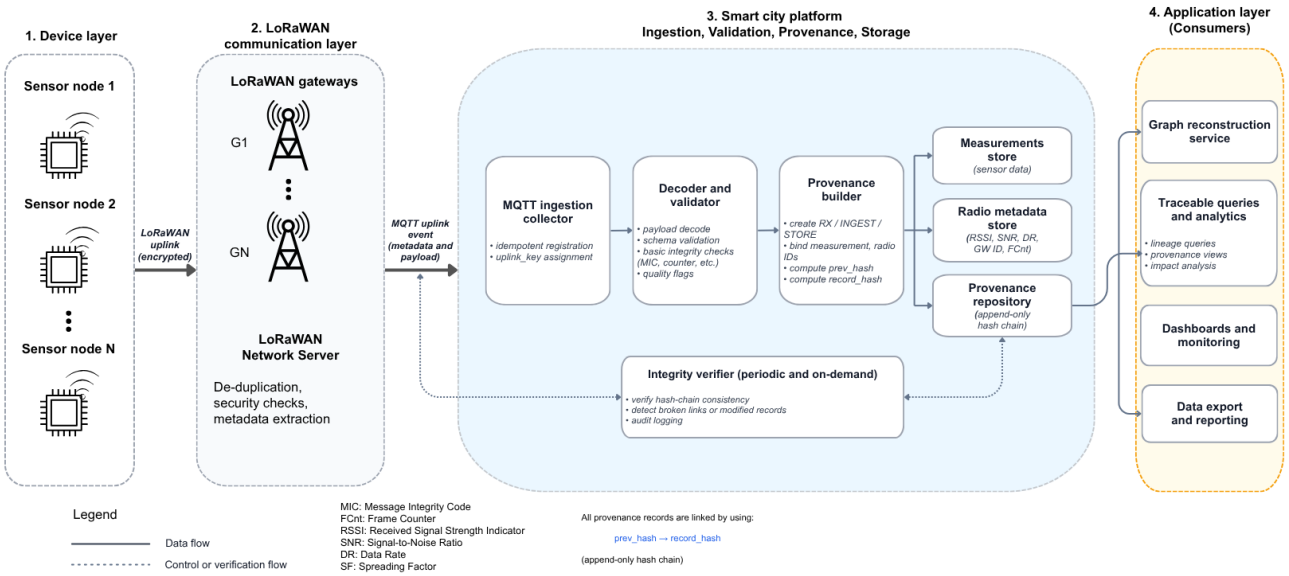


Figure 3.1 - Internal platform architecture and data flow

In the data collection phase, the system was deployed on an edge server (Intel NUC 9 Pro, 32 GB RAM, Ubuntu 22.04) over a period of 12 months. The database was created with MySQL 8.0, and the deployment generated 403,900 provenance records. For the analysis of integrity verification, we exported the database to a CSV file and processed it with Jupyter Notebook and Python 3.10 on a laptop (Intel Core i5-10300H, 16 GB RAM, Windows 10 Pro). We decided to conduct the validation on a separate computer in order to avoid any additional computational load on the production server. In this way, the verification could also be evaluated independently of the production environment. Python was used to test the logic, together with the pandas library to load the data, as well as the built-in hashlib library to compare the SHA-256 hashes. The execution time was measured with time.perf_counter(), and each scenario was repeated 10 times. The results are reported in Chapter 4.

The ingestion service subscribes to the MQTT uplink topics of the network server, and processes the events in near real time. The database indices also play a significant role in performance. In particular, the indices on uplink_key, device_id, timestamp and eid, as well as on the identifiers of the provenance records, speed up the ingestion and the chain verification, together with the traceability queries. Figure C.1 shows the workflow of the processing of MQTT uplinks. Table 3.1 lists the implemented components and links each of them to the components of the formal model in Formula (2.1). Note that the gateway and the network server are grouped together, since both of them provide the reception and protocol metadata.

Table 3.1 - Implemented components and responsibilities in the provenance-aware system

Component	Main responsibility	Provenance role
1	2	3
End device (D)	Senses the environmental variables and transmits compact LoRaWAN uplinks	Produces the initial logical observation

Continuation of Table 3.1

Component	Main responsibility	Provenance role
1	2	3
LoRaWAN gateway and network server (G, N)	Receives the uplinks, validates the LoRaWAN metadata and publishes the MQTT events	Provides reception and communication evidence
MQTT broker (N)	Delivers uplink events to the platform	Enables platform-side provenance capture
Provenance collector (P)	Parses, decodes, validates, assigns identifiers, and appends records	Create RX, INGEST, and STORE provenance records
Measurements database	Stores decoded measurement values and timestamps	Stores measurement entities referenced by provenance records
Radio metadata repository	Stores gateway-level radio metadata	Preserves reception evidence linked to uplinks
Provenance repository (P)	Stores hash-linked provenance records	Maintains append-only per-device provenance chains
Verification service (V)	Recomputes hashes and validates chain consistency	Detects modification, deletion, insertion, or reordering
Graph reconstruction service (Q)	Reconstructs graph views from chain records	Supports backward, forward, and cross-device tracing

As shown in Table 3.1, the implemented system and its components provide a complete pipeline for the acquisition of air quality data in smart cities, together with the preservation of metadata and the capture of provenance. In summary, the system covers the whole path of the air quality data, namely the LoRaWAN transmission, the MQTT ingestion, the separate storage of measurements, radio metadata and provenance records, as well as the integrity verification and the provenance queries.

3.2 LoRaWAN deployment and data acquisition setup

The setup includes end devices, gateways and a private network server, while the provenance functions run on a local platform. The aim of such setup is long-term monitoring under limited communication resources, with each stored measurement linked to its reception context and to its processing history.

The end devices are sensing nodes based on microcontrollers, equipped with air quality and environmental sensors. Each node operates according to a fixed sensing schedule and transmits compact hexadecimal uplink payloads with the measured variables, together with some lightweight information about the device status. The payload structure follows a compact and versioned format, in order to reduce the airtime and the energy consumption while keeping the compatibility with low data

rate LoRaWAN communications under the EU868 constraints [50]. Since the payload capacity and the airtime are limited, the provenance records are not embedded in the LoRaWAN packets. Instead, the platform reconstructs the provenance from the decoded payload and the reception metadata, as well as from the processing events on the platform side.

The end devices join the network and send uplinks according to the standard LoRaWAN procedures. One uplink may be received by several gateways. Such duplicate receptions are kept, since they show the coverage and the link quality, and they also support the reconstruction of provenance. In this sense, the gateways act only as packet forwarders and as sources of reception metadata, whereas the processing of provenance is performed on the platform.

The gateways forward the received uplinks to the private network server. The network server manages the LoRaWAN session and performs the checks at protocol level, like the MIC validation and the verification of the frame counter. Then, the network server publishes the uplink event to the platform through the MQTT integration. The network server is treated as the authoritative source of reception metadata, namely the gateway identifiers, the reception timestamps, the RSSI, the SNR, the spreading factor and the data rate. The platform uses such metadata for deduplication, for the analysis of communication quality and for the provenance records.

The experimental setup includes the end devices (i.e., air quality sensing nodes), the LoRaWAN gateways, a private LoRaWAN network server and an MQTT broker with the ingestion service. Moreover, the setup includes a measurements database, a repository for radio metadata and an append-only provenance repository, together with the services for verification and graph reconstruction.

End devices. Each sensing node is built on an Arduino Uno microcontroller board, which reads the sensors and formats the payload. A Dragino LA66 LoRaWAN Shield provides the LoRaWAN communication in the EU868 band. The uplinks are received by Dragino DLOS8N gateways and it forwarded to the local network server.

Sensing payload. Each node carries sensors suitable for long-term monitoring of urban air quality. More specifically, a ZEHS04 module measures CO, SO2, NO2 and O3, a ZH03B laser dust sensor measures PM1.0, PM2.5 and PM10, and a Bosch BME280 sensor measures temperature, relative humidity and pressure. In addition, a GPS module was used to record the location of the nodes. Table 3.2 summarizes the hardware configuration of the end devices, together with the sensor modules used.

Table 3.2 - End-device hardware configuration and sensing modules

Component	Model	Measured variables	Interface and output	Key operating notes
1	2	3	4	5
Microcontroller unit	ATmega328P	-	Integrated control and payload formatting	Low-power LoRaWAN sensing node with 10-minute reporting interval

Continuation of Table 3.2

Component	Model	Measured variables	Interface and output	Key operating notes
1	2	3	4	5
Particulate matter sensor	ZH03B	PM1.0, PM2.5, PM10	UART, PWM	Laser-based particulate sensor; warm-up required after power-on
Gas sensing module	ZEHS04 with ZE12 submodules	CO, SO_2, NO_2, O_3	UART, RS485	Multi-gas diffusion-type sensing module for urban air-quality monitoring
Environmental sensor	Bosch BME280	Temperature, relative humidity, pressure	I^2C, SPI	Provides environmental context and supports basic plausibility checks
GPS receiver	Integrated GNSS module	Time and coarse location reference	UART	Used intermittently during deployment and maintenance procedures
Sampling policy	-	-	-	Fixed sensing and transmission interval: 10 minutes

The detailed specifications of the sensors are provided in Appendix B.

The nodes were stationary and used an external power supply. The gas sensing module needs stable operating conditions, since its readings became unreliable after repeated power interruptions. For this reason, the gas sensors were powered continuously during the whole experiment. Table 3.3 summarizes the operating policy and the sampling schedule of the sensors.

Table 3.3 - Sensor operating policy and sampling schedule

Sensor/module	Operating policy	Native sensing interval	Reporting interval	Aggregation approach
1	2	3	4	5
ZEHS04 (CO, SO_2, NO_2, O_3)	Continuous operation	Approx. 1s	10 min	Mean value over a fixed observation window before uplink

Continuation of Table 3.3

Sensor/module	Operating policy	Native sensing interval	Reporting interval	Aggregation approach
1	2	3	4	5
ZH03B ($PM_{1.0}$, $PM_{2.5}$, PM_{10})	Continuous operation	1s	10 min	Mean value over a fixed observation window before uplink
BME280 (temperature, RH, pressure)	Continuous operation	Configurable	10 min	Mean value over a fixed observation window before uplink

The data collection policy was designed in order to find a compromise between the monitoring frequency and the communication overhead. The sensor measurements were collected continuously, and the platform generated one monitoring report every 10 minutes. In this way, the fixed reporting interval reduced the short-term fluctuations and limited the number of LoRaWAN transmissions.

Gateways. The deployment used two Dragino DLOS8N outdoor LoRaWAN gateways, based on the Semtech SX1302 concentrator. The gateways were installed on fixed outdoor structures, at a height of approximately 4 m above the ground level. Such configuration provided radio coverage of the monitoring area, and allowed the reception of the same uplink by both gateways under different communication conditions.

The gateways received the LoRaWAN uplink frames from the sensing nodes, and forwarded them to the private LoRaWAN network server by means of the standard LoRaWAN packet forwarding mechanism. No modifications were made to the communication protocol or to the gateway software.

A private LoRaWAN deployment was selected, since it provides direct access to the communication metadata required by the proposed framework. For each received uplink, the network server reports the gateway identifiers, the reception timestamps, the RSSI, the SNR, the spreading factor and the data rate, as well as the frame counter. Such metadata are stored together with the decoded measurements, and later incorporated into the provenance records.

Network server. ChirpStack [150] was installed on a locally controlled server, and used as the private LoRaWAN network server. Over-the-Air Activation (OTAA) was used in the final deployment, whereas Activation by Personalization (ABP) was limited to the early integration tests. Adaptive Data Rate (ADR) remained enabled during the whole experiment. ChirpStack published the uplink events and the reception metadata through MQTT, for their ingestion by the local platform.

LoRaWAN LoRaWAN protected the uplink communication by means of AES-128 encryption, the validation of the message integrity code, the frame counter checks and the management of session keys. ChirpStack completed such checks before forwarding each uplink to the local platform. Then, the provenance framework

processed the decoded payload and the reception metadata, without any change in the LoRaWAN protocol.

ChirpStack is also able to write every received uplink event to a separate database by means of its built-in PostgreSQL integration, independently of the MQTT stream. However, such integration was not enabled during the deployment. As a result, the event log of the network server required by the attribution model (Section 2.2) is not available for the deployment period, and the losses at platform intake cannot be separated from the upstream losses in the field data. For future deployments, enabling the integration is sufficient to provide the log.

Temporary losses of connectivity, changes in gateway reception and restarts of the devices occasionally caused delayed or missing uplinks. After a restart, the device performs a new OTAA join, and its frame counter starts again from zero. In any case, the ingestion service recorded the reception metadata and the processing events for every received uplink.

ChirpStack assigned a reception timestamp to each uplink, and the ingestion service used such timestamp to order the records and to build the provenance chain of each device. When available, the device timestamps were kept as auxiliary metadata. In this way, the clock drift of the devices could not change the order of the records.

Table 3.4 lists the main parameters of the deployment, together with the metadata fields used in the provenance records and in the experimental evaluation.

Table 3.4 - Summary of LoRaWAN deployment configuration and captured metadata

Category	Parameter	Configured value and captured fields	Relevance to the framework
1	2	3	4
End devices	Sensing variables	$PM_{1.0}$, $PM_{2.5}$, PM_{10}, CO, SO_2, NO_2, O_3, temperature, humidity, pressure	Defines the heterogeneous environmental dataset used for provenance-aware monitoring
End devices	Sampling policy	Fixed reporting interval (10 minutes)	Supports long-term time-series analysis and missingness detection
End devices	Payload structure	Compact versioned hex payload with lightweight flags	Reduces airtime overhead and maintains compatibility with low-data-rate LoRaWAN communication

Continuation of Table 3.4

Category	Parameter	Configured value and captured fields	Relevance to the framework
1	2	3	4
LoRaWAN settings	Region and frequency plan	EU868	Determines regulatory constraints and communication characteristics
LoRaWAN settings	Data rate and ADR	Adaptive SF/DR operation with ADR enabled	Supports analysis of communication variability and missingness patterns
Gateways	Gateway deployment	Two outdoor gateways with overlapping reception coverage	Enables reception-level duplication evidence and coverage interpretation
Network server	Server functions	Private ChirpStack deployment with MIC validation, FCnt tracking, and uplink deduplication	Provides communication metadata and protocol-level integrity validation
Integration	Uplink event interface	MQTT uplink-event stream exported by the network server	Enables non-invasive provenance-aware ingestion
Captured radio metadata	Reception context	gateway_id, reception timestamp, RSSI, SNR, SF, DR	Supports communication-quality analysis and reception traceability
Captured protocol metadata	Protocol context	Frame-counter references and deduplication outcomes	Supports ordering consistency and duplicate-reception interpretation
Time handling	Temporal reference	Network-server reception time as primary ordering reference	Maintains stable provenance-chain ordering under device clock drift
Platform storage	Measurements and provenance storage	Measurements repository, radio metadata repository, append-only provenance chains	Separates measurements, reception evidence, and provenance records

Continuation of Table 3.4

Category	Parameter	Configured value and captured fields	Relevance to the framework
1	2	3	4
Platform queries	Graph-view reconstruction	Graph representation reconstructed from provenance chains through entity identifiers	Supports backward and forward provenance tracing across processing stages

ChirpStack ran on the local server during the whole deployment. The network server published the uplink events through MQTT, and recorded the corresponding reception data. In addition, stable device identifiers linked each measurement to its radio record, as well as to its provenance chain.

3.3 MQTT ingestion and provenance capture

MQTT provides a publish-subscribe communication between the private LoRaWAN network server and the ingestion service of the platform. In this protocol model, the clients publish messages to named topics, and the subscribers receive such messages through an MQTT broker [166]. ChirpStack used this mechanism to publish an event for each received uplink.

Each event contained the device identifier, the application payload, the reception timestamp, the gateway identifier, the RSSI, the SNR, the spreading factor and the data rate, as well as the frame counter when available. The ingestion service subscribed to the uplink topics and processed each incoming event. The integration used the standard outputs of the network server and required no changes in the LoRaWAN packets or in the security procedures.

The platform runs a provenance collector for the incoming MQTT events. The collector first parses each message and decodes the compact hexadecimal payload. Then, the collector normalises the reception data, validates the decoded values, identifies the duplicate receptions and stores the resulting records. The payload version and the status fields indicate the presence of optional values, together with the state of the sensor at the time of ingestion.

For each logical uplink, the collector creates a stable uplink identifier and a persistent entity identifier for the decoded measurement. The measurement repository stores the decoded values, whereas the radio repository keeps the individual gateway receptions. In this way, the duplicate receptions remain available for the analysis of communication, without creating duplicate measurement entities.

Moreover, the collector records the RX, INGEST and STORE stages. Each record is added to the append-only chain of the corresponding device, after a successful validation and storage. Since the provenance data are created on the platform, they do not occupy any space in the LoRaWAN payload.

Figure 3.2 shows the workflow of the collector, from the MQTT reception and the payload decoding to the data storage and the creation of provenance records.

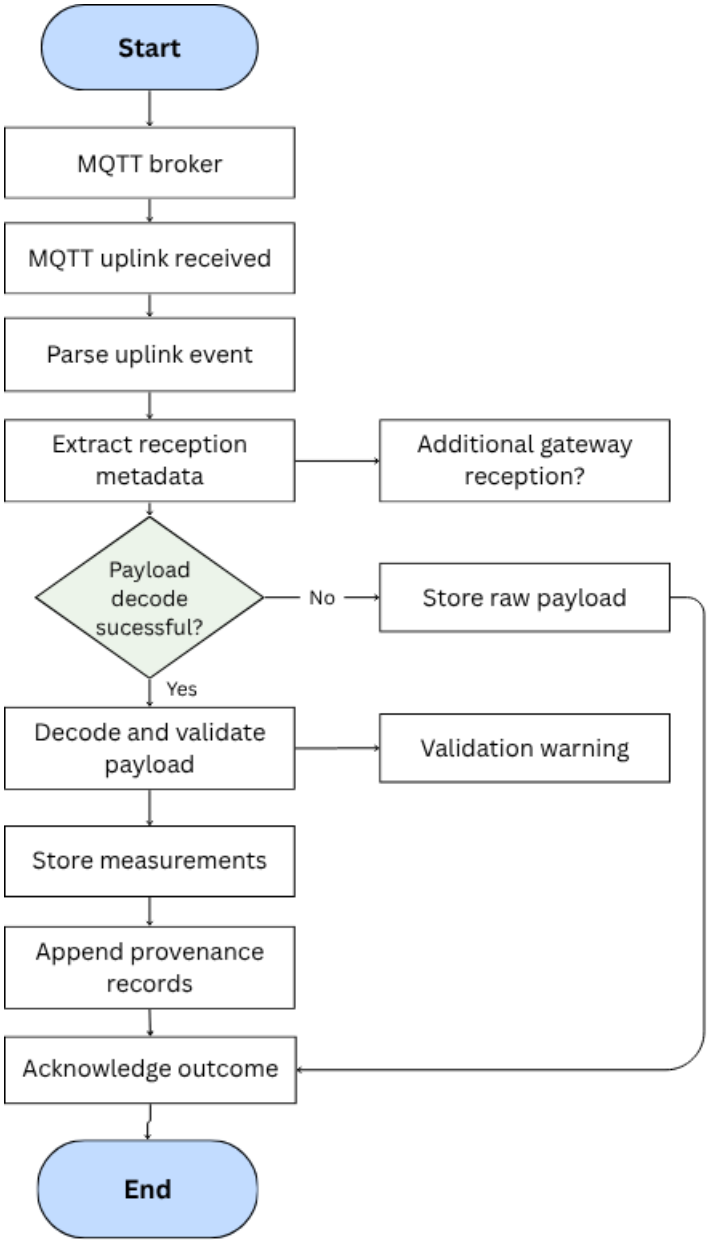


Figure 3.2 – MQTT uplink ingestion and decoding workflow

Figure 3.2 presents the processing path, from the reception of the MQTT uplink to the storage of the measurement and the creation of the provenance records. The collector handles the communication data and the decoded measurements as related, but separate, records. In this sense, stable identifiers keep the links between such records during the later processing and verification.

Each MQTT event provides the fields required for the ingestion. The implementation uses the device identifier, the reception timestamp and the encoded payload. From the reception context, the gateway identifier, the RSSI, the SNR, the spreading factor and the data rate are also used. Moreover, the implementation takes the frame counter, the FPort and the payload version, as well as the deduplication reference when available. Note that the collector does not store the complete MQTT message as one database record. Instead, the decoded sensor values are written to the measurements repository, and the individual gateway receptions to the radio metadata

repository. In addition, the processing identifiers, the event types and the storage results are written to the provenance repository. The original relation between such records is preserved by means of the uplink_key and the entity identifiers.

Before creating a new measurement entity, the collector checks whether the uplink has already been processed. The reception by several gateways may produce several reception entries for one logical uplink, and the network server may deliver the same MQTT event more than once. For this reason, the collector derives a persistent uplink_key (Formula 2.2). If the event contains the deduplicationId assigned by ChirpStack, such identifier is used as the key. Otherwise, the key is computed as a SHA-256 hash of the device identifier, the session epoch, the frame counter and the payload hash. The session epoch is incremented after a join event, or when the frame counter decreases, so the uplinks from different sessions with the same frame counter receive different keys. If the uplink_key already exists, the measurement entity has already been created. In such case, the new gateway receptions are added to the radio metadata repository and also linked to the existing uplink.

The payload decoding follows a predefined specification at byte level [167], [168]. In particular, the specification assigns each payload field to a measurement variable, and defines its byte position, its scaling factor and its engineering unit. The collector selects the decoding profile from the device configuration and the FPort, together with the payload version field. As a result, the changes in the payload layout can be handled by means of a new decoding profile, without any change in the earlier records. The stored payload version also identifies the decoding rules used for each measurement. However, the decoding cannot be repeated when the original payload is not available.

The collector validates the decoded values before their storage. Range checks identify the values outside the expected operating limits, whereas the status fields record the missing or unavailable measurements. Then, the validation results are stored with the corresponding entity and also included in the provenance record of the ingestion stage.

Table 3.5 summarizes the payload decoding specification used in the implementation, namely the encoded variables, the byte positions, the scaling factors and the engineering units, as well as the expected ranges and the quality control rules.

Table 3.5 - Payload decoding specification

Variable	Encoding and scaling	Unit	Typical validity range
1	2	3	4
$PM_{1.0}$	uint16 × 0.1	$\mu g/m^3$	0 - 1000
$PM_{2.5}$	uint16 × 0.1	$\mu g/m^3$	0 - 1000
PM_{10}	uint16 × 0.1	$\mu g/m^3$	0 - 1000
CO	uint16 × 0.1	ppm	0 - 1000
SO_2	uint16 × 0.01	ppb	0 - 2000
NO_2	uint16 × 0.01	ppb	0 - 2000

Continuation of Table 3.5

Variable	Encoding and scaling	Unit	Typical validity range
1	2	3	4
O_3	uint16 × 0.01	ppb	0 - 1000
temperature	int16 × 0.01	°C	-40 to +85
humidity	uint16 × 0.01	%RH	0 - 100
pressure	uint16 × 0.1	hPa	300 - 1100
Battery_v	uint16 × 0.001	V	2.8 - 4.2
seq_no	uint16	-	Ordering and gap analysis

When the payload decoding fails, the uplink is not discarded. Instead, the collector stores the raw payload for diagnosis, keeps the reception data in the radio repository and writes an INGEST provenance record, with the status FAIL and the reason for the failure. The attribution procedure (Section 3.7) reports such uplink as a processing failure (i.e., class P1).

The decoded measurements are validated before their storage. More specifically, the collector checks the required fields, the expected ranges and some basic plausibility conditions. The invalid values are not removed from the dataset, but stored together with quality indicators, which are also included in the INGEST provenance record. As a result, later queries and dataset exports may exclude such measurements, without deleting the original processing history. Moreover, the collector checks the sequence numbers and the LoRaWAN frame counters, when available, since gaps in such values indicate possible missing measurements.

Each measurement record includes the reporting interval, the temporal reference and the decoding profile, as well as the validation result. In this way, the record preserves the conditions under which the measurement entered the platform. Stable identifiers connect the measurement to its logical uplink, to its gateway receptions and to its provenance records.

The transformation of an MQTT event follows the provenance model defined in Section 2.4.2. Each logical uplink is represented by one uplink entity, with its stable uplink_key and the associated reception context. A successful decoding produces a measurement entity in the measurements repository. Figure C.2 shows some examples of the resulting database records.

The collector creates the provenance records for the RX, INGEST and STORE stages. The RX record describes the reception of the uplink. The INGEST record describes the parsing, decoding and validation, and its status field records the outcome (i.e., OK or FAIL). Last, the STORE record identifies the resulting database entity. Stable entity identifiers connect the input and output records of each stage. Then, each record is appended to the hash-linked chain of the device, and its hash is computed as described in Section 3.4.

The provenance repository stores the above records in sequential form, and the entity and activity identifiers preserve the relations required for the reconstruction of

the graph. As a result, the platform is able to trace a stored measurement back to its uplink and gateway receptions, or to follow an uplink forward to its decoded and stored records. Figure C.4 shows some examples of the stored provenance records.

Algorithm 3.1 presents the ingestion procedure of the collector, which covers the parsing of the message, the derivation of the uplink_key, the detection of duplicates, the payload decoding and the validation. Moreover, the procedure includes the creation of the entity, the updates of the repositories and the appending to the provenance chain. All the operations for one uplink run in one database transaction (Section 2.2). The measurement, radio and provenance tables are stored in the same MySQL 8.0 database and use the InnoDB storage engine. In this way, one COMMIT makes all the changes visible together, and one ROLLBACK removes all of them. The append operation reads the chain head of the device with a row lock, held until the end of the transaction. As a result, two uplinks of the same device cannot use the same predecessor hash.

The transaction has three possible outcomes. If the decoding and the validation succeed, the RX, INGEST (OK), measurement and STORE records are committed together. If the decoding fails, the RX record and an INGEST record with the status FAIL are committed (i.e., class P1). Last, if a database error occurs at any step inside the try block, the catch block rolls back the whole transaction, and no record of the uplink remains. In such case, the uplink can be found only in the event log of the network server (i.e., class P0).

The procedure maintains the links between the logical uplinks, the measurement entities and the gateway receptions, as well as the provenance records, along the whole ingestion pipeline. Such links provide the input required for the chain verification and the lineage reconstruction in the experimental evaluation.

Algorithm 3.1 - Provenance-aware ingestion procedure

```
    Input: mqtt_msg
    Output: SUCCESS(meas_eid), DUPLICATE(uplink_key), DECODE_FAIL, or STORAGE_FAIL
1   event ← parse_json(mqtt_msg)
2   uplink_key ← derive_uplink_key(event)    ▷ Formula (2.2)
3   BEGIN TRANSACTION    ▷ one InnoDB transaction covers all platform tables
4   try
5     if provenance.rx_exists(uplink_key) then    ▷ repeated delivery or late gateway reception
6       radio_rx.insert(uplink_key, event.rx_info_list)    ▷ radio metadata only, no new RX record
7       COMMIT
8       return DUPLICATE(uplink_key)
9     end if
10    radio_rx.insert(uplink_key, event.rx_info_list)
11    provenance.append(build_RX_record(event, uplink_key))    ▷ locks the chain head of the device
12    decode_result ← decode_payload(event.payload)
13    if decode_result.status = FAIL then
14      raw_payloads.insert(event.device_id, uplink_key, event.payload)
15      provenance.append(build_INGEST_record(uplink_key, FAIL, decode_result.reason))
16      COMMIT
17      return DECODE_FAIL    ▷ processing failure, class P1
18    end if
19    meas_eid ← generate_measurement_id(event.device_id)
20    meas_quality ← perform_validation_checks(decode_result.values)
21    provenance.append(build_INGEST_record(uplink_key, output_eid = meas_eid, status = OK))
22    measurements.insert(meas_eid, uplink_key, decode_result.values, meas_quality)
23    radio_rx.link(uplink_key, meas_eid)
24    provenance.append(build_STORE_record(meas_eid))
25    COMMIT
26    return SUCCESS(meas_eid)
```

```
27    catch DatabaseError
28      ROLLBACK
29      return STORAGE_FAIL    ▷ no record remains; visible only in the network-server log (class P0)
30    end try
```

The ingestion pipeline handles the duplicate gateway receptions, as well as the missing measurement intervals. Several gateways may receive the same logical uplink. In such case, the radio repository stores each reception record, whereas the measurement repository keeps only one measurement entity for the uplink. When such fields are available, the continuity of the frame counter and of the device sequence numbers is used to identify suspected gaps. The provenance records store the reception context, the decoding profile and the validation result, together with the quality status of each measurement. As a result, the duplicate receptions and the decoding failures can be distinguished from the delayed uplinks and the missing measurements during the analysis.

Use case 3.1 Handling missing data and duplicates in LoRaWAN monitoring. During long-term monitoring, a LoRaWAN deployment may produce both duplicate receptions and missing measurement intervals. Long-term LoRaWAN monitoring may produce duplicate receptions and missing measurement intervals. The ingestion pipeline is able to handle both conditions without deleting the available evidence. Duplicate receptions occur when several gateways receive the same uplink. ChirpStack may publish one event with several reception entries, or repeated events for the same transmission. The collector derives the uplink_key from the deduplicationId of the event or, when not available, from the device identifier, the session epoch, the frame counter and the payload hash (Formula 2.2).

The collector checks the uplink_key before creating a measurement entity. An existing RX record with the same key indicates that the logical uplink has already been processed, so no additional measurement entity is created. Each new gateway reception is stored in the radio metadata repository and linked to the existing uplink. As a result, the measurement time series contains one record for each logical uplink, whereas the radio repository keeps all the available reception evidence. The radio records identify all the gateways that received the uplink and preserve their link parameters, while the single RX provenance record links them to the chain through the uplink_key. Such records support the later examination of the gateway diversity, the RSSI, the SNR and the reception conditions, without introducing duplicate measurements.

Missing intervals may result from radio interference, coverage changes or device restarts, as well as from the loss of an uplink before it reaches the platform. The collector does not generate any replacement values during the ingestion. Instead, the collector checks the continuity of the LoRaWAN frame counters and the device sequence numbers, when such fields are available. A discontinuity is recorded as a suspected gap in the measurement stream. Later, the attribution procedure in Section 3.7 uses the frame counter gaps to decide at which stage each missing measurement was lost. The gap indicators are stored with the corresponding measurement entities, and they do not remove or replace the recorded values. In this sense, later queries

may keep all the measurements, or exclude some records according to the selected quality rules. Figure C.3 shows some examples of the stored radio metadata.

The decoding failures are handled separately from the missing uplinks. When a payload cannot be decoded, the collector keeps the raw payload and the reception record. The processing log records the failure, and an INGEST provenance record identifies the decoding result. In this way, an analyst is able to distinguish an uplink that was not received from a received uplink that failed during the decoding (i.e., class P1 in Section 2.4.5). The provenance chain connects each accepted measurement to its reception and processing records. The INGEST record identifies the decoding profile and the validation result, while the STORE record identifies the stored measurement entity. All together, the links provide the evidence required to examine the duplicates, the gaps and the decoding failures during the later analysis.

In summary, the MQTT ingestion pipeline connects the reception of LoRaWAN uplinks with the storage of measurements, radio metadata and provenance. The next section describes the database schema, together with the indices used for the repository queries and the graph reconstruction.

3.4 Provenance-aware storage and database schema

The platform stores the decoded measurements, the radio reception data and the provenance records in separate repositories. The measurement repository contains the validated environmental observations, whereas the radio repository contains the gateway receptions and the link quality parameters. The provenance repository, in turn, contains the append-only records, which are arranged in hash-linked chains per device. From the implementation point of view, the three repositories are separate tables of the same MySQL 8.0 database, and all of them rely on the transactional InnoDB storage engine.

Stable identifiers connect the records across the three repositories and there each measurement refers to its logical uplink, and each reception record refers to the same uplink identifier. Moreover, the provenance records contain the entity identifiers created during the RX, INGEST and STORE stages. In this way, the relation between the received packet, the decoded measurement and the stored database record is preserved.

The separation of the repositories is motivated by their different storage and query requirements. In particular, the measurement records are used for the analysis of time series, the radio records for the analysis of communication and coverage, and the provenance records for the chain verification and the lineage reconstruction. The queries across repositories use indexed identifiers, rather than duplicated event content. Figure 3.3 presents the storage architecture and the links between the repositories, while Figure 3.4 shows the implemented relational schema, together with the main identifier fields.

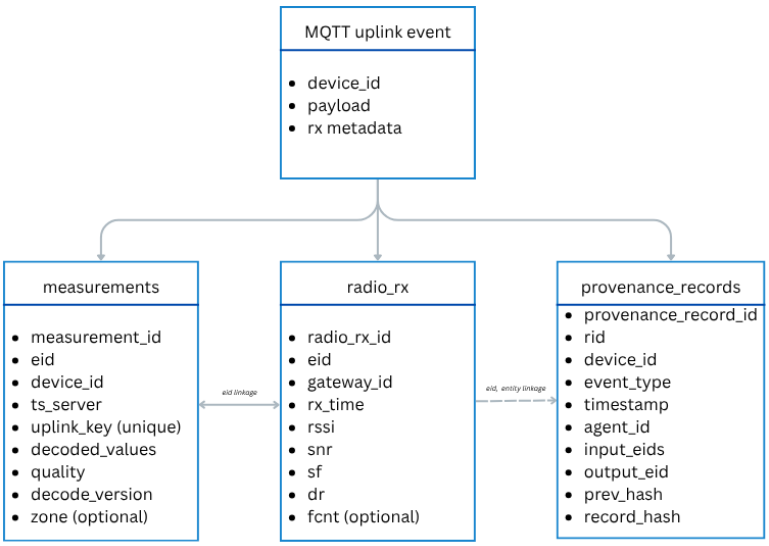


Figure 3.3 - Provenance-aware storage architecture and identifier linkage across repositories

Figure 3.3 shows the separate storage of the measurements, the radio data and the provenance records, where stable identifiers link the records created during the reception, ingestion, validation and storage stages.

The measurements repository stores the decoded environmental values, together with their validation results. The radio repository, in turn, stores the individual gateway receptions of each logical uplink. Each reception record includes the gateway identifier, the RSSI, the SNR, the spreading factor and the data rate, as well as the frame counter when available. The provenance repository stores the RX, INGEST and STORE records. In particular, the fields *input_eids*, *output_eid*, *prev_hash* and *record_hash* define the links between entities and the structure of the hash chain.

One logical uplink may have several gateway receptions, but only one measurement entity. The common uplink_key and the entity identifiers connect such records without duplicating the decoded measurement. Moreover, each device has its own append-only provenance chain for the integrity verification.

The structure of the repositories supports different types of queries. Most often, the measurement records are used for the analysis of environmental time series, while the radio records are used for the analysis of communication. The provenance records, in turn, are used for the chain verification and the reconstruction of the processing history.

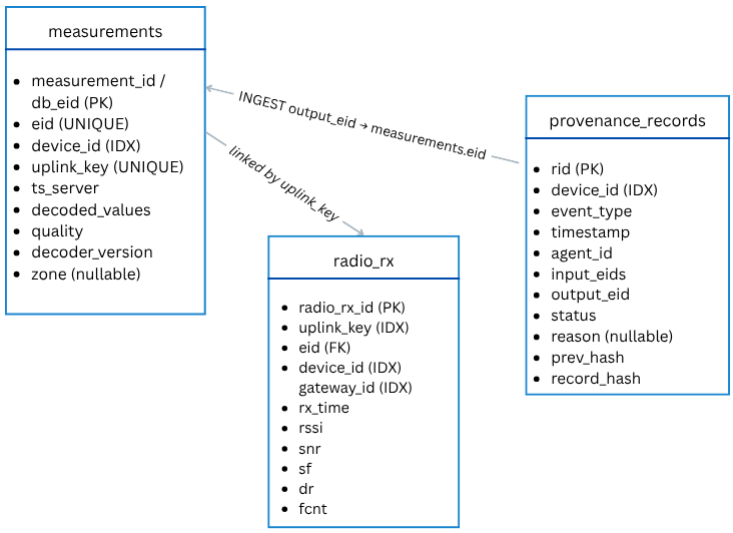


Figure 3.4 - ER-style relational schema of the provenance-aware platform databases

Figure 3.4 shows the relations between the measurement, radio and provenance tables. One measurement entity, identified by eid, may correspond to several records in *radio_rx*, when the same uplink is received by several gateways. The related provenance records describe the RX, INGEST and STORE stages of the same entity. Within the chain of each device, the *prev_hash* field links every provenance record to the previous one. In this way, the chain can be verified and the processing history can be reconstructed without any change in the stored measurement values.

Measurement entities and time-series storage. Each successfully decoded uplink produces one measurement entity, with a stable entity identifier (i.e., eid). The measurements repository stores the device identifier, the reception time at the server and the decoded sensor values, together with the assigned quality indicators. The reception time at the server is used as the main timestamp for the analysis, whereas the time on the device side is kept as auxiliary metadata, when available.

Moreover, the repository stores the decoding profile and the payload version used for each measurement. Such fields identify the rules applied during the interpretation of the payload, so the earlier records remain interpretable after any change in the payload format. However, the fields do not allow an independent re-decoding when the original payload is not available. Last, a unique constraint on *uplink_key* prevents the repeated MQTT events from creating duplicate measurement entities.

Radio metadata storage and multi-gateway reception evidence. A LoRaWAN uplink may be received by more than one gateway. For this reason, the radio repository stores a separate reception entry for each gateway observation. Each entry contains the gateway identifier, the reception timestamp, the RSSI, the SNR, the spreading factor and the data rate, while the references to the frame counter are added when available.

Note that the radio metadata are not included in the hash of the provenance records. The entries from additional gateways may arrive after the RX record has been committed and including them would require changing a stored record. Instead, such entries are linked to the provenance chain through the uplink_key and the eid.

The eid field links each reception entry to the corresponding measurement entity. As a result, one measurement may have several radio records without repeating the decoded sensor values. In this way, the evidence of multi-gateway reception is preserved, and the number of gateways, the strongest RSSI and the best SNR can be calculated later for each uplink. Moreover, the changes in such parameters can be examined together with the delayed or missing measurements.

Append-only provenance repository and record structure. The provenance repository stores the records in a separate chain for each device. Every record contains a record identifier (i.e., rid), the device identifier, an event type, a timestamp and an agent identifier, as well as the references to input and output entities and a status (i.e., OK or FAIL). The prev_hash field holds the hash of the previous record of the same device, whereas record_hash protects the content of the current record. During normal operation, the existing provenance records are not overwritten. A correction, a failure or a later processing result is represented by a new record. In this way, the original history remains available for verification and audit.

The chains per device preserve the order of events within each measurement stream, while the entity identifiers also support queries across devices and repositories. Such structure separates two functions of the provenance storage. On one side, the hash-linked chain supports the integrity verification. On the other side, the references to input and output entities support the lineage reconstruction.

Hash computation and key handling. The record hash is computed when the record is appended. The fields of the record relevant for integrity, including *prev_hash* and excluding *record_hash*, are serialised as compact JSON with sorted keys, and encoded in UTF-8. Then, the hash is computed as HMAC-SHA-256 over the previous hash and the resulting canonical form (Formula 2.24) [169]. The key is stored outside the database, in the configuration of the provenance service and of the verifier, so the access to the database alone is not enough to produce valid hashes. Moreover, a key identifier is stored in the metadata of the record, in order to support the rotation of keys. Note that the chains created during the 12-month deployment were computed with unkeyed SHA-256 (Formula 2.15).

Chain-head checkpoints. After each successful verification, the verifier stores a checkpoint for the device, namely the device identifier, the hash of the last verified record, the chain length and the time (Formula 2.22). The checkpoints are kept in the audit log, which is written only by the verification service, and are used to detect the truncation of the chain tail (Section 3.5).

Indices and query-oriented schema choices. The database schema includes indices for the time-series queries and the provenance tracing. The measurements repository is indexed by device_id, the analysis timestamp, eid and uplink_key. The radio repository is indexed by eid, uplink_key, the gateway identifier and the reception timestamp. Last, the provenance repository is indexed by device_id, rid, the event timestamp, input_eids and output_eid.

Such indices support two main directions of query. A backward query starts from a stored measurement, and retrieves the uplink, the reception records and the processing stages that produced it. A forward query, in turn, starts from an uplink or

an early-stage entity, and retrieves the measurements and storage records derived from it.

Graph-view reconstruction readiness. Graph-view reconstruction readiness. The platform stores the provenance records as ordered chains, rather than as a permanent graph. Nevertheless, each record contains explicit references to its input and output entities. In this way, the entities listed in input_eids are used by the activity identified by rid, and such activity generates the entity referenced by output_eid.

For a lineage request, the query service retrieves the relevant provenance records and builds the corresponding graph view. The backward traversal identifies the source and the processing history of a stored measurement, whereas the forward traversal identifies the entities derived from an earlier uplink or processing result. The graph view follows the provenance model defined in Chapter 2, and can be used in audit reports and dashboards, as well as in dataset exports.

In summary, the database schema keeps the measurement, radio and provenance records separate, but connected by means of stable identifiers. The measurements remain available for the time-series queries, the radio records preserve the reception evidence, and the provenance chains of each device support the integrity verification. The next section describes the verification and audit procedures applied to such chains.

3.5 Integrity verification and audit workflow

The platform verifies the provenance chains either on demand or according to a fixed schedule. The on-demand verification is used for lineage queries, dataset exports and report generation, whereas the periodic verification checks the newly added records during the long-term operation. In both cases, the same verification procedure is applied.

Each device has an independent provenance chain. The verifier reads the selected records in order and recomputes the hash of every record with the same keyed function used during the ingestion. Then, the verifier compares the result with the stored record_hash and checks the prev_hash reference. A mismatch indicates a modified record, a missing record or a wrong order of the records.

A valid segment of the chain receives the status integrity_ok, and the platform records the verification time and the verified range in the audit log. Such information can be included by the export and reporting procedures in the generated dataset or report.

A failed check does not change the stored provenance records. Instead, the verifier records the device identifier, the first failing rid and the failure type, as well as the affected time range. The related measurements can then be excluded from the workflows that require a verified provenance, while the original records remain available for a later inspection.

The data ingestion continues after a verification failure, and the new provenance records are still appended to the stream of the device. The platform starts a new verification segment after the detected break of the chain and keeps the failed

segment in the audit history. In this way, the data collection is not interrupted, and the earlier records are not rewritten.

The scope of the verification depends on the operation. A lineage query checks the segment of the chain related to the requested entity, while a dataset export checks the records within the selected time range. In the case of periodic verification, the procedure starts from the last verified chain head, and processes only the records added since that point. As a result, the complete historical chain is not recalculated at every scheduled check.

Algorithm 3.2 presents the verification procedure for one segment of the chain of a device. The algorithm returns an OK or FAIL status, and reports all the inconsistencies of hashes and links detected within the selected segment. The same failure contexts are also written to the audit log.

Algorithm 3.2 – Chain verification procedure (per device)

```
Step    Pseudocode
Input   device_id, records[] in per-device chain order, expected_prev_hash, key K
Output  status, invalid_hash_count, broken_link_count, failure_contexts[], verified_head
1       n ← LENGTH(records)
2       invalid_hash_count ← 0
3       broken_link_count ← 0
4       failure_contexts ← EMPTY_LIST
5       expected_prev ← expected_prev_hash
6       if n = 0 then
7          return (OK, 0, 0, failure_contexts, expected_prev)
8       end if
9       for i ← 0 to n − 1 do
10         fields ← INTEGRITY_FIELDS(records[i])   ▷ all fields except record_hash
11         fields.prev_hash ← expected_prev
12         canonical ← JSON(fields, sorted keys, compact separators)
13         recomputed_hash ← HMAC_SHA256(K, UTF8(expected_prev ‖ canonical))
14         if records[i].prev_hash ≠ expected_prev then
15            broken_link_count ← broken_link_count + 1
16            APPEND(failure_contexts, (broken_link, records[i].rid, device_id))
17         end if
18         if records[i].record_hash ≠ recomputed_hash then
19            invalid_hash_count ← invalid_hash_count + 1
20            APPEND(failure_contexts, (hash_mismatch, records[i].rid, device_id))
21         end if
22         expected_prev ← records[i].record_hash
23      end for
24      if invalid_hash_count > 0 OR broken_link_count > 0 then
25         return (FAIL, invalid_hash_count, broken_link_count, failure_contexts, NULL)
26      end if
27      return (OK, 0, 0, failure_contexts, expected_prev)
```

The canonical representation contains all the fields of the record relevant for integrity, except record_hash. The prev_hash field is set to the expected hash of the predecessor, the fields are serialised as compact JSON with sorted keys, and the result is encoded in UTF-8. Then, the keyed hash is computed over the expected hash

of the predecessor, followed by such canonical form. Note that, for the chains created during the deployment, SHA-256 is used instead of HMAC-SHA-256.

For the verification of a complete chain of a device, expected_prev_hash is initialised with GENESIS_HASH. For the verification of a selected segment, expected_prev_hash contains the verified hash of the record right before the segment. The algorithm checks separately the stored reference to the predecessor and the recomputed hash of the content. The verified chain head is returned only when no inconsistencies of hashes or links are detected. After a successful check, the algorithm returns the hash of the last verified record. The platform stores such value, together with the chain length, as the checkpoint of the device, and uses it as the starting point of the next incremental check.

However, Algorithm 3.2 alone is not able to detect the truncation of the chain tail, since the remaining records stay consistent. For this reason, before the chain check, the verifier compares the current chain with the latest checkpoint (Formula 2.23). More specifically, the chain must contain at least as many records as at the time of the checkpoint, and the record at the checkpointed position must have the checkpointed hash. If the chain is shorter, a truncation is reported. If the hash differs, a modification of the checkpointed part of the chain is reported.

Table 3.6 lists the verification modes used by the platform, namely the trigger, the scope of the verification and the audit output for lineage queries, dataset exports and report generation, as well as for the scheduled verification.

Table 3.6 - Verification modes and audit outputs

Verification mode	Trigger	Typical scope	Stored output
1	2	3	4
On-demand verification	Lineage query for a selected entity	Local chain segment related to the producing records	Verification status; audit event if verification fails
On-demand verification	Dataset export or report generation	All provenance records within the requested export window	Export verification status; export job log; audit event if verification fails
Periodic verification	Scheduled audit routine	Incremental segments since the last verified chain head per device	Updated verified chain head; per-device audit status; audit event if verification fails

Such modes, as shown in Table 3.6, allow the platform to find a compromise between the audit coverage and the computational overhead. The on-demand verification checks the records required for a lineage query, a dataset export or a report, whereas the periodic verification checks the records added after the last verified chain head. In this way, the platform verifies the recent provenance records, without processing the complete historical chain at every audit.

Appendix C provides some screenshots of the prototype, of the processing of MQTT uplinks and of the stored provenance records, as well as of the results of the chain verification.

3.6 Provenance queries and graph-view reconstruction

The platform supports backward and forward provenance queries, which correspond to the lineage sets $L^{-}(e)$ and $L^{+}(e)$ defined by Formula (2.7). A backward query starts from a stored measurement, a dashboard value or an exported result, and follows its processing history back to the source uplink. A forward query, in turn, starts from an uplink or a measurement entity, and identifies the records and outputs derived from it. The query layer retrieves the provenance records by entity identifier, while the indices on input_eids and output_eid support the traversal between the related entities and processing activities. Moreover, the query may also retrieve the verification status of the corresponding segment of the chain of the device. The communication context is obtained from the radio_rx repository. Since a measurement entity may refer to several gateway receptions through eid, the query is able to return the gateways and the link quality parameters associated with the same logical uplink.

The provenance records are stored as ordered chains per device, rather than as a permanent graph. For each query, the platform retrieves the relevant records and reconstructs a bounded graph view from the stored links between entities. In particular, the RX records add the reception context, the INGEST records describe the payload decoding and validation, and the STORE records identify the resulting database entities. The agent identifiers, in turn, associate such activities with the corresponding services of the platform.

The backward tracing identifies the device, the gateway receptions, the decoding profile and the validation result, as well as the storage event associated with a selected entity. The forward tracing identifies the measurements, the database records and the exported outputs derived from an earlier entity. The queries across devices use indexed fields like device_id, zone, timestamp and event type. In practice, a query may first return a summary for several measurements and then reconstruct the full lineage of a selected entity. In summary, the stored chains support the integrity verification, whereas the reconstructed graph views support the lineage analysis.

3.7 Missing-data attribution procedure

The attribution model of Section 2.4.5 is implemented as a separate analysis tool in Python. The tool does not change the ingestion pipeline, and reads only the data that the platform already keeps, namely the frames of each device from the event log of the network server and the RX, INGEST and STORE records from the provenance repository. For each device, the tool also needs the nominal reporting interval (i.e., 10 minutes in the deployment) and the observation period.

The tool works in three steps. Initially, every frame is mapped to an expected slot, by means of the tracked schedule phase of Formula (2.25). Then, each slot that

contains a frame is labelled as STORED, P1 or P0, according to the provenance records of such frame (Formula 2.26). Later, the empty slots between consecutive frames are labelled by comparing the number of empty slots with the gap of the frame counter (Formulas 2.27-2.29). The slots before the first frame and after the last frame remain undetermined (i.e., U3). Algorithm 3.3 presents the procedure.

Algorithm 3.3 - Missing-data attribution procedure (per device)

Step	Pseudocode
Input	Δt, K expected slots, frames F = {(t_j, f_j)} from the network-server event log, provenance records of the device, α
Output	label[0..K−1], number of slots in each class, coverage Cov
1	label[k] ← U3 for all k ▷ *boundary slots stay U3*
2	φ ← INITIAL_PHASE(F)
3	**for each** frame (t_j, f_j) in F **do**
4	k_j ← round((t_j − φ) / Δt); φ ← φ + α (t_j − φ − k_j Δt)
5	**if** RX, INGEST(OK) and STORE records exist **then** label[k_j] ← STORED
6	**else if** RX and INGEST(FAIL) records exist **then** label[k_j] ← P1
7	**else** label[k_j] ← P0 ▷ *frame in the log, no RX record*
8	**end for**
9	**for each** pair of consecutive frames with known FCnt (k_a, f_a), (k_b, f_b) **do**
10	S ← k_b − k_a − 1; m ← number of frames without FCnt between them
11	**if** S = 0 **then continue**
12	**if** f_b > f_a **then**
13	L ← f_b − f_a − 1 − m
14	**if** L ≥ S **then** empty slots ← U1
15	**else if** L = 0 **then** empty slots ← U2
16	**else** empty slots ← MIXED (L × U1, S − L × U2)
17	**else if** m > 0 **then** empty slots ← U3
18	**else** ▷ *FCnt reset after a re-join*
19	n_U1 ← min(f_b, S); last n_U1 empty slots ← U1; other empty slots ← U3
20	**end if**
21	**end for**
22	Cov ← (n_P1 + n_P0 + n_U1 + n_U2) / (K − n_STORED)
23	**return** label, class counts, Cov

The output contains one label for each expected slot, the number of slots in each class and the attribution coverage (Formula 2.32). The mixed gaps are reported together with their numbers of U1 and U2 slots. Such labels can be stored with the measurement series, or exported with a dataset. In this way, a user is able to see not only which measurements are missing, but also where they were lost.

Since the event log of the network server was not retained during the deployment, the tool is not able to separate the losses at platform intake (i.e., P0) from the upstream losses (i.e., U1) in the field data. For this reason, the accuracy of the tool is evaluated on simulated traces with a known ground truth, generated by means of the ns-3 LoRaWAN module.

Conclusions for Chapter 3.

In this chapter we presented the implementation of the framework as a prototype on the platform side, on top of a private LoRaWAN network with ChirpStack, which was operated for 12 months. Provenance is captured from the standard MQTT integration, and no changes were made to the end devices, to the payload format or to

the gateways. The collector processes each uplink in one database transaction, and creates the RX, INGEST and STORE records. The duplicate receptions are merged through the uplink_key, whereas the decoding failures are kept as INGEST records with the status FAIL. Moreover, a failed transaction leaves no partial records.

The measurements, the radio metadata and the provenance records are stored in separate repositories, linked by means of stable identifiers. The provenance records form hash chains per device. In its current version, the prototype uses HMAC-SHA-256 with a key kept outside the database and stores the checkpoints of the chain heads in the audit log. The verifier checks the segments of the chain both on demand and periodically. As a result, modified, missing and reordered records are detected, as well as the truncation of the chain tail, by means of the comparison with the checkpoint. The backward and forward provenance queries reconstruct a graph view from the stored records. Last, the attribution procedure assigns each missing measurement to a stage of the pipeline or classifies it as undetermined.

4 EXPERIMENTAL EVALUATION OF THE PROPOSED FRAMEWORK

4.1 Experimental setup, dataset, and evaluation metrics

The evaluation uses the data collected during a 12-month LoRaWAN deployment for air quality monitoring. Three sensing nodes transmitted measurements at fixed intervals of 10 minutes through two outdoor gateways. The local platform processed the received uplinks, and stored the measurement, radio and provenance records.

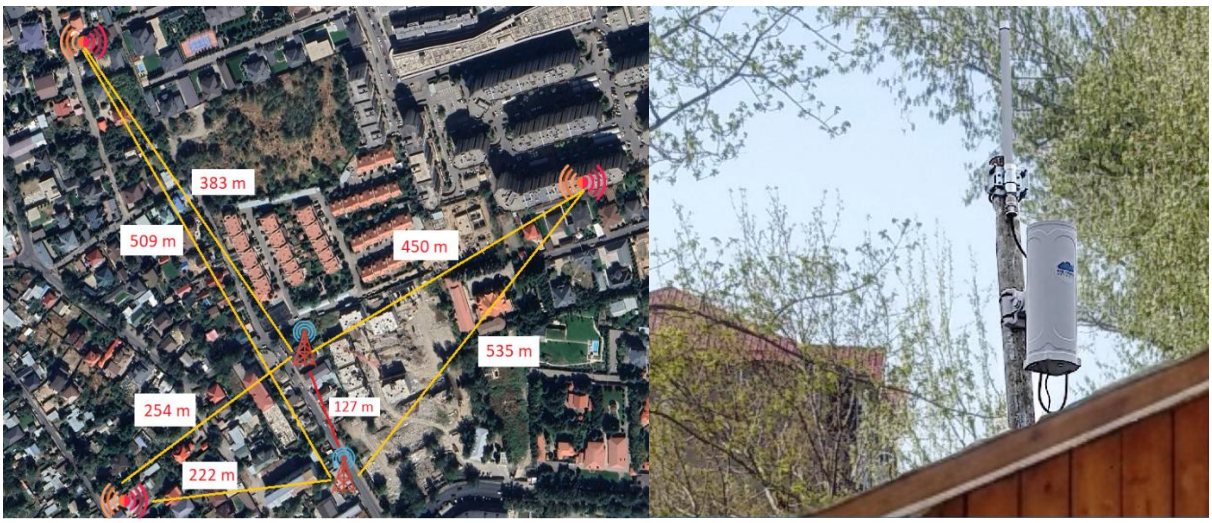
The field dataset contains regular observations, as well as missed intervals, delayed uplinks and receptions through several gateways. Such dataset forms the basis for the evaluation of the system under normal, long-term operating conditions. In parallel, separate controlled experiments were conducted in order to measure the ingestion performance, and to verify the integrity of the provenance under specified modifications of the records.

The evaluation covers six areas, namely data completeness, attribution of missing data, LoRaWAN communication quality, ingestion performance, storage cost and provenance verification. The completeness metrics describe the number of expected and received measurements, together with the duration of the missing intervals, whereas the attribution assigns each missing expected measurement to the stage of the pipeline where it was lost. The connectivity metrics include the RSSI, the SNR, the spreading factor and the data rate, as well as the number of gateway receptions per uplink.

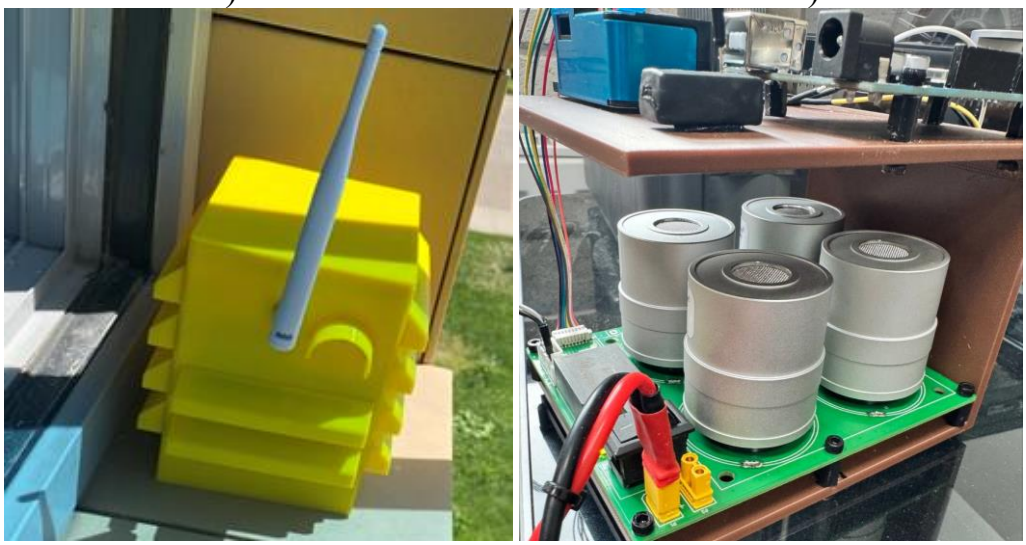
The performance of the platform is assessed in terms of ingestion time and processing throughput. The storage cost corresponds to the database space occupied by the measurements, the radio metadata and the provenance records. The integrity evaluation measures the time needed for the verification, and for the detection of modifications, deletions, insertions and reorderings in the provenance chains. Last, the traceability is assessed by means of queries along the backward and forward provenance paths of the stored entities.

Note that the attribution model described in Section 2.4.5 was not applied to the field data. The event log of the network server, which is necessary to separate the losses at platform intake from the upstream losses (Section 2.2), was not stored during the deployment. Moreover, the field data do not contain reliable information about the cause of each uplink loss. For this reason, the attribution model was evaluated by means of simulated deployments, in which the cause of each loss is known (Section 4.6).

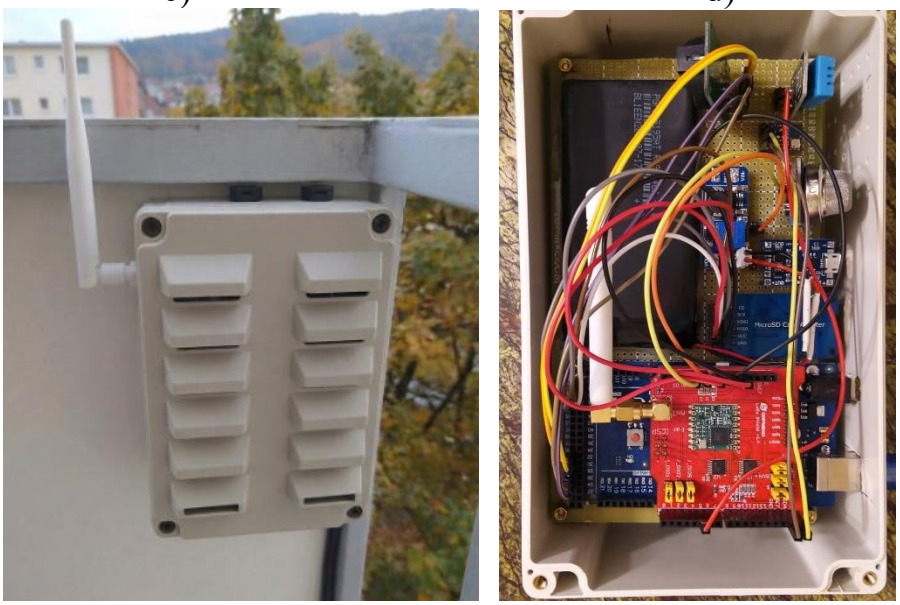
During the monitoring, the deployment produced 403,900 provenance records. The field data were stored in MySQL 8.0 on an edge server. The offline validation experiments were conducted with the exported provenance dataset and the Python implementation described later in this chapter. Figure 4.1 shows the deployment, namely a sensor node, a gateway with its antenna and the locations of the sensor nodes and gateways. The map provides the spatial context for the analysis of data completeness, signal quality and reception through several gateways.



a) b)



c) d)



e) f)

Figure 4.1 - Experimental LoRaWAN air-quality monitoring deployment: a) locations of the sensing nodes and gateways; b) gateway with antenna; c) sensing node, side view; d) inside the sensor node; e) sensing node at the second location; f) inside the sensor node at the second location

The dataset covers the period from 1 August 2024 to 31 July 2025, namely 365 days of monitoring. Three sensing nodes operated at two deployment locations, and each node transmitted one uplink every 10 minutes. Under continuous operation, such schedule corresponds to 144 uplinks per device and per day, and to 157,680 uplinks for the complete deployment.

The collected dataset reflects the normal operation of a LoRaWAN monitoring system. Some uplinks were delayed or missing, due to temporary link losses, device downtime and maintenance, as well as interruptions at gateway or network server level. In some cases, the same transmission was also received by more than one gateway. The repeated MQTT deliveries were included in the ingestion statistics, while the payload decoding failures were kept as part of the operational dataset.

In the deduplicated dataset, one record represents one logical uplink, identified by the platform through a stable uplink_key. A separate radio record represents each gateway reception. As a result, the number of radio records may exceed the number of logical uplinks, when several gateways receive the same transmission. One RX provenance record represents the reception of one deduplicated logical uplink on the platform side, whereas the individual gateway observations are stored separately in the radio_rx repository and linked to the same uplink_key. Each successfully processed uplink produces one measurement entity, together with the corresponding RX, INGEST and STORE provenance records created by the ingestion service. Stable identifiers link the measurement entity to its radio receptions, as well as to its processing history.

The environmental measurements include PM1.0, PM2.5 and PM10, the concentrations of gaseous pollutants, the air temperature, the relative humidity and also the barometric pressure. When available, the information about the device status was also kept (e.g., battery voltage, charging status, uptime and reset counters).

The evaluation examines the data completeness, the reception quality, the ingestion performance, the storage growth, the integrity verification and the lineage reconstruction. Table 4.1 lists the main characteristics of the deployment and of the dataset used in the evaluation.

Table 4.1 - Deployment and dataset summary

Item	Value	Notes
1	2	3
Monitoring period	01.08.2024 - 31.07.2025	Total duration: 12 months (365 days)
Number of sensing nodes	3	Unique device identities (DevEUI)
Number of zones (locations)	2	Coarse-grained zone labels used for analysis
Nominal sampling interval	10 min	144 expected uplinks per device per day
Total expected uplinks	157,680	3 devices × 144 uplinks/day × 365 days

Continuation of Table 4.1

Item	Value	Notes
1	2	3
Total uplink events received (MQTT)	146,200	Includes repeated MQTT events and decode-fail events
Unique uplinks after idempotent deduplication	135,000	Rows in uplinks.csv; one row per logical uplink
Total gateway reception records (radio_rx rows)	170,800	Reception-level rows in radio_rx
Successfully decoded measurement entities	133,900	Stored measurement entities with eid
Decode failures	1,100	Raw payloads preserved for audit and debug
Repeated-event rate	7.7%	(146200 − 135000) / 146200
Repeated-event rate in uplinks.csv	0%	No repeated uplink_key values after deduplication
Observed-uplink missingness	14.4%	Based on expected schedule vs deduplicated logical uplinks
ADR enabled	YES	DR/SF transitions evaluated in Section 4.3
LoRaWAN class	Class A	Typical for low-power monitoring
Confirmed uplinks used	No, except rare cases	Approximately 0-2% of uplinks

Before the deduplication, the collector processed 146,200 deliveries of MQTT uplink events. Such value was obtained by counting the event deliveries at the ingestion input and does not represent the number of complete raw MQTT messages kept in the database. After an idempotent deduplication by uplink_key, 135,000 logical uplinks remained in uplinks.csv. The difference corresponds to a rate of repeated events of 7.7%, and no repeated uplink_key values remain in the deduplicated dataset.

The uplinks.csv dataset is used for the analysis of completeness, missing data, decoding failures and ADR, where each row represents one logical uplink. The analysis at reception level, in turn, uses radio_rx.csv, which contains 170,800 gateway records. In this case, several rows may refer to the same logical uplink, when more than one gateway receives the transmission.

The radio records provide the gateway identifier, RSSI, SNR, spreading factor and data rate, as well as information about reception diversity. Such records are not counted as duplicate uplinks and are not used to calculate completeness.

Moreover, the platform generated separate datasets for processing performance, storage growth and provenance evaluation. Each dataset represents a different stage of the data lifecycle and is used only for the corresponding analysis.

Table 4.2 lists the evaluation datasets, together with their role in the analysis.

Table 4.2 - Evaluation datasets and analytical role

Dataset	Analytical level	Main fields	Role in evaluation
1	2	3	4
uplinks.csv	Deduplicated uplink level	uplink_key, device_id, zone, ts, sf, dr, decode_status, eid	Completeness, missingness, decode failures, ADR/SF/DR analysis
radio_rx.csv	Reception level	uplink_key, gateway_id, rssi, snr, sf, dr, rx_n	RSSI/SNR distribution, gateway reception diversity
latency.csv	Pipeline-stage level	stage, p50_ms, p95_ms, n	Ingestion latency and processing-stage cost
db_growth.csv	Storage level	ts, storage-size fields	Database growth and storage contribution
verification.csv	Verification-run level	records_validated, runtime_ms	Provenance verification runtime
lineage_edges.csv	Provenance graph level	source, target, relation	Graph-view lineage reconstruction

Separate datasets were used for the analysis at uplink level and at reception level. Completeness and ADR behavior were evaluated on the deduplicated uplink dataset, whereas RSSI, SNR and reception diversity were evaluated on the radio reception dataset. In this way, the receptions by several gateways did not affect the completeness results.

Data completeness describes the proportion of expected observations that are present in the collected dataset. In IoT studies, such metric is commonly calculated from the number of obtained and expected values within a defined time interval [170], [146], [57]. In the present study, the expected number of observations follows the configured reporting schedule of 10 minutes.

Completeness is commonly treated as a dimension of data quality, which reflects the presence of the data required for a defined application [56]. In this study, the uplink completeness was calculated as the ratio between the number of observed logical uplinks and the number of uplinks expected from the configured reporting schedule:

$$C_{w,z} = \frac{U_{w,z}^{obs}}{U_{w,z}^{exp}} \tag{4.1}$$

$$U_{w,z}^{exp} = N_{w,z}^{active} \times 144 \times D_w \tag{4.2}$$

The effective missingness rate is calculated as:

$$M_{w,z} = 1 - C_{w,z} \tag{4.3}$$

where $C_{w,z}$ is the completeness rate for week $w$and zone z; $U_{w,z}^{obs}$ is the number of observed deduplicated logical uplinks; $U_{w,z}^{exp}$ is the expected number of uplinks; $N_{w,z}^{active}$ is the number of active devices; D_w is the number of days in week w. The constant 144 corresponds to the expected number of uplinks per device per day under a 10-minute nominal reporting interval.

Decode-failure rate is calculated as the number of uplinks with unsuccessful payload decoding divided by the total number of received unique uplinks. Reception diversity is defined as the number of gateway reception records linked to the same uplink_key.

The defined metrics form the basis for the experimental results presented in the following sections. Completeness and missingness use deduplicated uplink records because one logical uplink may have several gateway receptions. RSSI and SNR use reception-level records because each gateway reports its own link-quality values. Spreading factor and data rate are examined at the uplink level. Platform performance is evaluated from the ingestion and storage datasets, whereas provenance evaluation uses the verification and lineage datasets. Each metric is therefore calculated from the dataset that represents the corresponding stage of the data lifecycle.

4.2 Data completeness and missingness analysis

The analysis covers the availability of the LoRaWAN monitoring data over the whole deployment period. Completeness is calculated from the deduplicated uplink dataset. The records at reception level are not used, since several gateways may receive the same logical uplink. The expected number of uplinks follows the reporting schedule of 10 minutes, so each active device was expected to transmit 144 uplinks per day. For three sensing nodes and 365 days, the expected total was 157,680 uplinks.

The deduplicated dataset contains 135,000 logical uplinks. Among them, 133,900 were successfully decoded and stored as measurement entities. The observed-uplink completeness was 85.6% (i.e., 135,000/157,680), and also the effective completeness was 84.9% (i.e., 133,900/157,680).

The observed-uplink completeness represents the proportion of expected transmissions received by the platform, whereas the effective completeness includes only the measurements that were successfully decoded and stored. The difference between both values corresponds to the received uplinks that failed during the payload decoding. Over the whole monitoring period, the effective missingness was 15.1%.

In terms of the model of Section 2.4.3, observed-uplink completeness corresponds to N_{rx}/M. Because the network-server event log was not retained, N_{uplink} is approximated by N_{rx}, i.e. $P_{intake} = 1$ is assumed, and platform intake losses, if any, are counted as upstream losses. The conditional probability of successful storage was $P_{store|rx} = 133{,}900/135{,}000 = 99.2\%$, so $C_{effective} = 0.856 \times 0.992 = 0.849$, in agreement with Formula 2.14.

Figure 4.2 presents the weekly completeness. The weekly aggregation reduces the short-term variation and shows the longer changes in data availability. Completeness is calculated from unique uplinks, while the rate of decoding failures is calculated from the deduplicated uplinks that could not be decoded during the same week.

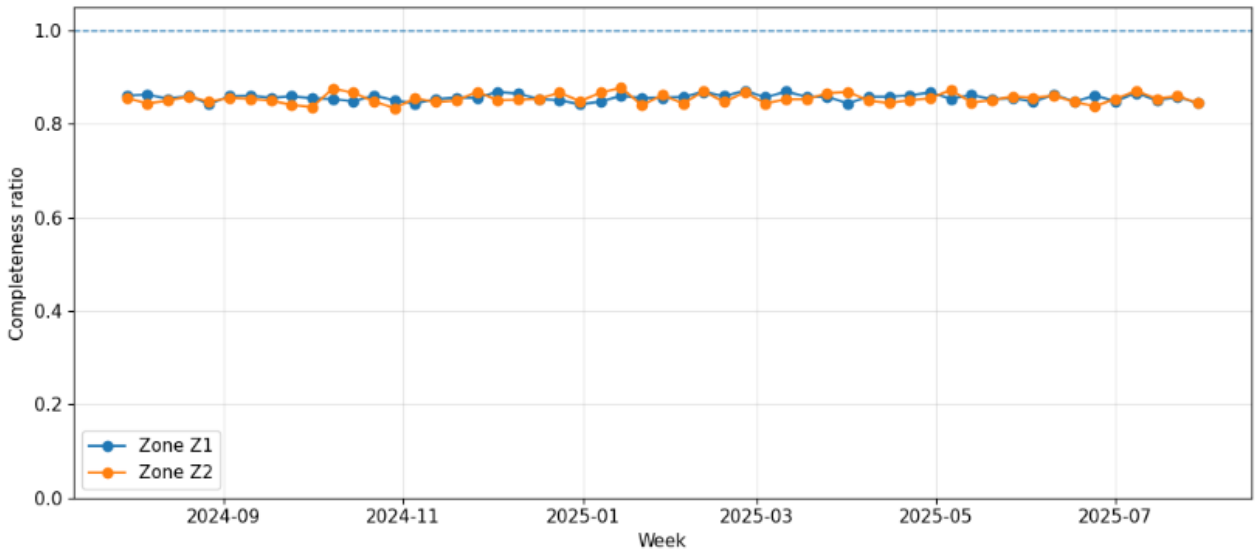


Figure 4.2 - Weekly data completeness by zone during the monitoring period from August 2024 to July 2025

Figure 4.2 plots the weekly completeness for the two monitoring zones over the whole observation period. The general trend shows that most of the weekly values ranged from 0.84 to 0.87. In particular, both zones followed similar temporal patterns, and none of the locations accounted for most of the missing data. The distance to the ideal value of 1.0 is a consequence of missing uplinks and temporary operational interruptions.

The ingestion records separate three conditions of data availability. Some expected uplinks were not received by the platform, some received uplinks failed during the payload decoding, and other uplinks were decoded and stored with quality flags assigned to individual variables. Such cases were analysed separately, since they represent different stages of data loss or degradation.

The deployment data allow classifying the outcomes of the expected uplinks into three categories, namely not received by the platform (22,680 uplinks, 14.4%), received but not decoded (1,100 uplinks, 0.7%, class P1), and decoded and stored (133,900 uplinks). For each received uplink, the frame counters and the reception timestamps were stored, and the location of the gaps was noted. However, a more precise attribution of the missing uplinks was not performed on the field data. Without the event log of the network server, the losses at platform intake cannot be distinguished from the upstream losses, and the field data do not provide any ground truth against which to verify the attribution. For this reason, the attribution model is evaluated in Section 4.6 by means of simulated deployments, in which the cause of each missing uplink is known.

In summary, an effective completeness of 84.9% was achieved over the 12-month deployment. The remaining 15.1% consisted of missing uplinks and of received messages that did not result in the storage of a measurement entity. The available reception, decoding and quality records were used in the subsequent analysis of the origin of such gaps.

4.3 LoRaWAN link quality, ADR behavior, and reception diversity

The link quality metadata provide the communication context for the missing uplinks and the receptions by several gateways. RSSI and SNR are analysed on the radio_rx dataset, since each gateway reports its own reception values. Spreading factor and data rate, in turn, are analysed on the deduplicated uplink dataset, since they describe the transmission parameters of each logical uplink.

The reception diversity represents the number of gateway receptions associated with one logical uplink, and is calculated as:

$$RD(u_i) = | \{r_{ij} : r_{ij}.uplink_key = u_i\} | \quad (4.4)$$

The average reception diversity over the dataset was calculated as:

$$\bar{RD} = \frac{N_{radio}}{N_{rx}} \quad (4.5)$$

where $RD(u_i)$ is the reception diversity of logical uplink u_i; r_{ij} is a gateway reception record associated with u_i; N_{radio} is the total number of gateway-level reception records; and N_{rx} is the total number of deduplicated logical uplinks. A value of $RD(u_i) = 1$ indicates single-gateway reception, while $RD(u_i) > 1$ indicates multi-gateway reception.

RSSI and SNR were used as operational indicators of the radio link conditions. The values were taken directly from the records of the LoRaWAN gateways, and were not treated as calibrated physical measurements.

Table 4.3 lists the RSSI and SNR statistics for the monitoring period. The calculations use radio_rx.csv, where each row represents one gateway reception. Such values provide the numerical basis for the distributions presented in Figure 4.3.

Table 4.3 - Reception-level RSSI and SNR summary statistics

№	Metric	N	Median	IQR	Minimum	Maximum
1	2	3	4	5	6	7
0	RSSI (dBm)	170800	-102.00	11.0	-119.0	-38.0
1	SNR (dB)	127569	-0.75	8.0	-24.0	14.0

The median RSSI was -102 dBm, with an interquartile range of 11 dB, as shown in Table 4.3. As a result, most of the receptions occurred under weak to moderate signal conditions. The median SNR was -0.75 dB, with an interquartile range of 8 dB. Negative SNR values were common, while the minimum and maximum values were

-24 dB and 14 dB, respectively. The presence of negative SNR values is explained by the capacity of LoRaWAN to receive packets below the noise floor.

The dataset contained 170,800 valid RSSI records and 127,569 valid SNR records, so the SNR was not available for 43,231 gateway receptions. Each indicator was analysed on its available values only.

The recorded distributions cover a wide range of radio conditions. Such results provide the basis for the analysis of reception diversity, ADR behaviour and data completeness.

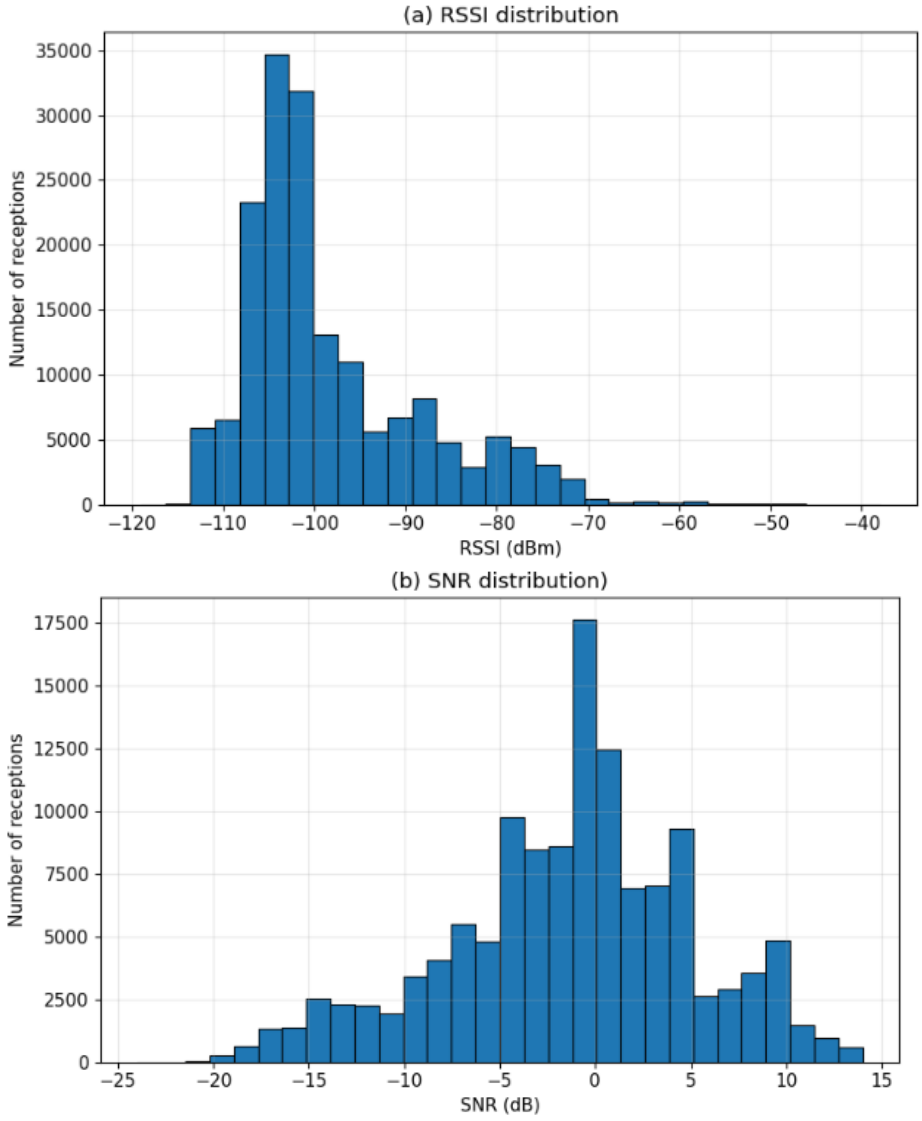


Figure 4.3 - Reception-level link-quality distributions: a) RSSI distribution; b) SNR distribution.

Figure 4.3 presents the distributions of RSSI and SNR at gateway level, recorded during the monitoring period. Most of the RSSI values lie between approximately -110 dBm and -95 dBm, and also the peak of the distribution is found from -105 dBm to -100 dBm. Only few receptions show stronger signal levels, with values reaching approximately -70 dBm. A smaller number of receptions appear at

stronger signal levels, and the distribution extends up to approximately -70 dBm. Such shape indicates that most uplinks reached the gateways under weak or moderate signal conditions, whereas stronger receptions occurred less frequently.

The SNR distribution covers a similarly broad range. Most of the observations fall near 0 dB, with a large proportion between approximately -5 dB and +5 dB. Negative values remain common, and extend to about -20 dB. As mentioned above, LoRaWAN is able to receive packets when the signal level falls below the noise floor, which explains the presence of such low SNR values. The positive values form a smaller tail, and represent receptions under clearer radio conditions.

RSSI and SNR describe different aspects of the received signal, RSSI represents the total received signal power reported by the gateway, whereas SNR represents the relation between the received signal and the surrounding noise level. In this sense, a reception may have a low RSSI value and still be successfully decoded, as long as the corresponding SNR remains within the operating range of the selected LoRa spreading factor. For this reason, both indicators were examined together, in order to describe the radio conditions present in the field dataset. The recorded distributions show that the deployment did not operate under uniform short-range conditions. Instead, the dataset includes strong links and weak links, as well as receptions close to the practical sensitivity limits of the gateways. Buildings, node placement, local interference, antenna orientation and changing environmental conditions may contribute to such variation. The analysis does not isolate the effect of each factor, and the recorded values only provide the communication context for the received uplinks.

The radio metadata repository preserves the RSSI and SNR of each gateway reception. Several gateways may report different values for the same logical uplink, since each reception follows a different propagation path. In this way, the gateway observations can be compared, the strongest reception can be identified, and the reception diversity can be examined for individual transmissions. Moreover, the same evidence supports the interpretation of periods with lower completeness or irregular uplink delivery.

Spreading factor and data rate were examined after the analysis at reception level. RSSI and SNR belong to individual gateway receptions, whereas SF and DR describe the transmission configuration of the logical uplink. For this reason, the analysis used the deduplicated uplink dataset for such parameters, and one logical uplink contributed one SF and DR observation, even when several gateways received the same transmission.

Figure 4.4 presents the SF and DR distributions for the three sensing nodes. The results show the transmission settings used during the deployment, and indicate whether the nodes remained at fixed configurations or moved between several data rates. The differences between devices may reflect their location, their link conditions and the ADR decisions made by the network server.

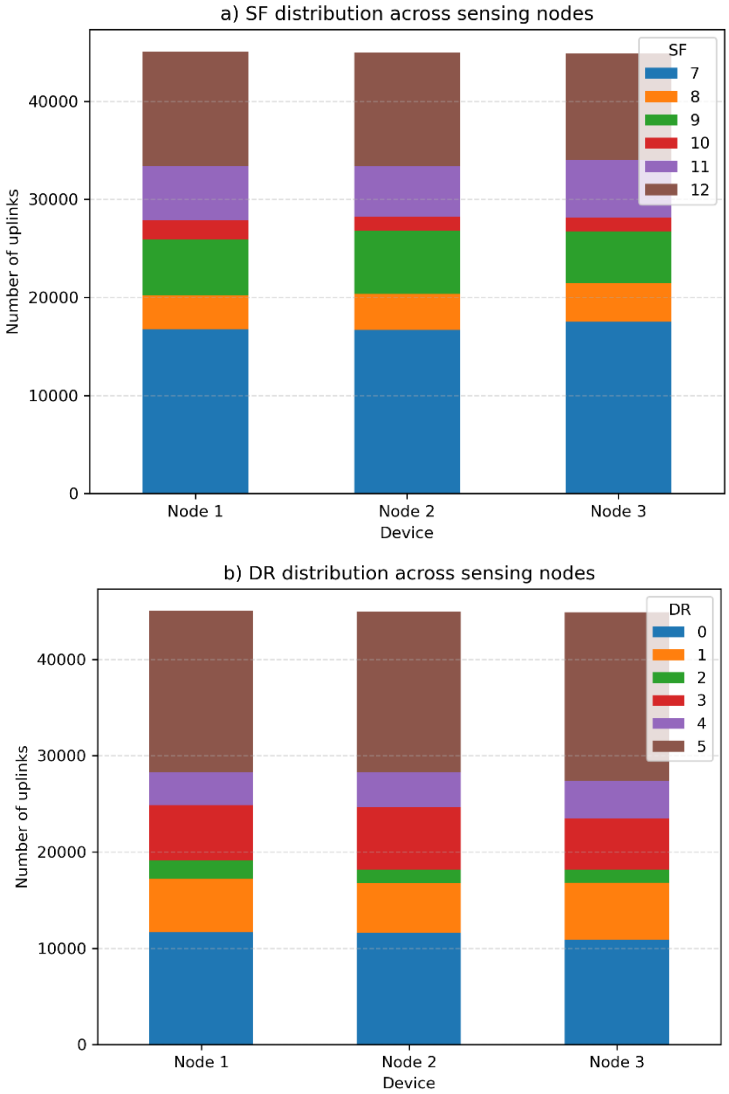


Figure 4.4 - Distribution of LoRaWAN transmission parameters across devices: a) SF distribution; b) DR distribution.

Figure 4.4 shows the distribution of the LoRaWAN transmission parameters for the three sensing nodes. All the nodes used several spreading factors during the monitoring period. SF7 accounts for a large share of the uplinks, but higher values like SF11 and SF12 also appear in the dataset. In this sense, the presence of higher spreading factors indicates that some transmissions used more robust settings, at the expense of a longer airtime.

The three nodes show broadly similar SF profiles. Node 3 has a slightly larger share of lower spreading factors, whereas all the nodes keep a visible proportion of SF12 transmissions. Such distributions reflect mixed communication conditions across the deployment, and none of the nodes operated under a single stable link profile during the whole monitoring period. Similarly, the DR distributions show broadly similar profiles across the three nodes. Node 1 contains comparable proportions of the main DR categories, with DR5 representing the largest share. Node 2 has a slightly larger share of DR3 transmissions, whereas Node 3 has slightly higher contributions from DR1, DR4 and DR5. As a result, the transmission settings varied across devices, as well as over time.

For this reason, the parameters related to ADR were stored for every logical uplink. A single SF or DR value for the whole deployment would not describe the transmission history of the three nodes. Instead, the records at uplink level preserve the actual configuration used for each transmission, and allow a later comparison between devices, zones and time periods. Moreover, SF and DR complement the RSSI and SNR records at gateway level. RSSI and SNR describe the conditions observed at reception, whereas SF and DR describe the transmission settings selected for the uplink. All together, such values provide a more complete account of the communication conditions associated with each measurement.

The distributions in Figure 4.4 describe the overall use of SF and DR during the deployment. However, they do not show whether the selected settings changed gradually, or only during specific periods. For this purpose, the weekly median SF was calculated for each sensing node, since the weekly aggregation reduces the short-term variation and reveals the longer changes in the transmission configuration.

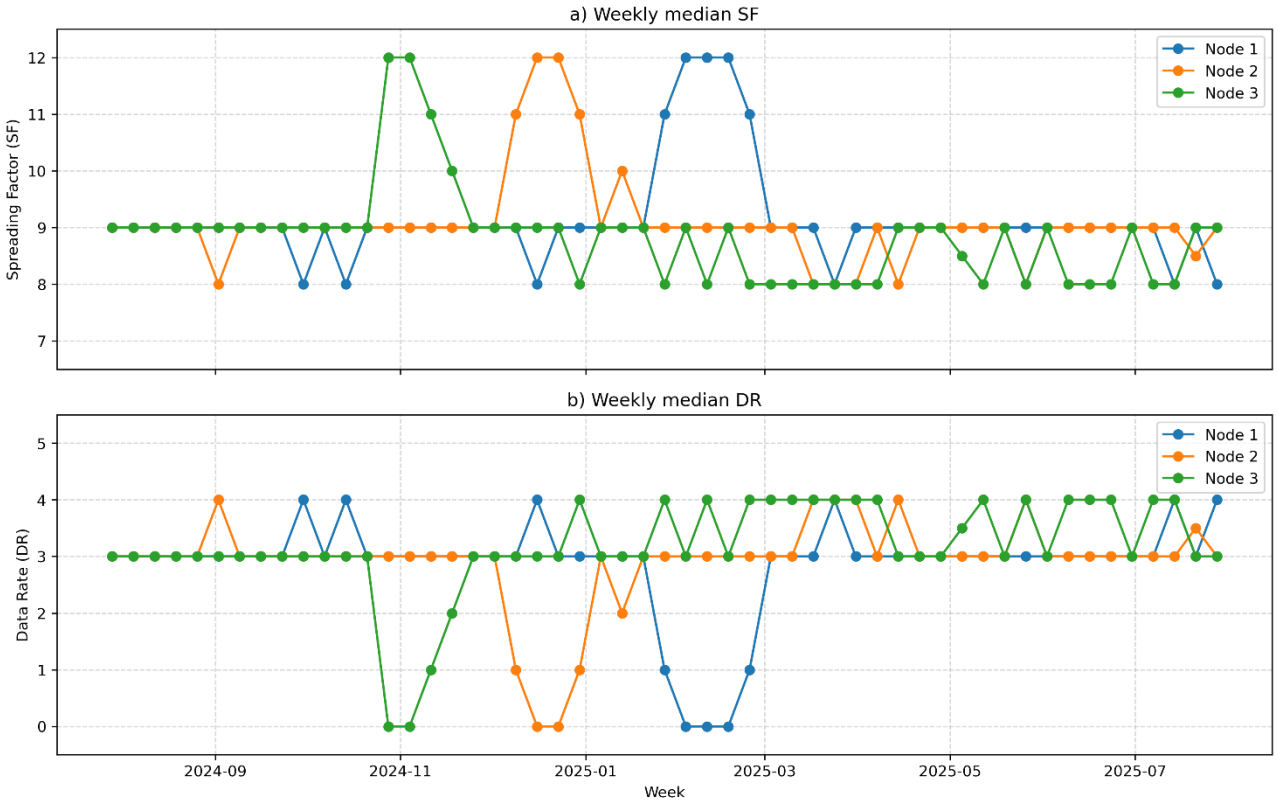


Figure 4.5 - Temporal ADR behavior by sensing node: a) weekly median SF; b) weekly median DR

Figure 4.5 shows how the transmission parameters related to ADR changed over the monitoring period. In Figure 4.5(a), the weekly median spreading factor remained near SF8 or SF9 during many weeks. However, several periods depart from such pattern and reach SF11 or SF12, which correspond to more robust transmission settings with a longer airtime.

The temporal profiles differ between the three nodes. Node 1 shows a marked increase of the median SF in late January and February 2025, while Node 2 remains near SF12 for several weeks around December 2024 and January 2025. Node 3, in

turn, shows shorter increases in October and November 2024. As a result, the timing and duration of such changes were not the same across the deployment.

Figure 4.5(b) presents the weekly median data rate for the same nodes. In general, lower DR values appear during the periods of higher SF, following the inverse relation between spreading factor and data rate in LoRaWAN. The duration of the periods with low DR also varies between nodes along the monitoring period.

All together, the SF and DR trends show that the transmission configuration changed over time. Rather than following a common fixed profile, each node followed its own sequence of parameter adjustments. Such differences may reflect changes in the radio conditions, in the gateway reception or in the ADR decisions of the network server during the deployment.

For this reason, the framework stores SF and DR with every logical uplink. The recorded values preserve the transmission parameters associated with each measurement, and support the later comparison of communication behavior, data availability and link conditions of each node (Table 4.4).

Table 4.4 - ADR transmission parameter summary by sensing node

Node	Number of uplinks	Median SF	Dominant SF	SF range	Median DR	Dominant DR	DR range
1	2	3	4	5	6	7	8
Node 1	45,071	9.0	7	7 - 12	3.0	5	0 - 5
Node 2	45,002	9.0	7	7 - 12	3.0	5	0 - 5
Node 3	44,927	9.0	7	7 - 12	3.0	5	0 - 5

Table 4.4 summarizes the transmission parameters related to ADR for the three sensing nodes. The deduplicated uplink counts are closely balanced, namely 45,071 for Node 1, 45,002 for Node 2 and 44,927 for Node 3. In this way, each node contributed a similar share of the uplink dataset.

All three nodes had a median spreading factor of 9, and a dominant spreading factor of 7. The observed range extended from SF7 to SF12. SF7 was used most often, but the higher values show that the nodes also operated with more robust transmission settings during part of the monitoring period.

The SF and DR values were obtained directly from the reception metadata reported by the LoRaWAN network server and were analysed as recorded. In other words, the DR values were not derived from the observed spreading factors. The recorded range of data rates extended from DR0 to DR5. Table 4.4 reports the observed SF and DR statistics separately for each sensing node. The results show similar overall ADR profiles across the nodes, although the selected transmission settings changed over time. SF and DR were kept with each logical uplink, since they describe the transmission configuration associated with each measurement.

Then, the relation between the transmission settings and the radio conditions was examined. For this purpose, the RSSI values at reception level from radio_rx.csv were grouped by the spreading factor of the corresponding logical uplink, and both datasets were matched through uplink_key.

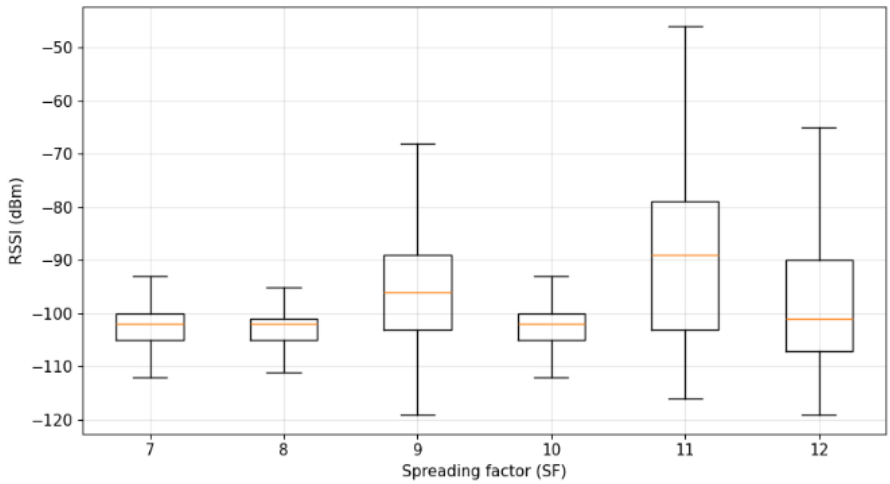


Figure 4.6 - Distribution of reception-level RSSI values grouped by uplink spreading factor

Figure 4.6 shows the distribution of the RSSI values at reception level for each spreading factor. SF7 and SF8 are associated with stronger median RSSI values, whereas SF11 and SF12 occur more often at weaker signal levels, with a wider variation. Such pattern agrees with the normal operation of LoRaWAN, where higher spreading factors are most often used under less favourable link conditions. However, the relation is not absolute, since the ADR decisions may also depend on the packet history, the gateway observations and the settings of the network server.

The reception diversity was examined separately from the relation between SF and RSSI. One logical uplink may produce several gateway reception records, each of them with the gateway identifier, the reception timestamp, the RSSI, the SNR, the spreading factor and the data rate. For this reason, the analysis used radio_rx.csv, where every row represents one gateway reception (Figure 4.7).

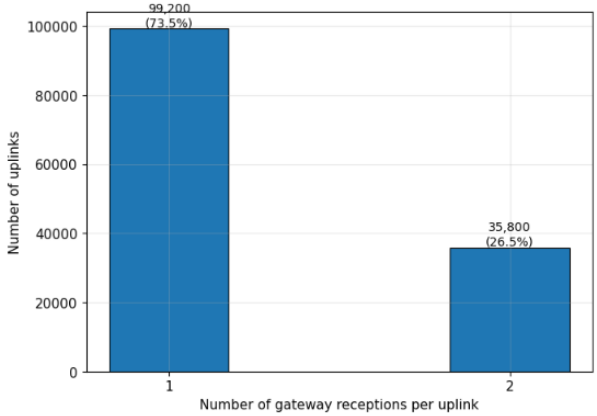


Figure 4.7 - Distribution of gateway reception counts per logical uplink

Figure 4.7 shows the number of gateways that received each logical uplink. A single gateway received 99,200 uplinks (i.e., 73.5% of the dataset), while the remaining 35,800 uplinks (i.e., 26.5%) were received by both gateways. As a result, approximately one quarter of the observed transmissions were received by several gateways.

Then, the weekly missingness was compared with the median RSSI and SNR, in order to examine whether the periods of weaker reception coincided with a lower data availability. Link quality was treated as only one of the operational factors, since device downtime, network interruptions and processing failures may also produce missing measurements.

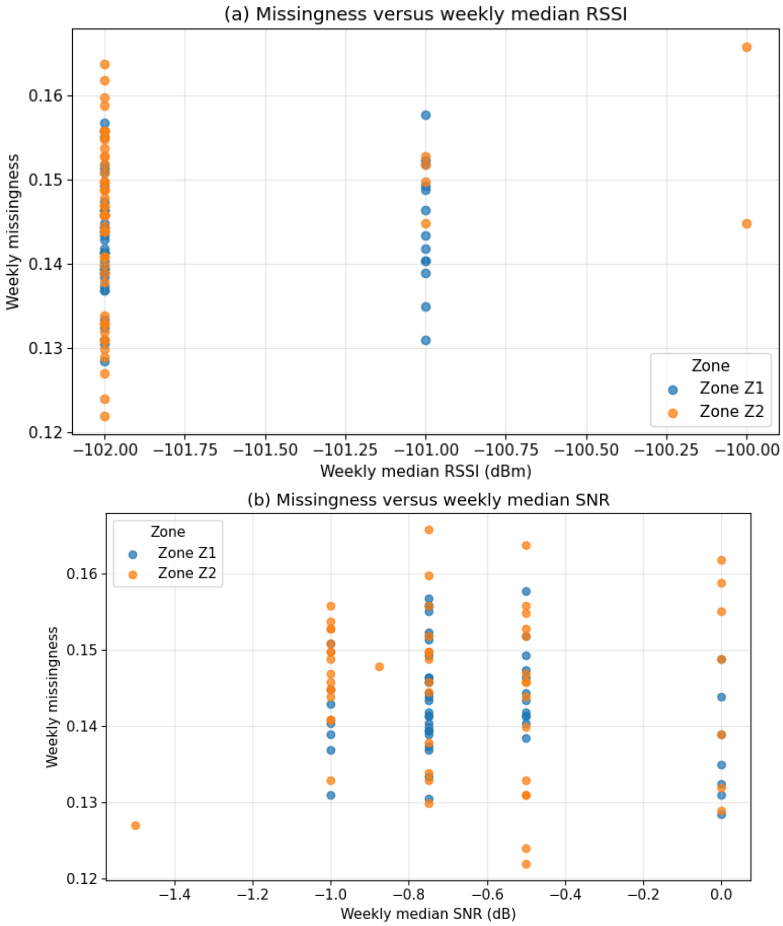


Figure 4.8 - Relationship between weekly missingness and link quality: a) missingness versus weekly median RSSI; b) missingness versus weekly median SNR

Figure 4.8 compares the weekly missingness with the weekly median RSSI and SNR. None of the panels shows a clear monotonic pattern, and weeks with similar link quality values still have different levels of missingness.

The Pearson and Spearman coefficients were calculated for both indicators. For RSSI, the Pearson coefficient was weak and positive ($r = 0.162, p = 0.098$), and the Spearman coefficient was also weak ($\rho = 0.121, p = 0.216$). For SNR, both coefficients were very weak and negative ($r = -0.080, p = 0.413$; $\rho = -0.092, p = 0.349$). In other words, none of the results indicates a strong association with the weekly missingness.

Radio conditions therefore play only a partial role in the variation of data availability. Besides weak links, missing intervals also may also be caused by device downtime, gateway outages, interruptions of the network server or failures during

processing. Despite this, RSSI and SNR remain useful as supporting operational evidence, although they cannot be taken as a direct explanation of the missing data.

4.4 Storage growth and replay-based platform profiling

Then, the evaluation examined the operational cost of the provenance capture. Two aspects were considered, namely the database growth during the 12-month deployment and the processing time under a controlled replay. The storage values were obtained from the deployed database, whereas the processing time was measured after the deployment, by replaying the recorded input data under fixed test conditions. The database contains measurement records, gateway reception records and provenance records. For this reason, the storage growth was measured separately for each component.

Figure 4.9 shows the database size over the monitoring period. The total storage reached 164.08 MB at the end of the deployment, with a growth close to linear as new measurements, radio records and provenance records were accumulated.

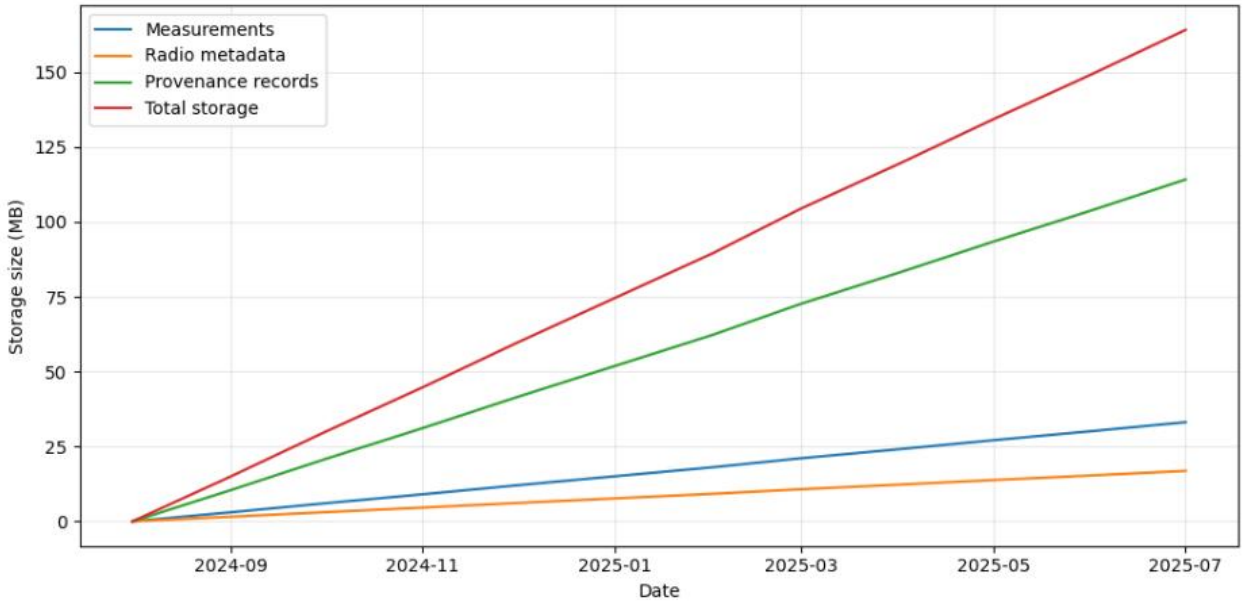


Figure 4.9 - Database storage growth by component during the monitoring period

The final database size was divided into measurement records, radio metadata and provenance records. The storage share of the provenance records was calculated as:

$$O_{prov}^{total} = \frac{B_{prov}}{B_{total}} \times 100\% \quad (4.6)$$

where B_{prov} denotes the final size of the provenance repository, also B_{total} denotes the combined size of all three storage components.

At the end of the monitoring period, the measurement records occupied 33.10 MB and the radio metadata 16.84 MB. The provenance records occupied 114.14 MB, for a total database size of 164.08 MB. The corresponding storage shares were

20.17% for measurements, 10.26% for radio metadata and 69.56% for provenance records.

The provenance records formed the largest part of the database, since each logical uplink generated records for the RX, INGEST and STORE stages. Even with such additional information, the complete 12-month dataset required only 164.08 MB for three sensing nodes, so the observed storage demand remained practical at the evaluated deployment scale. Moreover, the number of provenance records agrees with Formula (2.13). With 135,000 logical uplinks received by the platform and 133,900 stored measurements, the formula gives $2 \times 135{,}000 + 133{,}900 = 403{,}900$ records, which equals the number of records in the deployed provenance repository (Figure C.5). As a result, the provenance volume is determined by the number of received and stored uplinks, and can be estimated in advance for larger deployments.

Although the provenance records formed the largest storage component, the total size remained moderate for a 12-month deployment with three sensing nodes. In this sense, the storage overhead of the provenance capture was manageable at the evaluated scale. However, in larger deployments, the growth of provenance records should be taken into account during the database sizing and the design of the retention policy.

A separate controlled benchmark was conducted in order to measure the ingestion overhead on the platform side, with and without the generation of provenance. The same set of 5,000 logical uplinks was processed in baseline and provenance-aware modes. The baseline mode stored the measurement and the associated radio reception data, whereas the provenance-aware mode also generated the corresponding RX, INGEST and STORE provenance records (see Table 4.5).

Table 4.5 - Controlled platform-side ingestion benchmark with and without provenance generation

Metric	Baseline	Provenance-aware
1	2	3
Input per run	5,000 logical uplinks	5,000 logical uplinks
Mean total processing time	8.183 s	11.544 s
Mean throughput	611.06 ± 3.81 uplinks/s	433.19 ± 4.75 uplinks/s
Mean run-level median latency	1.565 ± 0.003 ms	2.189 ± 0.008 ms

As shown in Table 4.5, the provenance-aware ingestion showed a measurable processing overhead with respect to the baseline, across the repeated benchmark runs. The mean processing time increased from 8.183 s to 11.544 s, while the median latency per uplink increased from 1.565 ms to 2.189 ms. The mean throughput, in turn, decreased from 611.06 to 433.19 uplinks/s. Such results quantify the additional cost on the platform side for the generation and storage of provenance records, under the evaluated conditions of MySQL transactions.

A separate replay experiment was conducted after the field deployment, since the continuous processing latency had not been recorded during the data collection. The experiment used 133,900 successfully decoded measurement rows, and passed them through the implemented procedures of parsing, validation, SQL and

provenance generation. Three provenance records were created for each row, for a total of 401,700 RX, INGEST and STORE records. The replay ran on a Windows 10 computer with Python 3.10 and MySQL 8.0, and the database commits were performed in batches of 1,000 rows.

The median processing time was 0.2781 ms per measurement row, with a p95 value of 0.3522 ms. The appending of provenance required a median of 0.0734 ms, with a p95 value of 0.0954 ms. In this sense, the measured times show a low local processing cost for the provenance generation under the selected replay conditions.

Note that the replay results do not represent the end-to-end latency of the deployment. The input already contained decoded measurement rows, and the experiment excluded the LoRaWAN transmission, the processing at the network server, the MQTT delivery and any concurrent database activity. As a result, the values describe only the local cost of the replayed platform operations.

In summary, the storage analysis and the replay experiment show that the provenance capture added a measurable cost to the system. The provenance records accounted for the largest share of the database storage, whereas the local generation of records required less than one millisecond per measurement row. The runtime of the integrity verification is evaluated separately in the next section.

4.5 Integrity verification and provenance traceability

The integrity evaluation used the final provenance dataset collected during the 12-month deployment. The dataset contained 403,900 records, organised in three chains (i.e., one per device). Each record stored a SHA-256 hash, together with the hash of the previous record.

The verifier processed the records in chain order. For each record, the verifier recalculated record_hash from the canonical representation defined in Section 2.4.4, and then compared prev_hash with the hash of the previous record. The chain of a device was accepted only when all the record hashes and the links to predecessors were valid.

The experiment first measured the runtime required to verify the complete dataset. Then, modified copies of the chains were used to test five changes in the stored history, namely the modification of content, the deletion of an intermediate record, the insertion of a fake record, the reordering of consecutive records and the truncation of the tail. The same provenance records were also used for the traceability tests, where a selected measurement was followed backward to its uplink and reception records, and forward through the INGEST and STORE stages.

The complete verification processed 403,900 provenance records in 5.87 s, which corresponds to 0.0145 ms per record. The original dataset contained no invalid hashes or broken links, and the verifier detected all five controlled changes.

The modification of content produced a mismatch between the stored and the recalculated record hashes. The deletion, insertion and reordering broke the expected relation between prev_hash and the previous record_hash. However, the truncation of the tail produced a different result. The last ten records were removed from the chain of one device, but the remaining 134,812 records kept valid internal hash links. As a

result, the ordinary chain verification reported no invalid hashes or broken links. The verifier then compared the current number of records and the last record hash with the previously stored checkpoint. Since both values differed from the checkpoint, the truncation was detected.

In the controlled scenario of tail truncation, the comparison with the checkpoint required 0.9391 ms. Such value represents the runtime of a single local comparison of the current number of records and the last chain hash with the stored checkpoint, rather than an average over repeated trials.

In the deletion scenario (Figure 4.10), one intermediate record was removed from a valid chain of a device. The next available record still referenced the hash of the deleted record, so the verifier reported a broken link of the chain at that position.

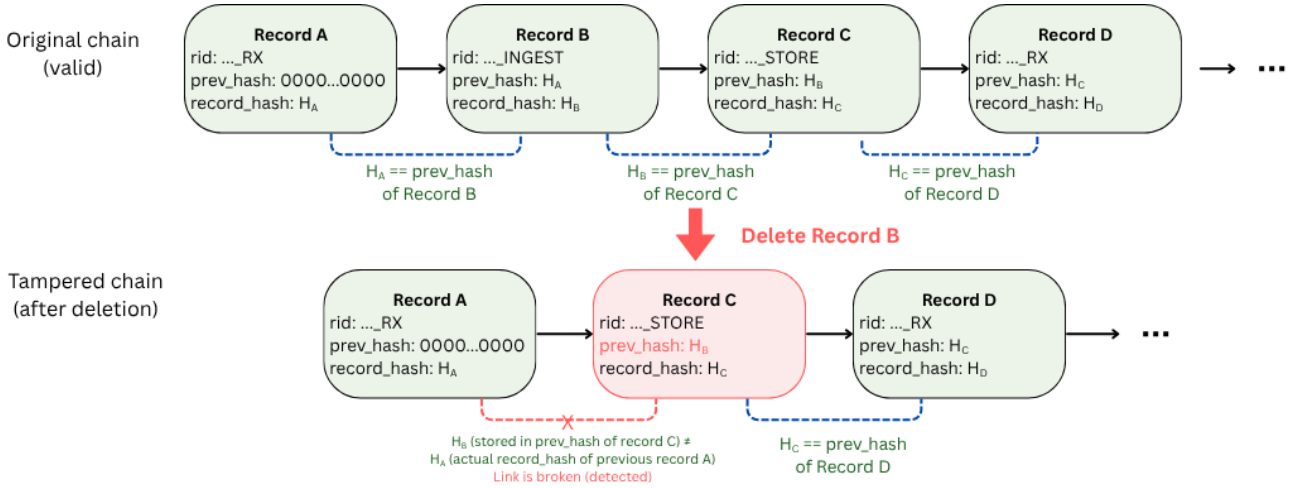


Figure 4.10 - Detection of intermediate record deletion in a hash-linked provenance chain

Figure 4.10 shows the effect of the deletion of Record B from the chain. Record C still contains the hash of Record B in its prev_hash field, even though Record A has become the previous available record. For this reason, the verifier reports a linkage error at Record C. Figure C.6 shows the corresponding output of the system.

The second scenario tests the reordering of records. Two consecutive provenance records are exchanged, without any change in their stored hashes. In such case, the new sequence no longer matches the references to predecessors recorded in the chain, and the verification detects the resulting linkage errors.

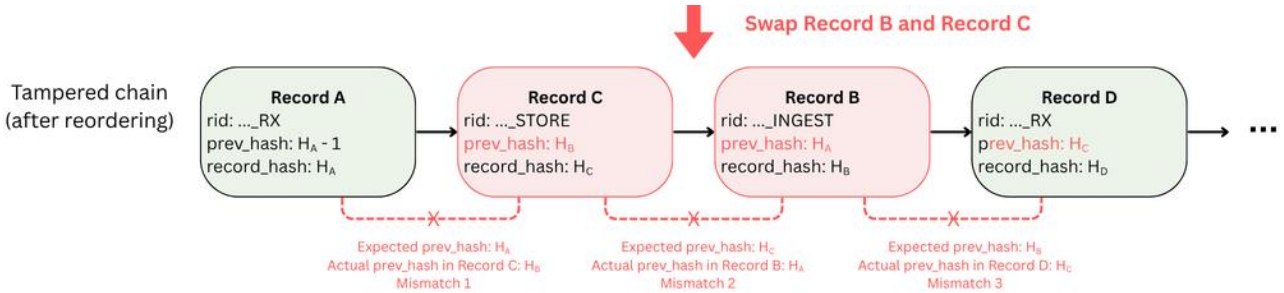


Figure 4.11 - Detection of record reordering in a hash-linked provenance chain

Figure 4.11 shows the effect of swapping two consecutive records. The modified order no longer matches the predecessor references stored in the chain. The verifier reports three linkage errors around the exchanged records because their prev_hash values do not match the record_hash values of the records that now precede them. Figure C.7 shows the corresponding system output.

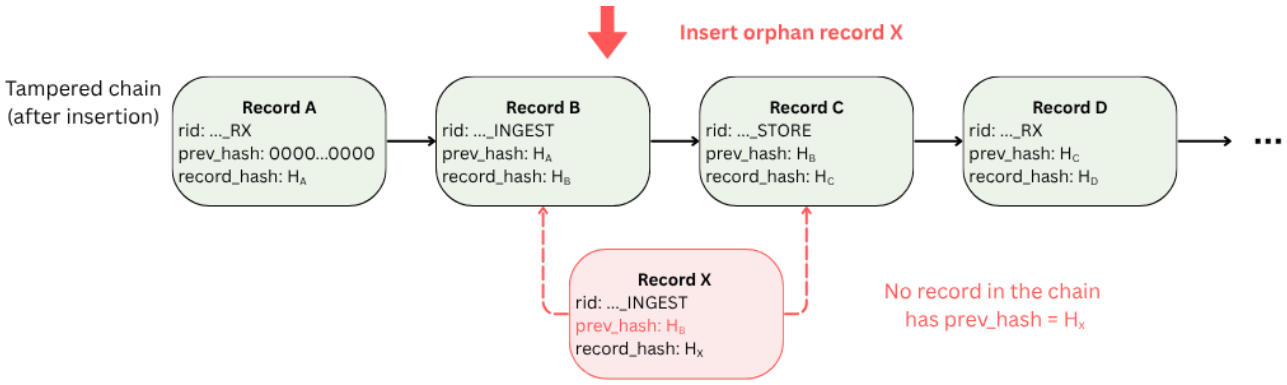


Figure 4.12 - Detection of orphan record insertion in a hash-linked provenance chain

Figure 4.12 shows the effect of inserting an orphan record into a valid chain. The inserted record does not contain the correct hash of the preceding valid record. The next valid record also retains the hash of its original predecessor. The verifier therefore reports two linkage errors. Figure C.8 shows the corresponding system output. Table 4.6 summarises the results of the five controlled tampering-detection experiments. The original chain was verified before modification. Each test then applied one artificial change to a copied dataset. The reported error count shows the local inconsistencies introduced by that change.

Table 4.6 - Results of the controlled tampering detection experiment

Scenario	Artificial modification	Detection condition	Result
1	2	3	4
Baseline valid chain	No modification	No hash, linkage, or checkpoint inconsistency	Valid
Content modification	Changed one integrity-relevant field without updating *record_hash*	Recalculated *record_hash* does not match the stored value	Detected
Intermediate-record deletion	Removed one record from the middle of the chain	The next available record contains an incorrect *prev_hash* reference	Detected
Orphan-record insertion	Inserted a fabricated record into a valid chain	The fabricated record and the following valid record produce hash or predecessor-link inconsistencies	Detected

Continuation of the Table 4.6

Scenario	Artificial modification	Detection condition	Result
1	2	3	4
Record reordering	Swapped two consecutive records	The stored *prev_hash* references do not match the new record order	Detected
Tail truncation	Removed the final ten records from one device chain	The current record count and final chain hash do not match the protected checkpoint	Detected

The original provenance chain passed all structural checks. Each modified copy produced the expected linkage errors. Deletion caused one error at the first record after the removed entry. Reordering caused three errors around the exchanged records. Orphan insertion caused two errors: one at the fabricated record and one at the following valid record. The tests show that the per-device hash chain records evidence of post-ingestion structural changes under the trusted-platform assumption. Appendix C contains the corresponding Jupyter Notebook outputs.

The five scenarios above change the chain without recomputing the downstream hashes. An adversary with write access to the provenance repository (adversary A1, Section 2.3) can, however, recompute an unkeyed chain, because SHA-256 and the canonical record format are public. Three additional scenarios therefore tested consistent rewriting on a synthetic chain set with the same size and record structure as the deployment (403,900 records in three device chains), with a checkpoint taken at 90% of the chain length. With the unkeyed chain, a rewrite of a record appended after the checkpoint passed both the chain verification and the checkpoint comparison, whereas a rewrite of a record in the checkpointed part was detected by the checkpoint. With the keyed chain (Formula 2.24), both rewrites were detected, because hashes recomputed without the key did not match the stored keyed hashes. In this benchmark, hash computation alone took 0.32 s for the SHA-256 chain and 0.95 s for the HMAC chain (median of five runs), i.e. about 1.6 µs of additional cost per record. Compared with the 5.87 s needed for full verification of the deployment dataset, which is dominated by data access, this is an increase of roughly 11% in verification time. The unkeyed chain therefore protects only the checkpointed part against an adversary with repository write access, and the checkpoint interval bounds the exposure window; the keyed chain removes this window at a small computational cost.

The evaluation then examined provenance traceability. Each stored measurement retains links to its uplink, gateway receptions, and processing records. These links allow the platform to reconstruct the path followed by a measurement before it entered the database.

Figure 4.13 shows one of the reconstructed provenance paths. The path begins with the LoRaWAN uplink, and continues through the RX, INGEST and STORE stages up to the final measurement record. Along the way, the gateway reception data and the entity identifiers connect the records of the different stages.

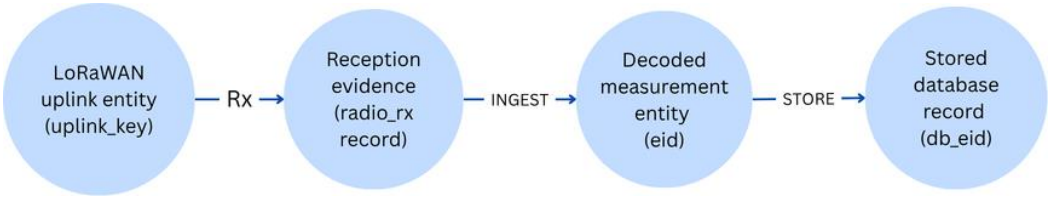


Figure 4.13 - Reconstructed provenance path from uplink reception to stored measurement

The reconstructed path connects each stored measurement to the corresponding uplink, gateway receptions, decoding result and storage event. In this way, an audit query is able to recover the communication and processing history of the selected value. The verification and traceability results address different properties of the provenance records. On one side, the verification of the hash chain checks whether the stored history remains internally consistent. On the other side, the lineage reconstruction shows how a measurement moved through the RX, INGEST and STORE stages.

In order to demonstrate the reconstruction of the provenance graph, a representative PM2.5 measurement was selected (Figure 4.14). The query followed the stored links between entities, from the logical uplink entity through the processing on the platform, up to the final database record.

	Audit item	Result
0	Selected PM2.5 value	14.0 µg/m³
1	Stored DB record	db_eid_000001_202408010000_65c71d
2	Original uplink	upl_000001_202408010000_65c71d
3	Gateway reception	[gw-1]
4	Decode failure	No
5	Forward trace RX → DB	Found
6	Backward trace DB → RX	Found
7	Hash-chain linkage	Valid

Figure 4.14 - Reconstructed provenance path for one PM2.5 measurement

Figure 4.14 shows the reconstructed processing path of the selected PM2.5 measurement. The logical LoRaWAN uplink entity is linked to the corresponding reception evidence and then to the INGEST and STORE activities. The final node represents the measurement record stored in the database. The reconstructed path preserves the sequence of platform-side operations applied to the selected uplink. In the audit output, the label Original uplink denotes the stored logical uplink identifier, not a retained raw MQTT message or FRMPayload.

The reconstruction result was also examined through backward and forward provenance queries. The resulting audit output identifies the selected measurement, its source uplink, gateway reception evidence, processing activities, stored database

entity, and corresponding provenance records. Figure 4.15 presents the audit results and the reconstructed graph nodes returned for this measurement.

	Step	Graph node	Role
0	1	rid_000001_202408010000_65c71d_RX	RX activity
1	2	upl_000001_202408010000_65c71d	Uplink entity
2	3	rid_000001_202408010000_65c71d_INGEST	INGEST activity
3	4	eid_000001_202408010000_65c71d	Decoded measurement entity
4	5	rid_000001_202408010000_65c71d_STORE	STORE activity
5	6	db_eid_000001_202408010000_65c71d	Stored DB record

Figure 4.15 - Audit results and reconstructed graph nodes for the selected PM2.5 measurement

Figure 4.15 presents the audit results for a PM2.5 value of 14.0 µg/m^3. The backward query recovered the logical uplink identifier and gateway reception records from the stored database entity. The forward query followed the uplink through the RX, INGEST, and STORE records to the stored measurement. Both query directions returned complete paths identifier-linked paths within the platform-side lifecycle.

The reconstructed view contained three activity records and three entity records. The RX, INGEST, and STORE records represented the platform processing stages. The uplink, decoded measurement, and database record represented the related entities. The corresponding hash-chain linkage was valid.

Case C1: Multi-gateway reception.

Node 1 transmitted the uplink identified by 000001_202408010502_a7ce55 at 2024-08-01 05:02:04, and the same transmission was received by two gateways. Gateway gw-1 reported an RSSI of -107 dBm and an SNR of 1.0 dB, while gateway gw-2 reported an RSSI of -105 dBm and an SNR of 4.0 dB. Both reception records contained SF7 and DR5.

The collector created only one measurement entity, namely eid_000001_202408010502_a7ce55. For the logical uplink, the provenance chain contained one RX record, one INGEST record and one STORE record. The two gateway observations remained in the radio metadata repository and referred to the same measurement entity.

Such case shows how the uplink_key prevents the repeated reception evidence from creating duplicate measurements. The radio data of each gateway remain available for the analysis of communication, whereas the decoded values and the processing history remain associated with one logical uplink.

Case C2: Missing interval without reception evidence.

Node 1 had no recorded uplink for the 10-minute interval at 2024-08-01 01:10:08. The neighbouring transmissions at 01:00 and 01:20 were received, decoded

and also stored under the uplink keys 000001_202408010100_c9470d and 000001_202408010120_ae376c.

For the 01:10 interval, no logical uplink was present. The radio repository contained no gateway reception, and the provenance repository contained no RX, INGEST or STORE record for Node 1 at that time. As a result, the missing interval occurred before the uplink was recorded by the platform, and was not caused by decoding, validation or database storage. However, the retained records cannot determine whether the device did not transmit, the frame was lost on the radio path, or it was lost between the network server and the collector. For such purpose, the sequence of frame counters is required, together with the event log of the network server, as described in Section 2.4.5 and evaluated in Section 4.6.

Other types of failure leave different evidence. A decoding failure keeps the received payload, the reception record and the INGEST outcome, whereas a validation failure produces a measurement record with the corresponding quality indicator. In this way, an uplink that was not received can be distinguished from an uplink that failed after its reception.

Cases C1 and C2 illustrate two different operating conditions. Case C1 contains several gateway observations for one logical uplink, but only one measurement entity. Case C2, in turn, contains no reception or processing evidence for an expected reporting interval. The links between repositories allow both conditions to be examined at event level.

The controlled tampering tests, the chain verification results and the reconstructed lineage cases assess separate functions of the provenance mechanism. The verification of the hash chain identifies changes in the stored provenance history, whereas the lineage queries recover the reception and processing context of individual measurements. In summary, the measured verification runtime was low for the evaluated dataset, while the reconstructed records provided an interpretable audit trail for both received and missing uplinks.

4.6 Simulation-based evaluation of missing-data attribution

During the field deployment, the true cause of each missing uplink could not be determined, and the event log of the network server, required to separate the losses at platform intake from the upstream losses, was not kept. For this reason, the attribution model described in Section 2.4.5 was evaluated by means of simulated deployment scenarios, in which the cause of each missing measurement is known.

The simulation has two layers. The radio layer was modelled with the ns-3 network simulator (version 3.48) and its LoRaWAN module (version 0.3.7) [171]. The module models the path loss, the LoRa spreading factors, the gateway sensitivity and the collisions between concurrent transmissions. For every transmission of the target devices, the module records which gateways received the frame. The second layer, written in Python, adds the failures that are not modelled by ns-3. In particular, the device failures are followed by an OTAA re-join with a specified probability, which resets the frame counter. Moreover, the second layer models the gateway failures, the losses at platform intake (i.e., message loss between the network server

and the collector, collector failure and transaction rollback, class P0), the payload decoding failures (i.e., class P1), the frames without a frame counter and the jitter of the transmission time. Then, the second layer produces only the data recorded by the platform, namely the event log of the network server (optional) and the provenance records written by Algorithm 3.1. Finally, the attribution procedure (Algorithm 3.3) is applied, and the resulting classes are compared with the true causes.

The path loss follows the log-distance model of the LoRaWAN module (with an exponent of 3.76), with a log-normal fading per packet (with a standard deviation of 6 dB). The positions of the devices were not measured in the deployment, so they were calibrated. More specifically, the positions were chosen so that the share of uplinks received by both gateways (27.3% in the simulation) and the median spreading factor (SF9) match the values of the deployment (26.5% and SF9, Section 4.3). With such geometry, 95.1% of the transmissions were received by at least one gateway. The resulting distances (Figure 4.16) are larger than in the urban deployment (Figure 4.1), since the log-distance model describes the propagation without buildings and other obstacles. In other words, the simulated geometry reproduces the path loss and the reception statistics of the deployment, rather than its physical layout. RSSI and SNR were not used for the calibration, since the values reported by commercial gateways cannot be compared directly with the received power computed by ns-3. ADR was disabled and the spreading factor was fixed, since the attribution rules do not use the transmission parameters. Last, the outage rates were set so that the simulated completeness (87.1 ± 0.4%) is close to the completeness of the deployment (85.6%).

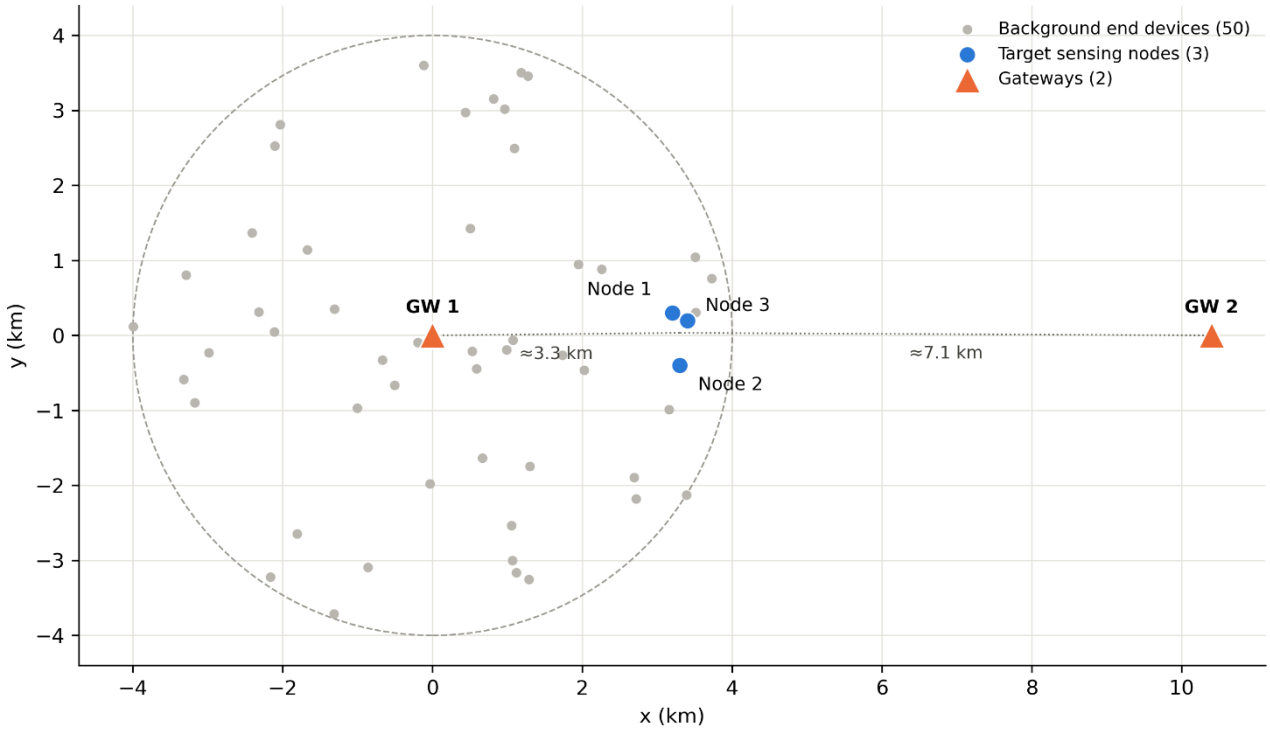


Figure 4.16 - Topology of the simulated network (radio seed 1): two gateways, three target sensing nodes at calibrated positions, and 50 background end devices

placed uniformly within 4 km of gateway 1. Distances are equivalent path-loss distances, not the physical distances of the deployment

Table 4.7 lists the parameters, together with the varied factors. The baseline consists of ten independent ns-3 runs of 365 days, each of them combined with two realisations of the second layer (i.e., 20 runs in total). Each sensitivity scenario uses five ns-3 runs with two realisations (i.e., 10 runs). The software environment, the source code, the commands used to run the simulation and its output are given in Appendix D.

Table 4.7 - Parameters of the attribution simulation

Layer	Parameter	Baseline value	Varied in scenarios
1	2	3	4
Radio (ns-3)	Duration, devices, interval	365 days; 3 target devices; 600 s	-
Radio (ns-3)	Gateways	2 (calibrated geometry)	1, 3
Radio (ns-3)	Path loss and fading	Log-distance (exponent 3.76), log-normal fading with SD 6 dB	SD 3 dB, 10 dB
Radio (ns-3)	Background devices	50	0; 500 (30-day runs)
Radio (ns-3)	Spreading factor	SF9, ADR disabled	-
Python	Device outages	0.5 per day, mean 3 h	0.05, 0.2, 1.0 per day
Python	Re-join after outage (FCnt reset)	Probability 0.7; spontaneous re-joins 0.02 per day	Probability 0, 0.5, 1
Python	Gateway outages	0.15 per day per gateway, mean 4 h	0, 0.5 per day
Python	Platform intake loss (P0)	Message loss 0.2%, rollback 0.05%, collector outages 0.02 per day (mean 1.5 h)	Network-server event log on / off
Python	Decoding failure (P1)	0.8%	-
Python	Frames without FCnt	0%	1, 5, 10, 20%
Python	Timing jitter (SD)	5 s	60, 120, 240 s

Frames without a frame counter are handled as described in Section 2.4.5: they are mapped to slots by their reception time, but they do not bound gaps. Three metrics were used. Coverage is the share of missing slots that receive a single-stage label (U1, U2, P0, or P1), as defined by Formula 2.32; the remaining slots are

labelled MIXED or U3. Accuracy is the share of these single-stage labels that match the true cause. Precision and recall were computed for each class.

In the baseline scenario, 406,987 of 3,153,596 expected slots were missing over 20 runs: 47.1% due to upstream loss (radio losses and gateway outages), 45.1% due to device outages, 2.4% due to platform intake losses, and 5.5% due to decoding failures. Figure 4.17 shows the resulting attribution.

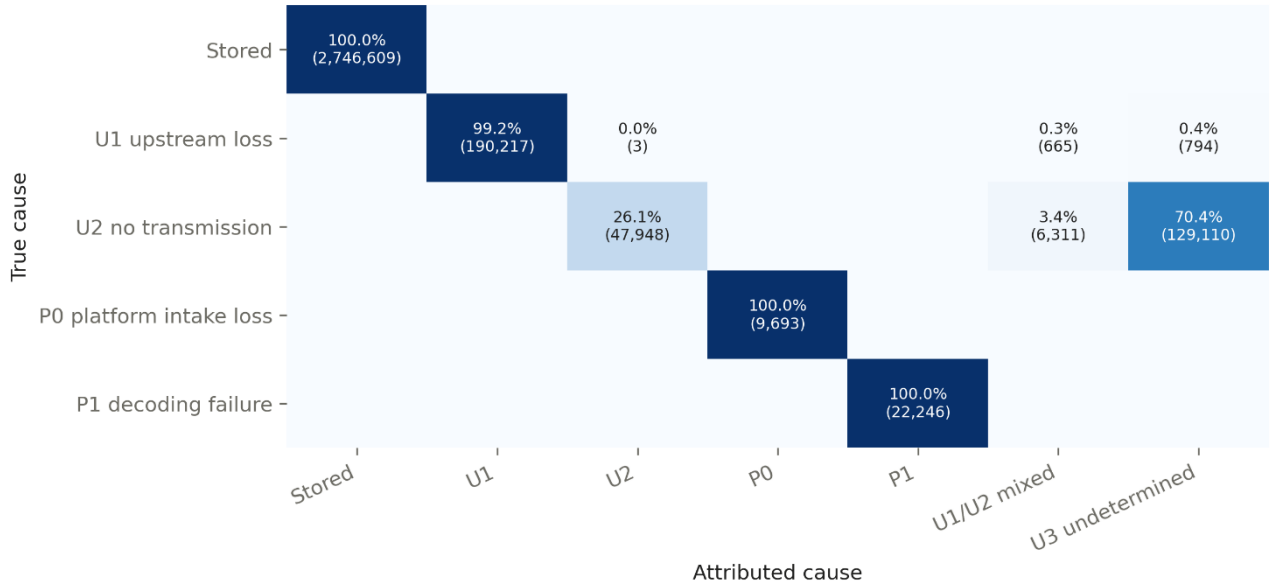


Figure 4.17 - Attribution of expected uplink slots in the baseline scenario (ns-3 radio layer, 365 days, three devices, 20 runs; row percentages and slot counts)

Among the 270,107 labels assigned to a single stage, only three were wrong. All the losses at platform intake (i.e., P0) and the decoding failures (i.e., P1) were identified, and 99.2% of the upstream losses were attributed to U1. The coverage was 66.4 ± 1.5%. The uncovered share consists almost entirely of device outages, of which 26.1% were attributed to U2, 3.4% fell into mixed gaps and 70.4% were labelled as undetermined. In the baseline, 70% of the device outages end with a re-join. In such case, the silent interval is not bounded by two values of the same frame counter, and the model labels it as U3, instead of guessing. As a result, the coverage is determined by the composition of the losses, rather than by errors of the method. Figure 4.18 shows the precision and recall per class, with and without the event log of the network server.

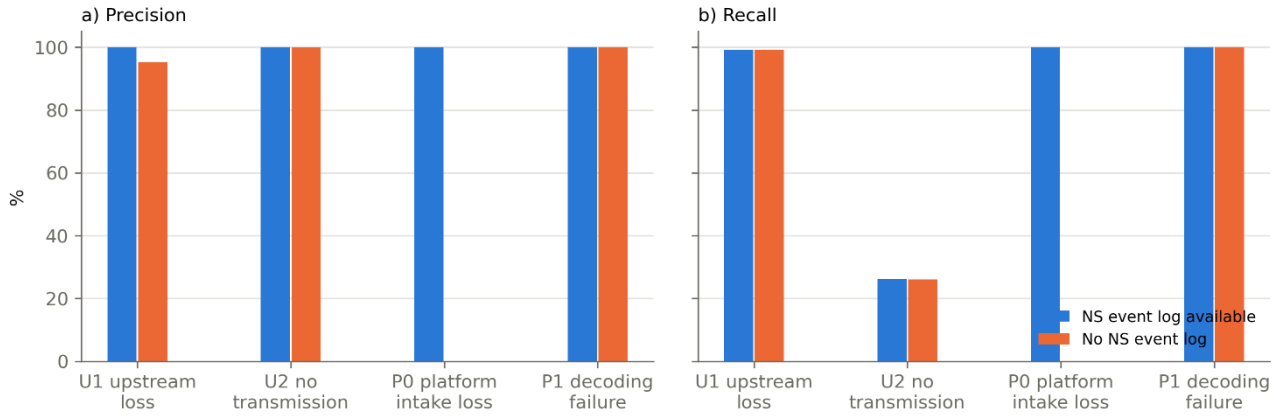


Figure 4.18 - Precision (a) and recall (b) of the attribution per class, with and without the network-server event log (baseline scenario)

With the event log, the precision of all four classes was at least 99.99%. Without the log, however, a frame accepted by the network server but not recorded by the platform leaves no evidence, and its counter value appears as a gap in the RX sequence. As a result, 97.8% of the losses at platform intake were attributed to upstream losses, the recall of P0 fell to zero, the precision of U1 fell to 95.3% and the overall accuracy fell to 96.5%. Since the coverage remained practically unchanged (66.3%), such error is not visible to the analyst. In this sense, the result supports treating the event log as a required component, and is also one of the reasons why the attribution was not applied to the field data. Figure 4.19 summarises the sensitivity scenarios.

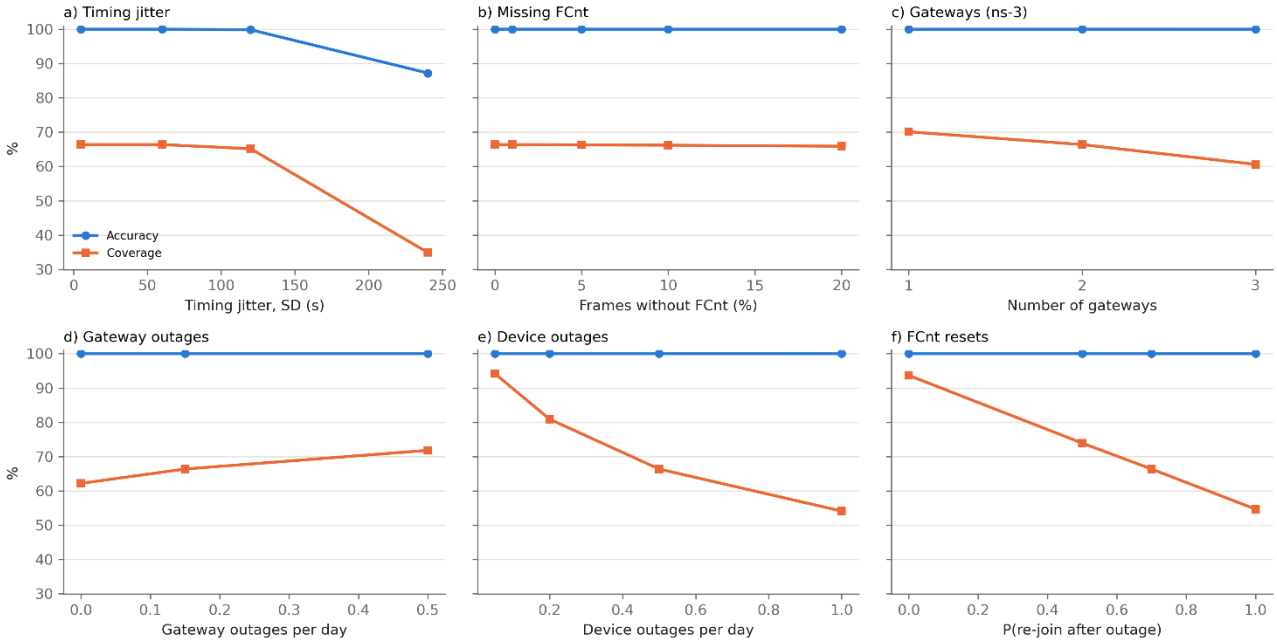


Figure 4.19 - Accuracy and coverage of the attribution under: a) timing jitter; b) missing frame counters; c) number of gateways; d) gateway outages; e) device outages; f) probability of a re-join after an outage

Accuracy remained above 99.9% in all the scenarios with the event log of the network server, except for a large timing jitter. A jitter up to 60 s had no effect. At

120 s (i.e., one fifth of the reporting interval), the accuracy was 99.8%. At 240 s, however, the frames were increasingly mapped to neighbouring slots, and the accuracy fell to 87.2%, with a coverage of 34.9%. The missing frame counters had almost no effect. With 20% of the frames without a counter, the coverage decreased only from 66.4% to 65.9%, since such frames still mark their slots and are taken into account in the counter difference.

The radio scenarios simulated with ns-3 changed the composition of the losses, but not the accuracy. With one gateway, 93.2% of the transmissions were received and the coverage was 70.1%, whereas with three gateways the values were 97.0% and 60.6%, respectively. A fading with a standard deviation of 3 dB and 10 dB gave reception rates of 99.9% and 88.1%, and coverages of 52.2% and 76.0%. The background traffic had practically no effect on the reception. With 0 and with 500 background devices, 95.0% of the transmissions were received, compared with 95.1% in the baseline with 50 devices. In all such cases, the coverage increased with the share of upstream losses, which are always determinate, and decreased with the share of device outages. The scenarios of device outages confirm such trend, since the coverage fell from 94.2% at 0.05 outages per day to 54.1% at one outage per day. Without re-joins, the coverage reached 93.7% and the recall of U2 87.1%. In contrast, when every outage ended with a re-join, the coverage was 54.7%, and almost all the device outages remained undetermined.

In summary, the simulation shows that the attribution rules are accurate whenever they assign a single stage, including under the radio conditions simulated with ns-3, the gateway outages and the missing frame counters. The main limitation of the rules is the coverage, rather than the accuracy. In particular, the device outages followed by a re-join cannot be distinguished from upstream losses by means of the frame counters alone, and are reported as undetermined. The device status fields already transmitted by the sensing nodes, like the reset counter and the uptime, could narrow such intervals, and this extension is left for future work. Moreover, the event log of the network server is required to separate the losses at platform intake from the upstream losses. Note that the results were obtained with the stated loss models, so the validation on a deployment that keeps the event log also remains as future work.

4.7 Discussion and limitations

Table 4.8 summarises the quantitative results reported by the closest studies from Table 1.5 together with the results of this work. The studies were evaluated with different methods, workloads, and hardware, and they report different metrics, so the table shows what each study measured rather than ranking the approaches.

Table 4.8 - Quantitative results reported by the closest studies and by this work

Study	Setting	Reported quantitative results	Main difference from this work
1	2	3	4
[97]	Generic IoT; simulation	Not reported	No integrity protection; no overhead evaluation

Continuation of the Table 4.8

Study	Setting	Reported quantitative results	Main difference from this work
1	2	3	4
[95]	Generic IoT; testbed	5% device latency, 20.6% event latency; 260 KB of provenance per day; queries 2-9 ms	Stored logs are not tamper-evident
[92]	Generic IoT (MQTT broker); testbed	Added latency 40-50 ms per message	Broker-level capture; no integrity or missing-data handling
[107]	Wi-Fi/BLE; real deployment	Usability 4.67/5; no performance figures	Device changes; blockchain without external verification
[108]	Generic IoT; simulation	Validation time +17.4%; CPU overhead 1.3%	Device changes; simulation only; no missing-data handling
[103]	Generic IoT; testbed	Query latency ≤ 0.2 ms (p99.9); graph build 25 min - 3 h for 17-77 GB	Network-layer forensics; provenance store not protected
[72]	WSN; simulation	Link-loss detection 75-100%; false positives ≤ 0.013; 2.2-4.2 mJ per 1000 transmissions	Device-side MACs; attributes malicious drops, not pipeline stages
[88]	Wi-Fi/BLE; testbed	Alert latency 1.5-5 min; 38 anchoring transactions per day	Device and gateway changes; only modification tested; gaps only detected
[114]	Generic IoT; testbed and simulation	Commit 2.5 ms (LAN), 58 ms (WAN); verification 0.88 ms (cold), 15 µs (repeated)	No LoRaWAN or W3C PROV; missing records only detected
[113]	LoRaWAN; testbed	Signing 7.4 ms; latency 62 ms; 145 TPS; 100% detection	Device-side signatures and blockchain; missing data only detected
This work	LoRaWAN; 12-month deployment and ns-3 simulation	+0.62 ms per uplink (from 1.565 to 2.189 ms); 403,900 records verified in 5.87 s; 5 out of 5 changes detected; 3 errors out of 270,107 attribution labels	Trusted local operator; attribution evaluated by means of simulation only

The studies report different metrics under different conditions, so the values in Table 4.8 are not directly comparable. However, the comparison reveals two

differences. First, studies with robust integrity guarantees either require device modifications, as in five out of ten studies, or do not use LoRaWAN and W3C PROV. The proposed framework adds no data to the device payload and verifies the stored 12-month LoRaWAN deployment history in seconds. Second, the studies reviewed either only detect missing records or attribute losses to malicious nodes; none of them attribute missing measurements to pipeline stages. The additional processing time of 0.62 ms per uplink is small compared to the 600-second reporting interval.

The results should be interpreted within the framework of three sets of assumptions and limitations.

Trust Assumptions. As with most platform-side integrity mechanisms, guarantees apply only as long as the platform components are considered trusted. These include the data ingestion service, provenance store, verifier, chain key, and checkpoints. This assumption was deliberately chosen. It avoids the latency and costs associated with blockchain while still detecting changes made by users interacting with the database. An attacker who controls the platform before records are created is outside the threat model. The same applies to an attacker who can replace both records and checkpoints. Storing checkpoints in write-once storage or with an independent party would extend protection to the platform operator. The 12-month deployment chains were keyless. A keyed chain also protects records added after the last checkpoint. It was added after deployment and tested on a synthetic dataset of the same size.

Scope of the recorded evidence. Provenance describes the history of a value, not its physical correctness. Calibration errors, sensor drift, or faulty equipment result in values that still match their provenance records. Detecting such values is the task of data quality assessment methods that can use recorded provenance as context. RSSI and SNR are values reported by gateways. They are used as indicators of link state, not as calibrated measurements. The prototype does not store raw data from successfully decoded uplinks, so decoding cannot be repeated independently. Storing raw FRMPayload data or a link to an archive with its hash and decoder version alleviates this limitation with minimal storage overhead.

Scope of evaluation. The deployment used three sensor nodes and two gateways. The cost of verification grows linearly with the number of records, and chains are stored for each device. Therefore, larger deployments can be processed through incremental verification and database partitioning. Processing latency was measured by replay. It represents the local cost of record generation, since radio transmission and MQTT delivery are not affected by this structure. The distribution of missing data was estimated using simulation, since the network server event log was not stored in the deployed system. To apply the method to field data, it is sufficient to enable the network server's built-in database integration. Device failures that result in reconnection remain undetermined. The reset counter and uptime fields, which are already transmitted by monitoring nodes, can partially resolve these gaps.

Conclusions on Chapter 4.

A 12-month deployment demonstrated that the platform can record data provenance for a real-world LoRaWAN monitoring system without changes to

devices or gateways. Provenance records were generated for all 135,000 received uplinks, with observed and effective completeness of 85.6% and 84.9%, respectively. The cost of data collection remained moderate. Median processing latency increased from 1.565 to 2.189 ms per uplink, and provenance records occupied 114.14 of the 164.08 MB of the database.

The stored history was auditable and traceable. A full audit of 403,900 records took 5.87 s, and all five controlled changes to the stored history were detected. Consistent rewrite after the last checkpoint were detected only using the keyed chain. Backward and forward queries reconstructed the lineage of selected measurements and identified reception through multiple gateways, missed intervals, and decoding errors.

Simulations using the ns-3 radio layer validated the attribution method. Incorrect attribution of missing data was detected in only 3 of 270,107 single-stage labels. Coverage was 66.4% and was limited to device failures that resulted in reconnection. Without the network server event log, accuracy dropped to 96.5%. Therefore, this framework represents a platform-side mechanism for ensuring provenance integrity, traceability, and attribution of missing data, rather than a comprehensive end-to-end security solution.

CONCLUSION

This dissertation examines the issue of trustworthy data management in LoRaWAN-based smart city systems. Air quality monitoring was used as an application example. In such a system, measurements undergo several stages: sensing, radio transmission, reception at the gateway, processing on the network server, delivery via the MQTT protocol, decoding, verification, and storage. At each stage, data can be lost, duplicated, or later modified. Without provenance information, it is impossible to reconstruct the origin and processing history of a stored value. The reason for the missing value also remains unknown.

This dissertation proposes and implements a framework for provenance-aware framework of air quality data in LoRaWAN systems. Provenance information is captured on the platform through standard network-server integration. Provenance data is not added to the LoRaWAN payload. End devices and gateways remain unchanged.

The first objective was to analyze existing methods for ensuring data provenance and integrity for IoT and smart city systems. A systematic review revealed that existing solutions address only one aspect of the problem. Some studies record provenance but do not protect it. Others protect records but require device changes or the use of blockchain infrastructure. None of the studies reviewed attributed missing measurements to stages of the data processing pipeline. These gaps identified five design requirements. These include: full data collection at each stage without device modification, a standard provenance model with proof of receipt, tampering verification within the stated threat model, limited overhead, and the attribution of missing data to a specific stage.

The second objective was to develop the provenance, integrity, and attribution model. The provenance model follows the entity, activity, and agent concepts of W3C PROV. Gateway reception evidence is represented as a separate entity. Stable identifiers link the RX, INGEST, and STORE records of each uplink. A directed provenance graph supports backward and forward tracing. Provenance records are stored in append-only hash chains, one chain per device. In the current version, the chains are keyed with HMAC-SHA-256. Chain-head checkpoints are stored outside the provenance repository and make tail truncation detectable. The attribution model uses frame counters, reception times, record status, and the network-server event log. It assigns each missing measurement to a pipeline stage. When the evidence is not sufficient, the measurement is marked as undetermined.

The third objective was to implement a prototype. The system included three sensing nodes, two LoRaWAN gateways, a private ChirpStack network server, an MQTT ingestion service, and a MySQL database. Each uplink is processed in a single database transaction. A successful uplink produces the RX, INGEST, and STORE records together with the measurement. A decoding failure is kept as an INGEST record with the status FAIL. A storage error rolls back the whole transaction. When several gateways receive the same uplink, only one measurement entity is created. Each gateway reception is still kept in the radio repository.

The fourth objective was to evaluate the framework. The field deployment ran

from 1 August 2024 to 31 July 2025. Of the 157,680 expected uplinks, the platform received 135,000. Of these, 133,900 were decoded and stored. The observed and effective completeness were therefore 85.6% and 84.9%. The database reached 164.08 MB, and provenance records occupied 114.14 MB of it. In the controlled benchmark, the median processing latency increased from 1.565 ± 0.003 ms to 2.189 ± 0.008 ms per uplink. The added cost was thus 0.62 ms per uplink. The deployment produced 403,900 provenance records, which matches the value given by the formal model.

The full verification of the 403,900 records took 5.87 s (i.e., 0.0145 ms per record), and the original chains contained no invalid hashes or broken links. The controlled tests detected the modification of content, the deletion, insertion and reordering of records, as well as the truncation of the tail. A consistent rewrite of the records after the latest checkpoint was detected only with the keyed chain. The backward and forward queries recovered the uplink, the gateway receptions, the decoding result and the storage event of the selected measurements.

The attribution method was evaluated by means of simulations, since the event log of the network server was not kept during the deployment. The radio layer was simulated with ns-3 and its LoRaWAN module, and calibrated to the reception statistics of the deployment. In the baseline scenario, only 3 out of 270,107 labels assigned to a single stage were wrong. The coverage was 66.4%, mainly limited by the device outages that end with a re-join. Without the event log of the network server, the accuracy decreased to 96.5%, which confirms that the log is a required component of the framework.

The scientific novelty of the dissertation consists of three results. The main one is a provenance model of the lifecycle of LoRaWAN uplinks, in which the evidence of gateway reception is an explicit PROV entity, without any change in the devices or in the protocol. Moreover, a tamper-evident storage scheme was adapted to the uplink lifecycle, based on hash chains per device and checkpoints of the chain heads. Last, a method was proposed for the attribution of missing measurements to the stages of the pipeline, where the undetermined cases are reported explicitly.

The practical significance lies in the long-term management of environmental IoT data. An operator is able to see which gateways received an uplink, and how the uplink was decoded and validated. In addition, the operator can check whether the provenance chain is intact and find the stage where a missing measurement was lost. Such functions support audit reports and municipal monitoring, as well as research datasets. Despite this, the cost remains small, since provenance adds 0.62 ms of processing per uplink and about 0.1 MB of storage per device and per day.

The framework has defined limits. The model assumes a trusted platform operator and does not confirm the physical correctness of the sensor values. Moreover, the deployment was small, and the attribution method was evaluated by means of simulation, rather than on field data.

Future work includes four directions. The checkpoints could be stored with an independent party, so that the protection also extends to the platform operator. The attribution method should be applied to a deployment that keeps the event log of the network server. In addition, the device status fields could be used to resolve the

outages that end with a re-join. Finally, the framework should be evaluated on larger networks, with more devices and gateways.

In summary, the results show that provenance can be integrated into LoRaWAN air quality monitoring without changing the devices or the protocol. The framework makes the stored measurements traceable and their history tamper-evident and also makes the gaps in the data explainable.

REFERENCES

1. Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033 [Electronic resource] // Statista. – 2025. – URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (accessed: 15.10.2025).
2. Hashem I. A. T., Chang V., Anuar N. B., Adewole K., Yaqoob I., Gani A., Ahmed E., Chiroma H. The role of big data in smart city // International Journal of Information Management. – 2016. – Vol. 36, No. 5. – P. 748–758.
3. Atzori L., Iera A., Morabito G. The Internet of Things: A survey // Computer Networks. – 2010. – Vol. 54, No. 15. – P. 2787–2805.
4. Kaur B., Dadkhah S., Shoeleh F., Neto E. C. P., Xiong P., Iqbal S., Lamontagne P., Ray S., Ghorbani A. A. Internet of Things (IoT) security dataset evolution: Challenges and future directions // Internet of Things. – 2023. – Vol. 22. – P. 100780.
5. Bhattacharjee S., Salimitari M., Chatterjee M., Kwiat K., Kamhoua C. Preserving Data Integrity in IoT Networks Under Opportunistic Data Manipulation // 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech). – Orlando, FL: IEEE, 2017. – P. 446–453.
6. Zanjani S. M., Shahinzadeh H., Kargar S. M., Moazzami M., Ebrahimi F., Hemmati M. Internet of Things Security: A Review on Challenges, Solutions and Research Directions // 2023 7th International Conference on Internet of Things and Applications (IoT). – Isfahan, Iran, Islamic Republic of: IEEE, 2023. – P. 1–8.
7. Tosh D., Shetty S., Liang X., Kamhoua C., Njilla L. L. Data Provenance in the Cloud: A Blockchain-Based Approach // IEEE Consumer Electron. Mag. – 2019. – Vol. 8, No. 4. – P. 38–44.
8. Ali S., Wang G., Bhuiyan M. Z. A., Jiang H. Secure Data Provenance in Cloud-Centric Internet of Things via Blockchain Smart Contracts // 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). – Guangzhou, China: IEEE, 2018. – P. 991–998.
9. Javaid U., Aman M. N., Sikdar B. BlockPro: Blockchain based Data Provenance and Integrity for Secure IoT Environments // Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems. – Shenzhen China: ACM, 2018. – P. 13–18.
10. Aman M. N., Basheer M. H., Sikdar B. Data Provenance for IoT With Light Weight Authentication and Privacy Preservation // IEEE Internet Things J. – 2019. – Vol. 6, No. 6. – P. 10441–10457.

11. Liu Z., Wu Y. An Index-Based Provenance Compression Scheme for Identifying Malicious Nodes in Multihop IoT Network // IEEE Internet Things J. – 2020. – Vol. 7, No. 5. – P. 4061–4071.

12. Imankulova B., Yafooz W. M. S., Alshammari K., Alshemali B., Yahya A. E., Abd Almisreb A. Tracing Data Origins in Smart Cities: An IoT Perspective // IEEE Access. – 2025. – Vol. 13. – P. 130201–130214.

13. Kenessary D., Kenessary A., Adilgireiuly Z., Akzholova N., Erzhanova A., Dosmukhametov A., Syzdykov D., Masoud A.-R., Saliev T. Air Pollution in Kazakhstan and Its Health Risk Assessment // Annals of Global Health. – 2019. – Vol. 85, No. 1. – P. 133.

14. Kerimray A., Azbanbayev E., Kenessov B., Plotitsyn P., Alimbayeva D., Karaca F. Spatiotemporal Variations and Contributing Factors of Air Pollutants in Almaty, Kazakhstan // Aerosol Air Qual. Res. – 2020. – Vol. 20, No. 6. – P. 1340–1352.

15. Sarsenova Z., Yedilkhan D., Yermekov A., Saleshova S., Amirgaliyev B. Analysis and assessment of air quality in astana: comparison of pollutant levels and their impact on health // Scientific Journal of Astana IT University. – 2024. – P. 98–117.

16. Bissengaliyeva D., Amanzholov T., Rakhimkul A., Yedilkhan D. Determination of the reliability of air pollution measurement data based on vehicular emission recognized as concomitant in Astana // sjaitu. – 2023. – P. 42–51.

17. Beisenova R., Kuanyshevich B. Z., Turlybekova G., Yelikbayev B., Kakabayev A. A., Shamshedenova S., Nugmanov A. Assessment of Atmospheric Air Quality in the Region of Central Kazakhstan and Astana // Atmosphere. – MDPI, 2023. – Vol. 14, No. 11. – P. 1601.

18. Ulman A., Salnikov V., Musralinova G., Tuimebayev Z., Polyakova S., Tazhibayeva T., Kauazov A., Kozhagulov S., Kissebayev D., Raimbekova Z. Air pollution in the West Kazakhstan region: A multi-source analysis // Atmospheric Research. – 2026. – Vol. 330. – P. 108560.

19. Mukhtarov R., Ibragimova O. P., Omarova A., Tursumbayeva M., Tursun K., Muratuly A., Karaca F., Baimatova N. An episode-based assessment for the adverse effects of air mass trajectories on PM2.5 levels in Astana and Almaty, Kazakhstan // Urban Climate. – 2023. – Vol. 49. – P. 101541.

20. Temirbekov N., Temirbekova M., Tamabay D., Kasenov S., Askarov S., Tukenova Z. Assessment of the Negative Impact of Urban Air Pollution on Population Health Using Machine Learning Method // IJERPH. – 2023. – Vol. 20, No. 18. – P. 6770.

21. Zhakiyev N., Khamzina A., Sarkulova Z., Biloshchytskyi A. Air Quality and Environmental Policy in Kazakhstan: Challenges, Innovations, and Pathways to Cleaner Air // Urban Science. – 2025. – Vol. 9, No. 11. – P. 464.

22. Naizabayeva L. Air Pollution Forecasting in Almaty using Ensemble Machine Learning Models // J. Appl. Data Sci. – 2025. – Vol. 6, No. 4. – P. 2461–2476.

23. Naizabayeva L., Zaitov D., Seilova N. Integrating Smart Traffic Systems with Real-Time Air Quality Monitoring to Minimize Emissions and Improve Urban Health. – 2024. – Vol. 251. – P. 603–608.

24. Zhunissova D., Topping D., Evans J. IoT climate resilience in Central Asia: understanding perceived risks through a user survey analysis applied to Kazakhstan // Front. Sustain. Cities. – 2025. – Vol. 7. – P. 1433488.

25. Mukash Z., Bakranova D., Serikkanov A., Bakranov N., Imankulova B. Energy-Aware AI-Enabled Intelligent Sensing Systems for IoT and Edge–Cloud Architectures. – Sensors. – Vol. 26, No. 21. – P. 6927.

26. Zhu L., Yu F. R., Wang Y., Ning B., Tang T. Big Data Analytics in Intelligent Transportation Systems: A Survey // IEEE Trans. Intell. Transport. Syst. – 2019. – Vol. 20, No. 1. – P. 383–398.

27. Roriz Junior M., De Oliveira R. P., Carvalho F., Lifschitz S., Endler M. Mensageria: A Smart City Framework for Real-Time Analysis of Traffic Data Streams // Big Social Data and Urban Computing / ed. Oliveira J., Farias C. M., Pacitti E., Fortino G. – Cham: Springer International Publishing, 2019. – Vol. 926. – P. 59–73.

28. Al-Ali A. R., Zualkernan I. A., Rashid M., Gupta R., Alikarar M. A smart home energy management system using IoT and big data analytics approach // IEEE Trans. Consumer Electron. – 2017. – Vol. 63, No. 4. – P. 426–434.

29. Chia M. H., Keoh S. L., Tang Z. Secure Data Provenance in Home Energy Monitoring Networks // Proceedings of the 3rd Annual Industrial Control System Security Workshop. – San Juan PR USA: ACM, 2017. – P. 7–14.

30. Sembay M. J., De Macedo D. D. J., Júnior L. P., Braga R. M. M., Sarasa-Cabezuelo A. Provenance Data Management in Health Information Systems: A Systematic Literature Review // JPM. – 2023. – Vol. 13, No. 6. – P. 991.

31. Bacco M., Delmastro F., Ferro E., Gotta A. Environmental Monitoring for Smart Cities // IEEE Sensors J. – 2017. – Vol. 17, No. 23. – P. 7767–7774.

32. Lola J., Serrão C., Casal J. Towards Transparent and Secure IoT: Improving the Security and Privacy through a User-Centric Rules-Based System // Electronics. – 2023. – Vol. 12, No. 12. – P. 2589.

33. Mick T., Tourani R., Misra S. LASeR: Lightweight Authentication and Secured Routing for NDN IoT in Smart Cities // IEEE Internet Things J. – 2018. – Vol. 5, No. 2. – P. 755–764.

34. Ahmad R. W., Salah K., Jayaraman R., Yaqoob I., Omar M. Blockchain for Waste Management in Smart Cities: A Survey // IEEE Access. – 2021. – Vol. 9. – P. 131520–131541.

35. Babun L., Denney K., Celik Z. B., McDaniel P., Uluagac A. S. A survey on IoT platforms: Communication, security, and privacy perspectives // Computer Networks. – 2021. – Vol. 192. – P. 108040.

36. Panagou I. C., Katsoulis S., Nannos E., Zantalis F., Koulouras G. A Comprehensive Evaluation of IoT Cloud Platforms: A Feature-Driven Review with a Decision-Making Tool // Sensors. – 2025. – Vol. 25, No. 16. – P. 5124.

37. Zaman M., Puryear N., Abdelwahed S., Zohrabi N. A Review of IoT-Based Smart City Development and Management // Smart Cities. – 2024. – Vol. 7, No. 3. – P. 1462–1501.

38. Raza U., Kulkarni P., Sooriyabandara M. Low Power Wide Area Networks: An Overview // IEEE Commun. Surv. Tutorials. – 2017. – Vol. 19, No. 2. – P. 855–873.

39. Mekki K., Bajic E., Chaxel F., Meyer F. Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT // 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). – Athens: IEEE, 2018. – P. 197–202.

40. Finnegan J., Brown S. A Comparative Survey of LPWA Networking. – arXiv, 2018.

41. Haxhibeqiri J., De Poorter E., Moerman I., Hoebeke J. A Survey of LoRaWAN for IoT: From Technology to Application // Sensors. – 2018. – Vol. 18, No. 11. – P. 3995.

42. Ertürk M. A., Aydın M. A., Büyükakkaşlar M. T., Evirgen H. A Survey on LoRaWAN Architecture, Protocol and Technologies // Future Internet. – 2019. – Vol. 11, No. 10. – P. 216.

43. Alliance L. A technical overview of LoRa and LoRaWAN // White Paper, November. – 2015. – Vol. 20.

44. Semtech. SX1272/3/6/7/8: LoRa Modem Design Guide [Electronic resource]. – 2013. – URL: https://www.mouser.com/pdfdocs/semtech-lora-modem-design.pdf (accessed: 15.05.2025).

45. Zorbas D., Papadopoulos G. Z., Maille P., Montavont N., Douligeris C. Improving LoRa Network Capacity Using Multiple Spreading Factor Configurations // 2018 25th International Conference on Telecommunications (ICT). – St. Malo: IEEE, 2018. – P. 516–520.

46. lora-alliance. LoRaWAN® Specification v1.0.1 [Electronic resource]. – 2015. – URL: https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-0-1 (accessed: 26.05.2025).

47. Acosta-Garcia L., Aznar-Poveda J., Garcia-Sanchez A. J., Garcia-Haro J., Fahringer T. Dynamic transmission policy for enhancing LoRa network performance: A deep reinforcement learning approach // Internet of Things. – 2023. – Vol. 24. – P. 100974.

48. Semtech. LoRa® and LoRaWAN® [Electronic resource]. – 2024. – URL: https://www.semtech.com/uploads/technology/LoRa/lora-and-lorawan.pdf (accessed: 26.07.2025).

49. LoRa Alliance. LoRaWAN® L2 1.0.4 Specification [Electronic resource]. – LoRa Alliance, 2023. – URL: https://resources.lora-alliance.org/document/ts001-1-0-4-lorawan-l2-1-0-4-specification (accessed: 20.01.2026).

50. LoRaWAN Regional Parameters [Electronic resource]. – LoRa Alliance, 2015. – URL: https://lora-alliance.org/wp-content/uploads/2020/11/lorawan_regional_parameters_v1_0-20161012_1397_1.pdf (accessed: 23.07.2026).

51. Butun I., Pereira N., Gidlund M. Analysis of LoRaWAN v1.1 security: research paper // Proceedings of the 4th ACM MobiHoc Workshop on Experiences with the Design and Implementation of Smart Objects. – Los Angeles California: ACM, 2018. – P. 1–6.

52. Gemalto, Actility, Semtech. A white paper prepared for the LoRa Alliance™ [Electronic resource]. – 2017. – URL: https://lora-alliance.org/wp-content/uploads/2020/11/lorawan_security_whitepaper.pdf (accessed: 20.07.2025).

53. Basford P. J., Bulot F. M. J., Apetroaie-Cristea M., Cox S. J., Ossont S. J. LoRaWAN for Smart City IoT Deployments: A Long Term Evaluation // Sensors. – 2020. – Vol. 20, No. 3. – P. 648.

54. Döníz B., Lajos B. Challenges of LoRaWAN technology in smart city solutions // Interdisciplinary Description of Complex Systems. – 2022. – Vol. 20, No. 1. – P. 1–10.

55. PROV-DM: The PROV Data Model.: W3C Recommendation / ed. Moreau L., Missier P. – World Wide Web Consortium, 2013.

56. Teh H. Y., Kempa-Liehr A. W., Wang K. I.-K. Sensor data quality: a systematic review // J Big Data. – 2020. – Vol. 7, No. 1. – P. 11.

57. Buelvas J., Múnera D., Tobón V D. P., Aguirre J., Gaviria N. Data Quality in IoT-Based Air Quality Monitoring Systems: a Systematic Mapping Study // Water, Air, and Soil Pollution. – 2023. – Vol. 234, No. 4.

58. Simmhan Y. L., Plale B., Gannon D. A survey of data provenance in e-science // SIGMOD Rec. – 2005. – Vol. 34, No. 3. – P. 31–36.

59. Cheney J., Chiticariu L., Tan W.-C. Provenance in Databases: Why, How, and Where // Foundations and Trends in Databases. – 2009. – Vol. 1, No. 4. – P. 379–474.

60. Muniswamy-Reddy K.-K., Holland D. A., Braun U., Seltzer M. Provenance-aware storage systems // Proceedings of the Annual Conference on USENIX ’06 Annual Technical Conference. – USENIX Association, 2006. – P. 43–56.

61. Hu R., Yan Z., Ding W., Yang L. T. A survey on data provenance in IoT // World Wide Web. – 2020. – Vol. 23, No. 2. – P. 1441–1463.

62. Alam M. M., Wang W. A comprehensive survey on data provenance: State-of-the-art approaches and their deployments for IoT security enforcement // Journal of Computer Security. – 2021. – Vol. 29, No. 4. – P. 423–446.

63. Elkhodr M., Alsinglawi B., Alshehri M. Data Provenance in the Internet of Things // 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA). – Krakow: IEEE, 2018. – P. 727–731.

64. Singh J., Cobbe J., Norval C. Decision Provenance: Harnessing Data Flow for Accountable Systems // IEEE Access. – 2019. – Vol. 7. – P. 6562–6574.

65. Moreau L., Clifford B., Freire J., Futrelle J., Gil Y., Groth P., Kwasnikowska N., Miles S., Missier P., Myers J., Plale B., Simmhan Y., Stephan E., Den Bussche J. V. The Open Provenance Model core specification (v1.1) // Future Generation Computer Systems. – 2011. – Vol. 27, No. 6. – P. 743–756.

66. Lebo T., Sahoo S., McGuinness D., Belhajjame K., Cheney J., Corsar D., Garijo D., Siland-Reyes S. PROV-O: The PROV Ontology. (W3C Recommendation)

[Electronic resource]. – 2013. – URL: https://www.w3.org/TR/2013/REC-prov-o-20130430/.

67. Page M. J., McKenzie J. E., Bossuyt P. M., Boutron I., Hoffmann T. C., Mulrow C. D., Shamseer L., Tetzlaff J. M., Akl E. A., Brennan S. E., Chou R., Glanville J., Grimshaw J. M., Hróbjartsson A., Lalu M. M., Li T., Loder E. W., Mayo-Wilson E., McDonald S., McGuinness L. A., Stewart L. A., Thomas J., Tricco A. C., Welch V. A., Whiting P., Moher D. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews // BMJ. – 2021. – P. n71.

68. Sanghera R., Thirunavukarasu A. J., El Khoury M., O'Logbon J., Chen Y., Watt A., Mahmood M., Butt H., Nishimura G., Soltan A. A. S. High-performance automated abstract screening with large language model ensembles // Journal of the American Medical Informatics Association. – 2025. – Vol. 32, No. 5. – P. 893–904.

69. Xu Q., Akhtar R., Zhang X., Wang C. Cluster-Based Arithmetic Coding for Data Provenance Compression in Wireless Sensor Networks // Wireless Communications and Mobile Computing. – 2018. – Vol. 2018.

70. Xu Q., Wang C. Stepwise Refinement Provenance Scheme for Wireless Sensor Networks // IEEE Internet of Things Journal. – 2022. – Vol. 9, No. 13. – P. 11126–11140.

71. Faraj O., Megías D., Garcia-Alfaro J. ZIRCON: Zero-watermarking-based approach for data integrity and secure provenance in IoT networks // Journal of Information Security and Applications. – 2024. – Vol. 85. – P. 103840.

72. Sultana S., Ghinita G., Bertino E., Shehab M. A Lightweight Secure Scheme for Detecting Provenance Forgery and Packet DropAttacks in Wireless Sensor Networks // IEEE Trans. Dependable and Secure Comput. – 2015. – Vol. 12, No. 3. – P. 256–269.

73. Siddiqui M. S., Rahman A., Nadeem A. Secure Data Provenance in IoT Network using Bloom Filters // Procedia Computer Science. – 2019. – Vol. 163. – P. 190–197.

74. Kamal M., Tariq M. Light-Weight Security and Data Provenance for Multi-Hop Internet of Things // IEEE Access. – 2018. – Vol. 6. – P. 34439–34448.

75. Suhail S., Hong C. S., Lodhi M. A., Zafar F., Khan A., Bashir F. Data trustworthiness in IoT. – 2018. – Vol. 2018-January. – P. 414–419.

76. Zeng Y., Zhang X., Akhtar R., Wang C. A Blockchain-Based Scheme for Secure Data Provenance in Wireless Sensor Networks. – 2018. – P. 13–18.

77. Aman M. N., Basheer M. H., Sikdar B. Data Provenance for IoT using Wireless Channel Characteristics and Physically Unclonable Functions. – 2019. – Vol. 2019-May.

78. Aman M. N., Basheer M. H., Sikdar B. A Lightweight Protocol for Secure Data Provenance in the Internet of Things Using Wireless Fingerprints // IEEE Systems Journal. – 2021. – Vol. 15, No. 2. – P. 2948–2958.

79. Baig M. A., Aman M. N. Data Provenance for IoT Devices by Exploiting Clock Variations // 2023 IEEE Conference on Dependable and Secure Computing (DSC). – Tampa, FL, USA: IEEE, 2023. – P. 1–8.

80. Hamadeh H., Tyagi A. An FPGA Implementation of Privacy Preserving Data Provenance Model Based on PUF for Secure Internet of Things // SN Computer Science. – 2021. – Vol. 2, No. 2.

81. Canovas Sanchez J. L., Bernabe J. B., Skarmeta A. F. Towards privacy preserving data provenance for the Internet of Things // 2018 IEEE 4th World Forum on Internet of Things (WF-IoT). – Singapore: IEEE, 2018. – P. 41–46.

82. Javed K., Ali Khan M., Ullah M., Naveed Aman M., Sikdar B. Securing Synchrophasors Using Data Provenance in the Quantum Era // IEEE Open J. Commun. Soc. – 2024. – Vol. 5. – P. 1594–1608.

83. Tunstad P., Khan A. M., Ha P. H. HyperProv: Decentralized resilient data provenance at the edge with blockchains. – 2019. – P. 3–4.

84. Honar Pajooh H., Rashid M., Alam F., Demidenko S. Hyperledger Fabric Blockchain for Securing the Edge Internet of Things // Sensors (Basel, Switzerland). – 2021. – Vol. 21, No. 2.

85. Ioini N. E., Pahl C. Trustworthy orchestration of container based edge computing using permissioned blockchain. – 2018. – P. 147–154.

86. Lautert F., Pigatto D. F., Gomes L. A fog architecture for privacy-preserving data provenance using blockchains. – 2020. – Vol. 2020-July.

87. Cui H., Chen Z., Xi Y., Chen H., Hao J. IoT data management and lineage traceability: A blockchain-based solution. – 2019. – P. 239–244.

88. Doubiz M., Banane M., Zakrani A., Erraissi A. Auditable Real-Time Cold-Chain Monitoring with IoT and Blockchain Anchoring // International Journal of Advanced Computer Science and Applications. – 2025. – Vol. 16, No. 10. – P. 1073–1082.

89. Palyvos-Giannas D., Gulisano V., Papatriantafilou M. GeneaLog: Fine-grained data streaming provenance in cyber-physical systems // Parallel Computing. – 2019. – Vol. 89.

90. Gordani Shahri M., Erlandsson A., Palyvos-Giannas D., Gulisano V. Poster: Twins, a Middleware for Adaptive Streaming Provenance at the Edge. – 2021. – P. 235–236.

91. Moravskyi R., Levus Y. Development of an adaptive data provenance control method for resource-aware real-time stream processing // Technology Audit and Production Reserves. – 2026. – Vol. 3, No. 2. – P. 60–65.

92. Markovic M., Edwards P. Enhancing transparency of MQTT brokers for IoT applications through provenance streams. – 2019. – P. 17–20.

93. Baddour L. S., Abdiwi F. G., Mahmoud S. M., Abu-Alshaeer M. J., Alwan G. M. Blockchain Protocols for Data Provenance and Energy Accountability in IoT-Based Energy Networks // International Journal on Informatics Visualization. – 2026. – Vol. 10, No. 2. – P. 777–786.

94. Yan T., Chen J., Zhang X., Hu G. High-Efficiency Traceability Mechanism for Multimedia Data in Consumer Internet of Things Combined with Blockchain // Sensors. – 2026. – Vol. 26, No. 1.

95. Wang Q., Hassan W. U., Bates A., Gunter C. Fear and Logging in the Internet of Things // Proceedings 2018 Network and Distributed System Security Symposium. – San Diego, CA: Internet Society, 2018.

96. Khodayarseresht E., Smolyakova S., Zhao L., Mansouri A., Majumdar S., Conti M. ForenThings: An Interactive Framework for Crime Scene Reconstruction in IoT Forensics // ACM Transactions on Internet of Things. – 2026. – Vol. 7, No. 1.

97. Da Silva D. L., Batista A., Corrêa P. L. P. Data Provenance in Environmental Monitoring. – 2017. – P. 337–342.

98. Serra Da Cruz S. M., Macário Costa R. J. Enabling Smart City Provenance-Based Applications to Improve Urban Mobility in Brazilian Cities // Sustainable Transportation and Smart Logistics. – Elsevier, 2019. – P. 439–466.

99. Sirqueira T., Viana M., Nascimento N., Lucena C. A software framework for data provenance. – 2017. – P. 615–618.

100. Liu Y., Futrelle J., Myers J., Rodriguez A., Kooper R. A provenance-aware virtual sensor system using the Open Provenance Model. – 2010. – P. 330–339.

101. Aktas M. S., Astekin M. Provenance aware run-time verification of things for self-healing Internet of Things applications. – 2019. – Vol. 31, No. 3.

102. Sadineni L., Pilli E. S., Battula R. B. Ready-IoT: A Novel Forensic Readiness Model for Internet of Things. – 2021. – P. 89–94.

103. Sadineni L., Pilli E. S., Battula R. B. ProvNet-IoT: Provenance based network layer forensics in Internet of Things // Forensic Science International: Digital Investigation. – 2022. – Vol. 43. – P. 301441.

104. Liu D., Ni J., Huang C., Lin X., Shen X. S. Secure and Efficient Distributed Network Provenance for IoT: A Blockchain-Based Approach // IEEE Internet of Things Journal. – 2020. – Vol. 7, No. 8. – P. 7564–7574.

105. Rong-na X., Hui L., Guo-zhen S., Yun-chuan G., Ben N., Mang S. Provenance-based data flow control mechanism for Internet of things // Trans Emerging Tel Tech. – 2021. – Vol. 32, No. 5. – P. e3934.

106. Rao Q., Wu J., Chen S., Pan Z., Lei Q., Liu Y., Feng Y., Jia X. A Metadata-Driven Architecture for Federated Data Asset Management and Visualization in Energy Monitoring Networks // Journal of Web Engineering. – 2026. – Vol. 25, No. 2. – P. 153–186.

107. Markovic M., Jacobs N., Dryja K., Edwards P., Strachan N. J. C. Integrating Internet of Things, Provenance, and Blockchain to Enhance Trust in Last Mile Food Deliveries // Frontiers in Sustainable Food Systems. – 2020. – Vol. 4.

108. Jaigirdar F. T., Tan B., Rudolph C., Bain C. Security-Aware Provenance for Transparency in IoT Data Propagation // IEEE Access. – 2023. – Vol. 11. – P. 55677–55691.

109. Rosendo D., Mattoso M., Costan A., Souza R., Pina D., Valduriez P., Antoniu G. ProvLight: Efficient Workflow Provenance Capture on the Edge-to-Cloud Continuum // 2023 IEEE International Conference on Cluster Computing (CLUSTER). – Santa Fe, NM, USA: IEEE, 2023. – P. 221–233.

110. Qasha R., Solayman H., Rashid S. Provenance-driven Versioning and Deployment Strategies for IoT System in the Cloud Continuum // International Journal of Computing and Digital Systems. – 2026. – Vol. 19, No. 1.

111. Chang Y.-S., Huang H.-C. A Blockchain-Assisted BIM–IoT Digital Twin Architecture for Trusted Operational Risk Prediction in Smart Buildings // Computers, Materials and Continua. – 2026. – Vol. 89, No. 1.

112. Werapun W., Werapun W. WindChain: Physics-Constrained Blockchain Attestation of Wind-Resource Provenance for Verifiable Renewable Energy Certificates in Smart Cities // Smart Cities. – 2026. – Vol. 9, No. 9.

113. Wahsheh H. A. M. Securing Blue Carbon Accounting: A Cryptographic Framework for Coastal Ecosystem Monitoring // International Journal of Advanced Computer Science and Applications. – 2026. – Vol. 17, No. 2. – P. 700–707.

114. Gawande K., Stantchev V. From Sealed to Anchored: What a Consortium Ledger Adds to HMAC-Sealed IoT Audit Logs in Ambient Assisted Living // Applied Sciences (Switzerland). – 2026. – Vol. 16, No. 17.

115. Crosby S. A., Wallach D. S. Efficient data structures for tamper-evident logging. – 2009. – P. 317–333.

116. Antonopoulos P., Kaushik R., Kodavalla H., Rosales Aceves S., Wong R., Anderson J., Szymaszek J. SQL Ledger: Cryptographically Verifiable Data in Azure SQL Database // Proceedings of the 2021 International Conference on Management of Data. – Virtual Event China: ACM, 2021. – P. 2437–2449.

117. Yang X., Wang S., Li F., Zhang Y., Yan W., Gai F., Yu B., Feng L., Gao Q., Li Y. Ubiquitous Verification in Centralized Ledger Database // 2022 IEEE 38th International Conference on Data Engineering (ICDE). – Kuala Lumpur, Malaysia: IEEE, 2022. – P. 1808–1821.

118. Yang X., Zhang R., Yue C., Liu Y., Ooi B., Gao Q., Zhang Y., Yang H. VeDB: A Software and Hardware Enabled Trusted Relational Database // PROCEEDINGS OF THE ACM ON MANAGEMENT OF DATA. – 2023. – Vol. 1, No. 2.

119. Dhingra D., Gandhi A. B., Sigh S., Kumar I., Tyagi S. Enhancing Trust in Digital Event Marketplaces Using Novel Verifiable Event Lifecycle Protocol. – 2025. – P. 780–786.

120. Ma D., Tsudik G. A new approach to secure logging // ACM Trans. Storage. – 2009. – Vol. 5, No. 1. – P. 1–21.

121. Marson G. A., Poettering B. Even More Practical Secure Logging: Tree-Based Seekable Sequential Key Generators // Computer Security - ESORICS 2014 / ed. Kutyłowski M., Vaidya J. – Cham: Springer International Publishing, 2014. – Vol. 8713. – P. 37–54.

122. Siddiqi M., Ali S. T., Sivaraman V. Secure Opportunistic Contextual Logging for Wearable Healthcare Sensing Devices // IEEE Transactions on Dependable and Secure Computing. – 2021. – Vol. 18, No. 2. – P. 753–764.

123. Chatzigiannis P., Baldimtsi F., Kolias C., Stavrou A. Black-Box IoT: Authentication and Distributed Storage of IoT Data from Constrained Sensors. – 2021. – P. 1–14.

124. Ning F., Wen Y., Shi G., Meng D. Efficient tamper-evident logging of distributed systems via concurrent authenticated tree. – 2017. – Vol. 2018-January. – P. 1–9.

125. El-Hindi M., Ziegler T., Binnig C. Towards Merkle Trees for High-Performance Data Systems. – 2023. – P. 28–33.

126. Fang V., Dauterman E., Ravoor A., Dewan A., Popa R. A. LegoLog: A configurable transparency log. – 2025. – P. 1082–1103.

127. Yang X., Yue C., Zhang W., Liu Y., Ooi B., Chen J. SecuDB: An In-enclave Privacy-preserving and Tamper-resistant Relational Database // PROCEEDINGS OF THE VLDB ENDOWMENT. – 2024. – Vol. 17, No. 12. – P. 3906–3919.

128. Zhou W., Cai Y., Peng Y., Wang S., Ma K., Li F. VeriDB: An SGX-based Verifiable Database. – 2021. – P. 2182–2194.

129. Yue C., Zhang M., Zhu C., Chen G., Loghin D., Ooi B. C. VeriBench: Analyzing the Performance of Database Systems with Verifiability // Proc. VLDB Endow. – 2023. – Vol. 16, No. 9. – P. 2145–2157.

130. Lupaiescu S., Cioata P., Turcu C. E., Gherman O., Turcu C. O., Paslaru G. Centralized vs. Decentralized: Performance Comparison between BigchainDB and Amazon QLDB // Applied Sciences (Switzerland). – 2023. – Vol. 13, No. 1.

131. Fekete D. L., Kiss A. A Survey of Ledger Technology-Based Databases // Future Internet. – 2021. – Vol. 13, No. 8. – P. 197.

132. Nash T., Olmsted A. Performance vs. security: Implementing an immutable database in MySQL. – 2018. – P. 290–291.

133. Sey C., Lei H., Qian W., Li X., Fiasam L. D., Kodjiku S. L., Adjei-Mensah I., Agyemang I. O. VBlock: A Blockchain-Based Tamper-Proofing Data Protection Model for Internet of Vehicle Networks // Sensors. – 2022. – Vol. 22, No. 20. – P. 8083.

134. Honar Pajooh H., Rashid M. A., Alam F., Demidenko S. IoT Big Data provenance scheme using blockchain on Hadoop ecosystem // Journal of Big Data. – 2021. – Vol. 8, No. 1.

135. Sukumaran S., Korath A., Menon G. A. Tamper-Evident Data and Model Provenance for IoT-Based Machine Learning Using Blockchain and Off-Chain Storage // Information (Switzerland). – 2026. – Vol. 17, No. 5.

136. Sifah E. B., Xia Q., Agyekum K. O.-B. O., Xia H., Smahi A., Gao J. A Blockchain Approach to Ensuring Provenance to Outsourced Cloud Data in a Sharing Ecosystem // IEEE Systems Journal. – 2022. – Vol. 16, No. 1. – P. 1673–1684.

137. Sigwart M., Borkowski M., Peise M., Schulte S., Tai S. A secure and extensible blockchain-based data provenance framework for the Internet of Things // Pers Ubiquit Comput. – 2024. – Vol. 28, No. 1. – P. 309–323.

138. Orrù D., Pinna A., Tonelli R. Low-Cost Tamper-Proof IoT Devices to Improve Data Origin Verification and Privacy in Blockchain-Based Energy Consumption Records. – 2024. – Vol. 3791.

139. Siddiqui M. S., Syed T. A., Nadeem A., Nawaz W., Albouq S. S. BlockTrack-L: A lightweight blockchain-based provenance message tracking in IoT // International Journal of Advanced Computer Science and Applications. – 2020. – Vol. 11, No. 4. – P. 463–470.

140. Mercan S., Cebe M., Tekiner E., Akkaya K., Chang M., Uluagac S. A Cost-efficient IoT Forensics Framework with Blockchain. – 2020.

141. Abouelkheir E. A Provenance-Driven Trust Framework with Physics-Consistent Validation for Secure Wireless Sensor Networks // Sensors. – 2026. – Vol. 26, No. 18.

142. Wang C., Yang L., Guo H., Wan F. Data protection and provenance in cloud of things environment: Research challenges // International Journal of Information and Computer Security. – 2020. – Vol. 12, No. 4. – P. 416–435.

143. Ebi C., Schaltegger F., Rust A., Blumensaat F. Synchronous LoRa Mesh Network to Monitor Processes in Underground Infrastructure // IEEE Access. – 2019. – Vol. 7. – P. 57663–57677.

144. Cappelli I., Peruzzi G., Pozzebon A., Scarpel E. Wireless Sensing in the Woodlands: Preliminary Tests for LoRaWAN Transmission in Vegetated Areas. – 2023.

145. Yavuz Ö. Ç. Privileged communication and coordination network for managerial decision support in disaster and emergency scenarios // Social Sciences and Humanities Open. – 2026. – Vol. 14.

146. Klier M., Moestue L., Obermeier A., Widmann T. Assessing Completeness of IoT Data: A Novel Probabilistic Approach // Bus Inf Syst Eng. – 2025. – Vol. 67, No. 6. – P. 797–814.

147. Buelvas J. H., Múnera D., Gaviria N. DQ-MAN: A tool for multi-dimensional data quality analysis in IoT-based air quality monitoring systems // Internet of Things (Netherlands). – 2023. – Vol. 22.

148. Ge C., Zhou L., Hancke G. P., Su C. A Provenance-Aware Distributed Trust Model for Resilient Unmanned Aerial Vehicle Networks // IEEE Internet of Things Journal. – 2021. – Vol. 8, No. 16. – P. 12481–12489.

149. Silva D., Heideker A., Zyrianoff I., Kleinschmidt J., Roffia L., Soininen J., Kamienski C. A Management Architecture for IoT Smart Solutions: Design and Implementation // JOURNAL OF NETWORK AND SYSTEMS MANAGEMENT. – 2022. – Vol. 30, No. 2.

150. ChirpStack, open-source LoRaWAN® Network Server [Electronic resource]. – URL: https://www.chirpstack.io/ (accessed: 23.07.2025).

151. Haerder T., Reuter A. Principles of transaction-oriented database recovery // ACM Comput. Surv. – 1983. – Vol. 15, No. 4. – P. 287–317.

152. Dolev D., Yao A. On the security of public key protocols. – IEEE Transactions on information theory, 1983. – Vol. 29, No. 2. – P. 198–208.

153. Kelsey J., Schneier B. Cryptographic Support for Secure Logs on Untrusted Machines // The Seventh USENIX Security Symposium Proceedings. – USENIX Press, 1998. – P. 53–62.

154. Imankulova B. Privacy challenges and framework design for IoT devices: A comprehensive approach // Bulletin of AUPET. – 2025. – Vol. 1, No. 68.

155. Imankulova B., Sagymbekova A. Privacy Challenges in Environmental IoT Systems: A Case Study on Air Quality Monitoring. – Astana, Kazakhstan: IEEE, 2025. – P. 1–6.

156. National Institute of Standards and Technology (US). Secure hash standard: NIST FIPS 180-4. – Washington, D.C.: National Institute of Standards and Technology (U.S.), 2015. – P. NIST FIPS 180-4.

157. Pipino L. L., Lee Y. W., Wang R. Y. Data quality assessment // Commun. ACM. – 2002. – Vol. 45, No. 4. – P. 211–218.

158. Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Data quality model: ISO/IEC 25012:2008. – International Organization for Standardization; International Electrotechnical Commission, 2008.

159. Bor M. C., Roedig U., Voigt T., Alonso J. M. Do LoRa Low-Power Wide-Area Networks Scale? // Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems. – Malta Malta: ACM, 2016. – P. 59–67.

160. Rundgren A., Jordan B., Erdtman S. JSON Canonicalization Scheme (JCS): RFC 8785. – Internet Engineering Task Force (IETF), 2020.

161. Haber S., Stornetta W. S. How to time-stamp a digital document // J. Cryptology. – 1991. – Vol. 3, No. 2. – P. 99–111.

162. Krawczyk H., Bellare M., Canetti R. HMAC: Keyed-Hashing for Message Authentication: RFC 2104.

163. Bellare M., Canetti R., Krawczyk H. Keying Hash Functions for Message Authentication // Advances in Cryptology — CRYPTO '96 / ed. Koblitz N. – Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. – Vol. 1109. – P. 1–15.

164. Gardner E. S. Exponential smoothing: The state of the art // Journal of Forecasting. – 1985. – Vol. 4, No. 1. – P. 1–28.

165. Chow C. On optimum recognition error and reject tradeoff // IEEE Trans. Inform. Theory. – 1970. – Vol. 16, No. 1. – P. 41–46.

166. Banks A., Briggs E., Borgendale K., Gupta R. MQTT Version 5.0. [Electronic resource]. – OASIS Standard, 2019. – URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed: 20.01.2026).

167. Imankulova B. B., Amanzholova S. T., Duisebekova K. S. Software and Technical Aspects of Data Collection From End Nodes: Problems and Prospects // AUPET Bulletin. – 2020. – No. 4. – P. 164–172.

168. Imankulova B., Kozina L., Duzbayev N., Amanzholova S. Transmission, storage and preprocessing of data received from sensors using LoRa technology // AUPET Bulletin. – 2024. – Vol. 1, No. 64. – P. 16–24.

169. Krawczyk H., Bellare M., Canetti R. HMAC: Keyed-Hashing for Message Authentication: RFC2104. – RFC Editor, 1997. – P. RFC2104.

170. Fizza K., Jayaraman P. P., Banerjee A., Georgakopoulos D., Ranjan R. Evaluating Sensor Data Quality in Internet of Things Smart Agriculture Applications // IEEE Micro. – 2022. – Vol. 42, No. 1. – P. 51–60.

171. Magrin D., Centenaro M., Vangelista L. Performance evaluation of LoRa networks in a smart city scenario // 2017 IEEE International Conference on Communications (ICC). – Paris, France: IEEE, 2017. – P. 1–7.

APPENDIX A
Search queries in Scopus and Web of Science databases

Table A.1 - Search queries in Scopus

Block	Topic	Query	Records
1	2	3	4
B1	Data provenance in IoT and smart city systems	TITLE-ABS-KEY("data provenance" OR "provenance data" OR "data lineage" OR "provenance tracking" OR "provenance capture" OR "provenance model*" OR "secure provenance" OR "provenance-based" OR "provenance-aware") AND TITLE-ABS-KEY("Internet of Things" OR "IoT" OR "sensor network*" OR "smart cit*" OR "cyber-physical" OR "edge computing" OR "fog computing") AND PUBYEAR > 2017 AND PUBYEAR < 2027 AND DOCTYPE(ar OR cp OR re) AND LANGUAGE(english)	313
B2	Secure and tamper-evident logging	TITLE-ABS-KEY (("tamper-evident" OR "tamper-proof" OR "tamper-resistant" OR "forward-secure" OR "append-only" OR "secure" OR "verifiable") W/3 (log OR logs OR logging OR "audit trail*" OR "audit log*")) AND TITLE-ABS-KEY ("hash chain*" OR "hash-chain*" OR "Merkle" OR "HMAC" OR "message authentication code*" OR "authenticated data structure*") AND DOCTYPE (ar OR cp) AND LANGUAGE (english) AND SUBJAREA (COMP OR ENGI)	68
B3	Ledger databases and verifiable storage	TITLE-ABS-KEY ("ledger database*" OR "verifiable database*" OR "immutable database*" OR "tamper-evident database*" OR "authenticated database*" OR "verifiable ledger*" OR "transparency log*") AND PUBYEAR > 2014 AND DOCTYPE (ar OR cp) AND LANGUAGE (english) AND NOT TITLE-ABS-KEY (cryptocurrenc* OR bitcoin OR "smart contract*" OR "certificate transparency") AND TITLE-ABS-KEY (audit* OR provenance OR "data integrity" OR "IoT" OR "Internet of Things" OR sensor* OR performance OR benchmark*)	80
B4	Provenance integrity protection	TITLE-ABS-KEY ("data provenance" OR "data lineage" OR "sensor data provenance" OR "provenance tracking" OR "secure provenance") AND TITLE-ABS-KEY ("data integrity" OR "tamper-evident" OR "integrity verification" OR "hash chain*" OR "provenance verification" OR "provenance integrity" OR "Merkle" OR "digital signature*") AND TITLE-ABS-KEY ("Internet of Things" OR IoT OR "wireless sensor network*" OR LoRaWAN) AND PUBYEAR > 2017 AND (DOCTYPE (ar) OR DOCTYPE (cp)) AND LANGUAGE (english)	54
B5	W3C PROV for sensor data	TITLE-ABS-KEY ("W3C PROV" OR "PROV-O" OR "PROV-DM" OR "PROV model" OR "provenance graph*") AND TITLE-ABS-KEY ("IoT" OR "Internet of Things" OR "sensor*" OR "environmental monitoring" OR "smart cit*") AND DOCTYPE (ar OR cp) AND LANGUAGE (english)	43
B6	Packet loss in LoRaWAN	TITLE-ABS ("LoRaWAN") AND TITLE-ABS ("packet loss" OR "packet delivery ratio" OR "packet error rate" OR "frame loss" OR	13

	deployments	"frame counter" OR "data completeness" OR "missing data" OR "data loss") AND TITLE-ABS ("field deployment*" OR "field test*" OR "field trial*" OR "real-world deployment*" OR "outdoor deployment*" OR "real deployment*" OR "long-term deployment*" OR "testbed") AND PUBYEAR > 2017 AND DOCTYPE (ar OR cp) AND LANGUAGE (english) AND TITLE-ABS ("environmental monitoring" OR "air quality" OR "air pollution" OR "smart cit*" OR "urban monitoring" OR "sensor network*")	
B7	Diagnosis of missing and faulty sensor data	TITLE-ABS ("data completeness" OR "missing data" OR "data loss" OR "data gap*" OR "packet loss") AND TITLE-ABS ("root cause*" OR "fault diagnosis" OR "failure diagnosis" OR "fault detection" OR "fault localization" OR "failure analysis" OR "cause analysis") AND TITLE-ABS ("IoT" OR "Internet of Things" OR "sensor network*" OR "air quality monitor*" OR "LPWAN" OR "LoRaWAN") AND NOT TITLE-ABS (patient* OR clinical OR medical OR healthcare OR "health care" OR hospital*) AND PUBYEAR > 2017 AND DOCTYPE (ar OR cp) AND LANGUAGE (english)	74
		Total	645

Table A.2 - Search queries in Web of Science Core Collection

Block	Topic	Query	Records
1	2	3	4
B1	Data provenance in IoT and smart city systems	TS=("data provenance" OR "provenance data" OR "data lineage" OR "provenance tracking" OR "provenance capture" OR "provenance model*" OR "secure provenance" OR "provenance-based" OR "provenance-aware") AND TS=("Internet of Things" OR "IoT" OR "sensor network*" OR "smart cit*" OR "cyber-physical" OR "edge computing" OR "fog computing"); refined by PY=(2018–2026), DT=(Article OR Proceedings Paper OR Review), LA=(English)	220
B2	Secure and tamper-evident logging	TS=(("tamper-evident" OR "tamper-proof" OR "tamper-resistant" OR "forward-secure" OR "append-only" OR "secure" OR "verifiable") NEAR/3 (log OR logs OR logging OR "audit trail*" OR "audit log*")) AND TS=("hash chain*" OR "hash-chain*" OR "Merkle" OR "HMAC" OR "message authentication code*" OR "authenticated data structure*") AND DT=(Article OR Proceedings Paper) AND LA=(English)	42
B3	Ledger databases and verifiable storage	(TS=("ledger database*" OR "verifiable database*" OR "immutable database*" OR "tamper-evident database*" OR "authenticated database*" OR "verifiable ledger*" OR "transparency log*") NOT TS=(cryptocurrenc* OR bitcoin OR "smart contract*" OR "certificate transparency")) AND PY=(2015-2026) AND DT=(Article OR Proceedings Paper) AND LA=(English)	71
B4	Provenance integrity	TS=("data provenance" OR "data lineage" OR "sensor data provenance" OR "provenance tracking" OR	38

	protection	"secure provenance") AND TS=("data integrity" OR "tamper-evident" OR "integrity verification" OR "hash chain*" OR "provenance verification" OR "provenance integrity" OR "Merkle" OR "digital signature*") AND TS=("Internet of Things" OR IoT OR "wireless sensor network*" OR LoRaWAN) AND PY=(2018-2026) AND DT=(Article OR Proceedings Paper) AND LA=(English)	
B5	W3C PROV for sensor data	TS=("W3C PROV" OR "PROV-O" OR "PROV-DM" OR "PROV model" OR "provenance graph*") AND TS=("IoT" OR "Internet of Things" OR "sensor*" OR "environmental monitoring" OR "smart cit*") AND DT=(Article OR Proceedings Paper) AND LA=(English)	34
B6	Packet loss in LoRaWAN deployments	(TI=("LoRaWAN") OR AB=("LoRaWAN")) AND (TI=("packet loss" OR "packet delivery ratio" OR "packet error rate" OR "frame loss" OR "frame counter" OR "data completeness" OR "missing data" OR "data loss") OR AB=("packet loss" OR "packet delivery ratio" OR "packet error rate" OR "frame loss" OR "frame counter" OR "data completeness" OR "missing data" OR "data loss")) AND (TI=("field deployment*" OR "field test*" OR "field trial*" OR "real-world deployment*" OR "outdoor deployment*" OR "real deployment*" OR "long-term deployment*" OR "testbed") OR AB=("field deployment*" OR "field test*" OR "field trial*" OR "real-world deployment*" OR "outdoor deployment*" OR "real deployment*" OR "long-term deployment*" OR "testbed")) AND PY=(2018-2026) AND DT=(Article OR Proceedings Paper) AND LA=(English) AND (TI=("environmental monitoring" OR "air quality" OR "air pollution" OR "smart cit*" OR "urban monitoring" OR "sensor network*") OR AB=("environmental monitoring" OR "air quality" OR "air pollution" OR "smart cit*" OR "urban monitoring" OR "sensor network*"))	8
B7	Diagnosis of missing and faulty sensor data	(TI=("data completeness" OR "missing data" OR "data loss" OR "data gap*" OR "packet loss") OR AB=("data quality" OR "data completeness" OR "missing data" OR "data loss" OR "data gap*" OR "packet loss")) AND (TI=("root cause*" OR "fault diagnosis" OR "failure diagnosis" OR "fault detection" OR "fault localization" OR "failure analysis" OR "cause analysis") OR AB=("root cause*" OR "fault diagnosis" OR "failure diagnosis" OR "fault detection" OR "fault localization" OR "failure analysis" OR "cause analysis" OR "anomaly detection")) AND (TI=("IoT" OR "Internet of Things" OR "sensor network*" OR "air quality monitor*" OR "LPWAN" OR "LoRaWAN") OR AB=("IoT" OR "Internet of Things" OR "sensor network*" OR "air quality monitor*" OR "LPWAN" OR "LoRaWAN")) NOT TS=(patient* OR clinical OR medical OR healthcare OR "health care" OR hospital*) AND PY=(2018-2026) AND DT=(Article OR Proceedings Paper) AND LA=(English)	143
		Total	556

APPENDIX B
Sensor specifications

Table B.1 - ZH03B particulate matter sensor specifications

Parameter	Value
Measured variables	PM1.0, PM2.5, PM10
Detection principle	Laser scattering
Particle diameter range	0.3 - 10 μm
Measurement range	0 - 1000 μg/m³
Response time	T90 < 45 s
Detection interval	1 s
Typical accuracy	±15 μg/m³ or ±15% of reading, depending on concentration range
Interface	UART / PWM
Operating conditions	-10 to 50°C; 0 - 80% RH (non-condensing)
Reliability (MTTF)	> 10,000 h continuous operation

The ZH03B sensor provides rapid particulate measurements suitable for long-term urban monitoring, but its measurements remain sensitive to environmental conditions and low-cost optical sensing limitations.

Table B.2 - ZEHS04 multi-gas module specifications

Parameter	Value
Target gases	CO, SO_2, NO_2, O_3
Module type	Diffusion-based multi-gas sensing module (ZE12 series)
Output interface	UART and RS485
Response and recovery time	≤ 120 s
Resolution	≤ 10 ppb
Operating conditions	-20 to 50°C; 15 - 90% RH (non-condensing)
Typical lifespan	Approx. 2 years (sensor-dependent)
Deployment role	Long-term low-cost urban gas monitoring

APPENDIX C
System implementation and verification outputs

Topic: application/1/device/70B3D57ED0000003/event/up QoS: 0 Time: 2025-05-11 04:10:00


```
{
   "applicationID": "airquality-app",
   "deviceName": "node-3",
   "devEUI": "[redacted]",
   "rxInfo": [
      {"gatewayID": "gw-1", "rssi": -117, "loRaSNR": -13.1, "loRaSF": 12, "loRaBW": 125000}
   ],
   "fCnt": 34859,
   "dr": 0,
   "zone": "Z2",
   "decode_status": "ok",
   "eid": "eid_000003_202505110410_0a2118",
   "time": "[redacted]"
}
```


Figure C.1 - Example of a LoRaWAN uplink message received through MQTT and processed by the ingestion service

Launcher | measurements.csv

Delimiter: ,

	node_name	zone	ts	fcnt	dr	sf	rssi_dbm	snr_db	pm1_ugm3	pm25_ugm3	pm10_ugm3	temp_c
133896	node-3	Z2	2025-07-31 23:10:00	44923	0	12	-122	-12.2	6.6	10.7	12.7	13.0
133897	node-3	Z2	2025-07-31 23:20:00	44924	0	12	-118	-14.6	28.2	51.0	74.8	15.4
133898	node-3	Z2	2025-07-31 23:32:28	44925	0	12	-121	-13.0	21.7	34.8	47.1	32
133899	node-3	Z2	2025-07-31 23:42:40	44926	0	12	-122	-13.0	22.4	30.3	32.2	27
133900	node-3	Z2	2025-07-31 23:51:30	44927	0	12	-122	-12.3	30.4	42.1	57.2	28

Figure C.2 - Example of decoded measurement records stored in the measurements table

measurements.csv | provenance_records_final_FA | radio_rx.csv

Delimiter: ,

	Unnamed: 0	uplink_key	_id	zone	ts	gateway_id	rx_n	fcnt	dr	sf	rssi	snr
29	29	1_202408010450_45df7e	01	Z1	2024-08-01 04:50:00	gw-1	1	25	5	7	-108.0	-1.0
30	30	l_202408010500_a7ce55	01	Z1	2024-08-01 05:00:00	gw-1	1	26	5	7	-107.0	1.0
31	31	l_202408010500_a7ce55	01	Z1	2024-08-01 05:00:00	gw-2	2	26	5	7	-105.0	4.0
32	32	_202408010520_049b1d	01	Z1	2024-08-01 05:20:00	gw-1	1	27	0	12	-108.0	-16.0
33	33	_202408010520_049b1d	01	Z1	2024-08-01 05:20:00	gw-2	2	27	5	7	-103.0	-3.5

Figure C.3 - Example of radio reception metadata stored in the radio_rx table

measurements.csv | provenance_records_final_FA

Delimiter: ,

	rid	event_type	timestamp	agent_id	input_eids	output_eid	zone	fcnt	prev_hash	record_hash
1	10000_65c71d_RX	RX	2024-08-01 00:00:00	network_server	_202408010000_65c71d	_202408010000_65c71d	Z1	1.0	0000000000000000000	3047a9cf68693f66cb9a37
2	0_65c71d_INGEST	INGEST	2024-08-01 00:00:02	provenance_collector	_202408010000_65c71d	_202408010000_65c71d	Z1		3047a9cf68693f66cb9a37	ccbdbb28ef0f318301c8d4
3	00_65c71d_STORE	STORE	2024-08-01 00:00:03	storage_service	_202408010000_65c71d	_202408010000_65c71d	Z1		ccbdbb28ef0f318301c8d4	fc3cd2c74b37f5dce27443
4	10010_a78303_RX	RX	2024-08-01 00:10:00	network_server	_202408010010_a78303	_202408010010_a78303	Z1	2.0	fc3cd2c74b37f5dce27443	4c71c620e058e62145e60
5	0_a78303_INGEST	INGEST	2024-08-01 00:10:02	provenance_collector	_202408010010_a78303	_202408010010_a78303	Z1		4c71c620e058e62145e60	33874789f38743e1cd501

Figure C.4 - Example of provenance records generated for RX, INGEST, and STORE events

In the exported evaluation dataset, the failed ingestion records are labelled as DECODE_FAIL, for the convenience of the analysis. Note that DECODE_FAIL is only an analysis label, rather than an additional type of provenance event. In the formal provenance model, such records correspond to INGEST records with status =

FAIL, where the decoding error is stored in the reason field. As a result, the formal INGEST count is 135,000, namely 133,900 successful records and 1,100 failed records. Verification outputs for the detection of tampering in the provenance chains, obtained in Jupyter Notebook.

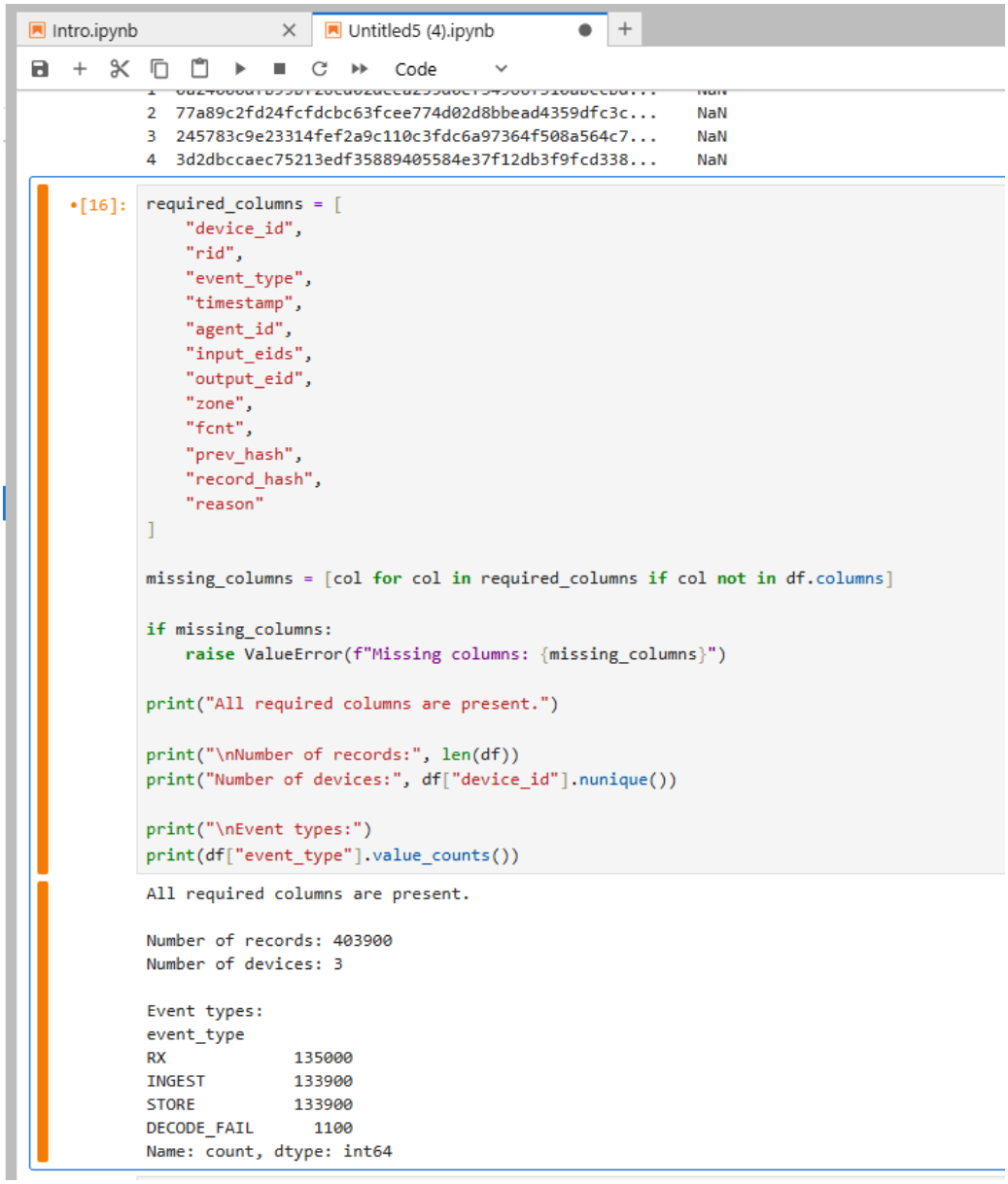


Figure C.5 - Structure, size, and event-label distribution of the exported provenance-record dataset



Figure C.6 - Intermediate record deletion detected through broken hash linkage

```
[31]: target_index = len(data_sorted) // 2

tampered_reorder = data_sorted.copy()

idx1 = target_index
idx2 = target_index + 1

row1 = tampered_reorder.loc[idx1].copy()
row2 = tampered_reorder.loc[idx2].copy()

tampered_reorder.loc[idx1] = row2
tampered_reorder.loc[idx2] = row1

reorder_link_errors = verify_chain_links_in_current_order(tampered_reorder)

print("Scenario 3: Record reordering")
print("Link errors:", len(reorder_link_errors))

display(reorder_link_errors.head())
```

```
Scenario 3: Record reordering
Link errors: 3
```

	device_id	rid	event_type	error_type	expected	actual
0	70B3D57ED0000002	rid_000002_202501300750_2dd3bf_INGEST	INGEST	prev_hash_mismatch	de54e82fb71e1f35c023e631930ad82edb7920c69ae431...	254785491e064c732d65d546825ff4d8f8780409e2aee7...
1	70B3D57ED0000002	rid_000002_202501300750_2dd3bf_RX	RX	prev_hash_mismatch	fc09e040c03690a373ba918ac4802ae6f7e97f9658740f...	de54e82fb71e1f35c023e631930ad82edb7920c69ae431...
2	70B3D57ED0000002	rid_000002_202501300750_2dd3bf_STORE	STORE	prev_hash_mismatch	254785491e064c732d65d546825ff4d8f8780409e2aee7...	fc09e040c03690a373ba918ac4802ae6f7e97f9658740f...

Figure C.7 - Detection of record reordering in a hash-linked provenance chain

```
[34]: tampered_orphan = data_sorted.copy()

fake_record = tampered_orphan.loc[target_index].copy()

fake_record["rid"] = "FAKE_RECORD"
fake_record["event_type"] = "FAKE"
fake_record["reason"] = "Artificial orphan insertion"
fake_record["prev_hash"] = "f" * 64
fake_record["record_hash"] = "a" * 64

tampered_orphan = pd.concat(
    [
        tampered_orphan.iloc[:target_index + 1],
        pd.DataFrame([fake_record]),
        tampered_orphan.iloc[target_index + 1:]
    ],
    ignore_index=True
)

orphan_link_errors = verify_chain_links_in_current_order(tampered_orphan)

print("Scenario 4: Orphan record insertion")
print("Link errors:", len(orphan_link_errors))

display(orphan_link_errors.head())
```

```
Scenario 4: Orphan record insertion
Link errors: 2
```

	device_id	rid	event_type	error_type	expected	actual
0	70B3D57ED0000002	FAKE_RECORD	FAKE	prev_hash_mismatch	254785491e064c732d65d546825ff4d8f8780409e2aee7...	ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff...
1	70B3D57ED0000002	rid_000002_202501300750_2dd3bf_INGEST	INGEST	prev_hash_mismatch	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...	254785491e064c732d65d546825ff4d8f8780409e2aee7...

Figure C.8 - Detection of orphan record insertion in a hash-linked provenance chain

Scenario	Records checked	Invalid hashes	Broken links	Checkpoint mismatch	Result
Original provenance chains	403900	0	0	No	PASS
Content modification	403900	1	0	No	Detected
Intermediate-record deletion	403899	1	1	No	Detected
Fake-record insertion	403901	2	1	No	Detected
Record reordering	403900	3	3	No	Detected
Tail truncation	134812	0	0	Yes	Detected

```
Controlled tampering scenarios detected: 5/5
```

Figure C.9 - Summary output of the five controlled provenance-chain tampering-detection tests

APPENDIX D
Simulation code and reproduction of the attribution experiment

In this appendix we describe the software environment, the source files and the commands used to obtain the simulation results of Section 4.6. All the results reported in Section 4.6 were produced by means of such procedure, on the computer of the author.

D.1 Software environment

The simulation was run on a laptop with an Intel Core i5-10300H processor (i.e., 8 logical cores), under Windows 10 Pro with the Windows Subsystem for Linux (WSL 2). From the system memory, 7.7 GB were available to WSL. The Linux environment was Ubuntu 26.04 LTS, with GCC 15.2.0, CMake 4.2.3 and Python 3.14.4, together with the NumPy, pandas and Matplotlib libraries. The radio layer used ns-3 version 3.48 (i.e., release tag ns-3.48 of the official ns-3 repository) and the LoRaWAN module version 0.3.7 (i.e., release tag v0.3.7 of the signetlabdei/lorawan repository), installed in the src/lorawan directory of ns-3.

Note that the LoRaWAN module 0.3.7 does not compile with ns-3.48 without a small change, since several source files use logging macros without including the corresponding header. For this reason, the line #include “ns3/log.h” was added to every source file of the module that uses such macros. No other change was made to ns-3 or to the module.

D.2 Source files

The experiment consists of three files:

1) attr-radio.cc, an ns-3 program in the scratch directory, which simulates the target devices, the background devices and the gateways, and writes one CSV row per transmission of a target device, with the receiving gateways and the received power;

2) runs.sh, which runs the 40 radio simulations listed in Table D.1;

3) attr_ns3.py, which implements the second simulation layer and the attribution procedure (Algorithm 3.3), and compares the result with the true causes.

Table D.1 - Radio simulation runs

Trace set	Gateways	Fading SD	Background devices	Duration	Seeds
1	2	3	4	5	6
base	2	6 dB	50	365 days	1 -10
gw1	1	6 dB	50	365 days	1 - 5
gw3	3	6 dB	50	365 days	1 - 5
fade3	2	3 dB	50	365 days	1 - 5
fade10	2	10 dB	50	365 days	1 - 5
load0	2	6 dB	0	365 days	1 - 5
load500	2	6 dB	500	30 days	1 - 5

In all the runs, three target devices transmit every 600 s with SF9, while ADR is disabled. The devices are located at (3200, 300), (3300, -400) and (3400, 200) m, and the gateways at (0, 0) and (10400, 0) m. In the gw3 trace set, the third gateway is located at (5200, 6000) m. The seed of each ns-3 run is passed as the RngRun parameter, while the second layer uses fixed seeds derived from the radio seed, so each run is deterministic.

D.3 Commands

```
# ns-3 and the LoRaWAN module
git clone --depth 1 --branch ns-3.48 https://gitlab.com/nsnam/ns-3-dev.git
git clone --depth 1 --branch v0.3.7 https://github.com/signetlabdei/lorawan.git ns-3-dev/src/lorawan
cd ns-3-dev
grep -rL '#include "ns3/log.h"' $(grep -rl 'NS_LOG' src/lorawan --include=*.cc) \
  | xargs sed -i '0,/^#include/s//#include "ns3\/log.h"\n#include/'
cp ../code/attr-radio.cc scratch/
./ns3 configure --build-profile=optimized --enable-modules=lorawan --disable-werror --disable-examples --disable-tests
./ns3 build -j4
cd ..

# radio layer (40 runs)
bash code/runs.sh

# second layer and attribution (26 scenarios in 4 parallel parts)
for p in 0 1 2 3; do
  python3 code/attr_ns3.py --traces traces --part $p --nparts 4 --out results_p$p &
done; wait

# figures and summary table
python3 code/make_figures_ns3.py results_p0,results_p1,results_p2,results_p3 figures
```

Figure D.1 shows the summary of the simmulations

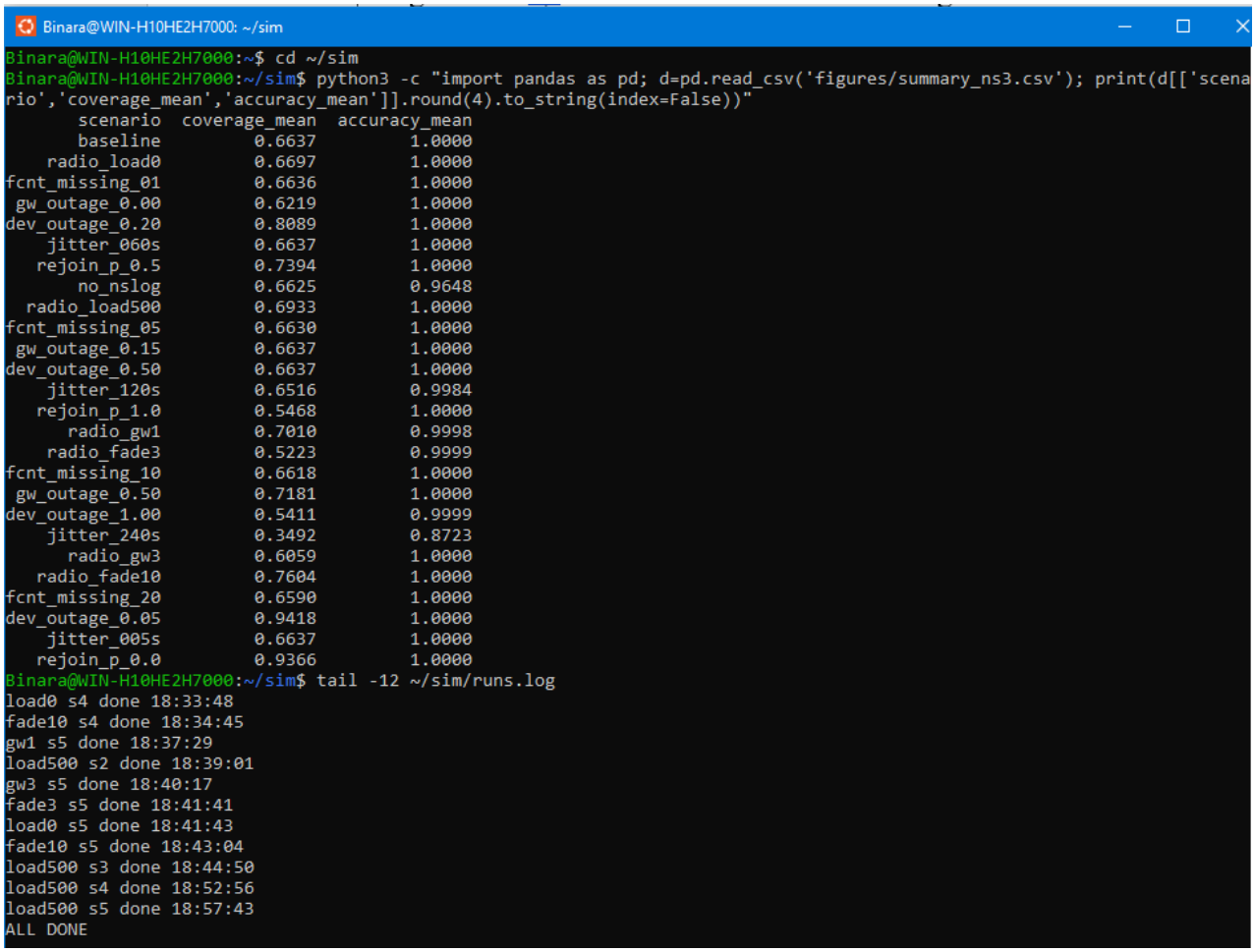


Figure D.1 - Output of the simulation on the computer: mean coverage and accuracy of the attribution for each scenario, rounded to four decimal places (upper part), and completion of the last ns-3 radio simulation runs (lower part)

APPENDIX E
Act of implementation

«SDU University»
мекемесі

Учреждение
«SDU University»

SDU
1996
UNIVERSITY

040900, Алматы облысы, Қарасай ауданы,
Қаскелең қаласы, Абылай хан көшесі 1/1
тел.:+7727 307-95-60, факс: 307-95-58

040900, Алматинская область, Қарасайский
район, г. Қаскелең қаласы, ул. Абылай хана
1/1 тел.:+7727 307-95-60, факс: 307-95-58

БЕКІТЕМІН
Бірінші проректор
_____Гаипов Д.
_____2026 ж.

Оқу-әдістемелік және ғылыми жұмыс нәтижелерін оқу процессіне енгізу туралы АКТ

1. **Жұмыс енгізілетін мекеме атауы:** «SDU University» мекемесі
2. **Ұсыныс атауы:** Implementation of the results of the dissertation research entitled "Design and implementation of a provenance-aware LoRaWAN IoT framework for smart city air-quality data management" into the educational, research, and project activities of the IoT Laboratory at SDU University.
3. **Оқу-әдістемелік құрал және ғылыми жұмыс нәтижесін оқу процессіне енгізу формасы:** The results of the research work have been integrated into the educational process through teaching materials, laboratory assignments, research projects, and practical case studies in IoT, LoRaWAN, smart city systems, environmental monitoring, and data provenance.

   The implemented results include the IoT architecture for air-quality monitoring, methods for collecting and processing sensor data, storage of measurements and radio metadata, and the use of provenance records to ensure data traceability.
4. **Енгізу аясы:** Educational and research activities of the IoT Laboratory at SDU University in the fields of IoT, LoRaWAN, and air-quality monitoring.
5. **Апробация мерзімі:** 2025-2026 academic year
6. **Апробация нәтижелері:** The developed scientific and technical solutions were used in the IoT Laboratory at SDU University for student research and project activities related to indoor air-quality monitoring, IoT systems, LoRaWAN communication, and sensor data analysis. The approbation also contributed to the preparation of the following publications:

   1) Q2, Scopus Q1: B. Imankulova et al., "Tracing data origins in smart cities: an IoT perspective," *IEEE Access*, vol. 13, pp. 130201–130214, 2025. DOI: https://doi.org/10.1109/ACCESS.2025.3591277

2) Scopus: B. Imankulova, A. Zhakipova, and A. Razaque, "IoT-based real-time indoor air quality monitoring and web-server management system using Raspberry Pi," *DTESI 2024*, CEUR Workshop Proceedings, vol. 3966, pp. 1–12, 2024.

3) KOKSON: B. B. Imankulova, "Privacy challenges and framework design for IoT devices: a comprehensive approach," *Bulletin of AUPET*, vol. 68, no. 1, pp. 256–267, 2025. DOI: https://doi.org/10.51775/2790-0886_2025_68_1_256

4) KOKSON: B. B. Imankulova, L. A. Kozina, N. T. Duzbayev, and S. T. Amanzholova, "Transmission, storage and preprocessing of data received from sensors using LoRa technology," *Bulletin of AUPET*, vol. 64, no. 1, pp. 16–24, 2024. DOI: https://doi.org/10.51775/2790-0886_2024_64_1_16

7. **Енгізуге жауапты:** Imankulova Binara
8. **Енгізу тиімділігі:** The implementation is necessary to support the use of real IoT systems and experimental data in education, and to prepare students for applied tasks in smart city systems, environmental monitoring, and sensor data analysis.
9. **Енгізудің қажеттілігі:** high

Мектеп деканы: **Ахмедов Р.**

Кафедра меңгерушісі: **Орынбекова К.**

Оқу-әдістемелік Кеңесінің № 10 **, «** 19 **»** шілде **2026 ж. хаттамасымен**