

Article

A Formal Model for Secure and Context-Based Data Dissemination in Federated Special IoT Environments

Jakub Sychowiec *  and Zbigniew Zieliński 

Faculty of Cybernetics, Military University of Technology, 00-908 Warsaw, Poland; zbigniew.zielinski@wat.edu.pl

* Correspondence: jakub.sychowiec@wat.edu.pl

Abstract

An increasing number of special Internet of Things (IoT) applications are being deployed within federated and zero-trust (ZT) environments. These ad-hoc networks consist of heterogeneous, resource-constrained devices from various administrative domains, all of which are susceptible to compromise. The dynamic nature of these environments necessitates near-real-time Situational Awareness (SA), where processed data varies with its sensitivity and reliability, without dependence on a central authority. Examples include NATO and non-NATO coalitions engaged in hybrid military operations or humanitarian aid scenarios. To address the challenges of security, reliability, and context-aware data dissemination, we propose *FedM*, a multi-level formal model designed for context-aware and policy-driven data dissemination in federated IoT environments. This model is built upon various access control models and Denning's research on information flow control (IFC), prioritizing the protection and reliability of data flows. A crucial element of this model is the distributed ledger, which facilitates the dynamic modification of label expressiveness, enhances resilience against disruption attacks, and separates policy logic from application functionality to mitigate risks associated with the benevolent developer. Additionally, we delineate a deterministic and history- and precedence-aware policy enforcement procedure to resolve conflicting actions and introduce processing primitives for the ongoing Data Quality Assessment (DQA) process. Our model also aligns with the concepts of Ubiquitous and Continuum Computing. Furthermore, in our paper we illustrate a policy-based dissemination pipeline, incorporating a bounded trustworthiness dimension. Additionally, we present a refined multi-layered framework that proposes the deployment of Information Flow Control (IFC) components, such as the Open Policy Agent decision engine, to facilitate policy-driven contextual data dissemination. We provide preliminary benchmarks for resource-constrained platforms, along with a formal threat model that addresses implicit flows, the benevolent developer problem, and the behavior of a distributed ledger under degraded network conditions. Finally, we conduct a formal verification of our model using the P framework.



Academic Editors: Lei Shu, Boja Catalin and Mihai Doinea

Received: 18 June 2026

Revised: 26 July 2026

Accepted: 28 July 2026

Published: 1 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: Internet of Things; multilevel security; formal model; blockchain

1. Introduction

The dynamic Observe–Interpret–Predict–Act cycle, referred to as Situational Awareness (SA), begins with the perception and recognition of elements in the environment, followed by the application of context and interpretation. This process is further enhanced by reasoning—including judgment, projection, and prediction—concerning the potential

evolution of these elements, ultimately culminating in a comprehensive understanding of the environment around the entity.

In essence, insights and experience are derived from raw data inputs provided by entities, processed into meaningful and actionable wisdom, which enhances the certainty of decision-making. In contrast, the uncertainty indicator reflects the challenge of accurately predicting the consequences of actions due to incomplete or unreliable information.

Moreover, the risk of high uncertainty is particularly prevalent in self-contained (or closed) networks formed by many heterogeneous and power-constrained entities capable of sensing and actuating data, referred to as Internet of Things (IoT) systems. The deployment of these networks can be either fixed or ad-hoc. Additionally, through the integration of edge gateway components, an IoT system can gain access to comprehensive cloud solutions, thereby unlocking the full potential of IoT entities for context-aware applications.

Notably, the inherent characteristics of IoT entities often result in limited computing capabilities, necessitating the involvement of designated entities within the environment for data processing. This situation aligns with the concepts of Pervasive (Ubiquitous) Computing [1] and Continuum Computing [2] that aim to integrate sensors and actuators into daily life, allowing them to communicate, sense, and adapt to the needs of end-users with minimal or no direct user interaction—being unobtrusive. At the same time, computing capabilities are provisioned across various layers of processing environments.

From the perspective of the military domain, the SA along with the IoT components have become firmly grounded [3,4], where an uncertainty indicator may have contributed to the mission's success or failure. Consequently, the increasing number of special IoT applications utilize a novel cybersecurity posture of Zero-Trust [5] environments and a federated structure that can impact and reduce the level of uncertainty. In essence, this posture means that all entities are treated as potentially compromised (malicious). This implies continuous verification and approval of entities and data flows through security processes such as authentication, authorization, and policy enforcement. In contrast, the federated structure aims to enable different entities to use shared resources and disseminate data securely and efficiently without relying on a central authority, thus facilitating cooperation and increasing the resilience of the entire system.

In summary, there is a demand for a context-aware, distributed, and federated environment that supports decision making and command and control processes. Figure 1 illustrates the high-level model we have adopted, drawing on research from Zaccarini et al. [2] with respect to continuous computing and the data quality assessment process proposed by Byabazaire et al. [6]. This model integrates special IoT networks with context-aware intelligence services and serves as the basis for our research. Examples of such platform utilization include the creation of federations of NATO countries and non-NATO organizations participating in hybrid military operations or the interaction of civilian services and the military when providing Humanitarian Assistance and Disaster Relief (HADR).

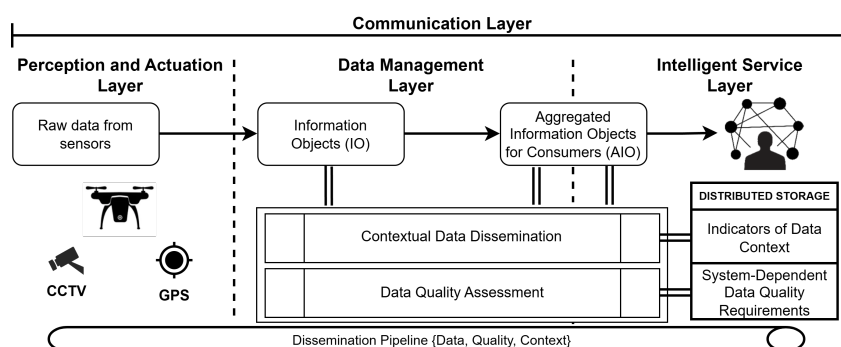


Figure 1. Special IoT context-aware architecture.

In general, the noted intelligence platform presents several challenges related to security, reliability, and dynamic (context-aware) data dissemination, which:

- Involves the Communication Layer that necessitates secure communication between heterogeneous entities that operate within the Perception and Actuation Layer and belong to various federated administrative domains;
- Pertains to the dynamic allocation of resources within the Data Management and Intelligent Service Layer that is crucial for effectively managing trade-offs in data dissemination and providing near-real-time data presentations to end users while considering their key performance indicators (KPIs);
- Involves the dynamic dissemination of data in response to the evolving information demands of end users, referred to as contextual dissemination. This challenge encompasses the dissemination and processing of raw data from sensors, aimed at generating relevant Information Objects (IOs). These IOs can then be aggregated for consumption by the Intelligent Service Layer, particularly for use by appropriate applications.

Until now, our research contributions to address these challenges have been articulated in articles [7–9]. These articles explore a novel framework for secure and dynamic data flow dissemination within multi-organizational federations utilizing Distributed Ledger Technology (DLT), which employs unique entity characteristics (fingerprints) for the authentication and verification of data flow [7]; the integration of a data-centric security paradigm that ensures origin-based data protection [8]; and an evaluation of the performance of the framework in both cloud-based and Raspberry Pi (RPi) resource-constrained environments [9].

In this article, we cover the contextual dissemination challenge. In essence, it varies on the basis of indicators (constraints) that help determine the context of data payloads. In addition, these indicators serve to specify the relevant attributes (labels) of the evaluated data payload from appropriate dimensional spaces, such as secrecy (*public*, *confidential*) or data categories (*military*, *HADR*). Notably, these labels can be combined to facilitate multi-labeled dissemination. Furthermore, these attributes, along with the indicators, can evolve throughout the data processing pipeline. On one hand, changes to attributes may occur due to intermediate transformations and fusion operations governed by relevant data policies. For example, a transformation operation that declassifies a data payload may change its secrecy label from *secret* to *public*. On the other hand, the evolving nature of data policies and their indicators can be redefined over time, directing data flow toward alternative groups of data consumers.

Delving deeper, within multi-class data types, the quality of sensed information can be compromised by environmental factors or low-quality sensors, leading to data quality issues such as data duplication or obsolescence, all of which impact overall Situational Awareness. Consequently, context-aware applications have different Data Quality Requirements based on their specific domains of use. These requirements are expressed through attributes of the Data Quality Dimensions, such as timeliness or trustworthiness. The values of these attributes are established through continuous objective (or subjective) quality measurements, known as the Data Quality Assessment process [6]. This is an example of a trustworthiness metric that can be employed to determine the validity, usefulness, and certainty of entity (source of data) and data itself in multi sourced environments.

Based on the aforementioned, our main contributions within this article are as follows:

1. We define our formal model for the secure and Context-Based Data Dissemination in Federated IoT Environments (*FedM*), where we prioritize the protection of data flows and the reliability of their delivery. Our model encompasses primitives and dependencies among these elements, enabling us to delineate—through data policy—the core and intermediate functional stages for the evolving contextual dissemination of

data. Moreover, we adhered to the conclusions proposed by Denning [10], asserting that a practical system must incorporate both access and dissemination controls to satisfy all security requirements. Consequently, our model encompasses both aspects. This formal model has not previously been defined in our previous work [7–9], which addressed the dissemination mechanism, its security, and its performance.

2. To mitigate the uncertainty indicator, simultaneously achieving a clearer understanding of the environment around the entity, we defined processing primitives; essentially, we determined the quality assessment elements that are fundamental for the DQA process. In addition, our model strictly aligns with the concepts of Ubiquitous and Continuum Computing. This DQA-oriented contribution is entirely new in relation to [7–9] and was not part of their scope;
3. We outline a practical deployment example of a model-driven architecture in which the primitives and dependencies of *FedM* are instantiated by concrete components (modules) similar to those utilized in Information Flow Control (IFC) systems, including the Policy Specification module and the Policy Enforcement Point (PEP) component. This is a new, IFC-oriented deployment layer dedicated to policy-driven contextual dissemination—not a revision of the DLT-based dissemination framework presented in [7], nor a modification of the security or performance results in [8,9]. However, it reuses selected technologies and components already available within our solution, keeping re-engineering effort low. In particular, a new integrated technology is the Open Policy Agent [11], which functions as a part of the PEP component.

The remainder of the article is structured as follows: Section 2 provides an overview of the relevant research that serves as the foundation for our formal model. Section 3 details the conceptual architecture of IFC systems, including IFC design questions and identified classes of IFC threats. Section 4 presents our formal logical model, *FedM*, describing its primitives and the dependencies among these elements. Section 5 presents an example of a data dissemination pipeline driven by data policy. In addition, we explain the application of a bounded trustworthiness dimension (uncertainty metric) along with the data quality assessment microservice through the utilization of our model elements. Section 6 outlines our model-driven, multi-layered framework, proposing a deployment of IFC components that facilitate policy-driven contextual data dissemination. Furthermore, we conducted a preliminary benchmarking of the OPA decision engine deployed on a resource-constrained platform. Section 7 presents a threat model for our framework, specifically focusing on three threats: implicit flows, the benevolent developer, and the behavior of the distributed ledger under degraded network conditions. To enhance confidence in the correctness of the *FedM* model and the Flow Preservation Theorem, we performed formal verification using the *P framework* [12] in Section 8. Finally, Section 9 summarizes our findings and outlines our goals for future work. The publication concludes with a definition of the abbreviations used throughout the text.

2. Related Works

Formal models in the realm of data-centric security provide rigorous mathematical or logical representations of primitives, the dependencies among them, invariants that must remain true across all system states, and constraints (rules)—specifying what is allowed and to what extent. Consequently, formal models ensure the enforcement of consistency, correctness, unambiguity, and security properties within computing environments.

In our analysis, we identify two primary categories of formal models. The first category encompasses access control formal models, which govern the actions of subjects regarding objects, specifically indicating who may access a resource at any given time. The second category pertains to information flow control (dissemination) models, which determine the

flow of data based on data-relevant attributes. Additionally, due to the nature of chained processing actions, such as transformations and fusions, these attributes may be combined, enhanced (strengthen) or relaxed (declassified). In essence, IFC models determine where data can flow as a result of data processing.

Beyond formal models, it is essential to define a data policy, which outlines the formal structure of data-relevant attributes, constraints, and actions relevant to those constrained data as established by the organization. Moreover, the data policy represents the data unambiguity framework in certain situations, ensuring clarity in data management through defined procedures, thereby facilitating effective dissemination of data flows among the entities involved in the computing environment's lifecycle.

Although this article primarily addresses data policy, it is important to note that such policies can extend beyond data management alone. Generally, the utilization of policies contributes to the establishment and maintenance of consistent and compliant environments, where uncertainty can be reduced and trustworthiness enhanced. Traditionally, a policy may also serve to formalize aspects of network management or resource orchestration.

A comparative analysis of access control models conducted by Penelova [13] identified 27 distinct access control models. Here, we choose to point out two representative models: Mandatory Access Control (MAC) and Discretionary Access Control (DAC), as these models form the classical foundation for subsequent alternatives, such as role-based and attribute-based access control models.

At their core, access control models abstract subjects—entities that process data, such as IoT sensors and actuators—and their associated objects—resources (storage mediums) that persist data at-rest—into relations organized within dedicated structures that encompass entities, objects, and their security or data-relevant attributes, such as secrecy or integrity.

For example, consider the structure of security labels, where labels assigned to entities are referred to as security clearances, while those assigned to objects are known as security classifications. Under the general system invariant termed “no read up”, an IoT sensor with secret clearance is permitted to read data from the secure storage classified as *secret* and lower classes dominated by it, such as *confidential* or *public*. Conversely, the “no write down” invariant ensures that sensitive data cannot leak downward through write operations, and only data written to the same label class or higher is allowed. These secrecy invariants proposed by the Bella–LaPadula (BLP) mathematical model [14], along with the integrity invariants derived from the Biba model [15], form the basis for the MAC model.

The MAC model adopts a centralized approach with rules that are strictly defined (governed) for the entire environment. This enables a uniform and holistic view that facilitates access decisions based on global policies. Furthermore, it ensures rigorous enforcement of constraints and suits multi-tenant environments. However, the rigidity of centrally maintained security labels introduces a burden in terms of policy maintenance and limits the operational flexibility available to individual resource owners.

In contrast, the DAC model is characterized by decentralization and user control, allowing resource owners to manage access permissions, including granting or revoking access to specific objects. Typically, a constraints check occurs at discrete designated points within a system, which can create a single point of failure. This is equivalent to the fact that the distributed manner of these points can lead to ungoverned enforcement gaps, i.e., implicit flows, as there is no centralized authority to ensure consistency in the constraints defined by individual owners. From the perspective of a small-scale organizational structure, DAC enables flexibility and ease of maintenance. However, in multi-tenant zero-trust environments, this model imposes a significant burden concerning the monitoring and definition of data-relevant attributes, as well as their correct assignment to entities.

Taking into account IFC models, the lattice model of secure information flow proposed by Denning [10,16] has had a significant impact on our field of research. Unlike access control models, IFC models encompass both the storage medium and entities within the object set. Furthermore, Denning's model abstracts the dedicated group of processing entities responsible for data flow operations, including data modification, into a set of processes. It is important to note that these processes do not function as subjects in the BLP sense. Instead, they serve as mechanisms through which data transits (flows) from one storage object to another. However, before this flow can occur, the enforcement mechanism checks the relevant constraints. Thus, in Denning's model, each object and process is bound (associated) with a security class that reflects either the information stored in that specific object or the processing purpose of the entity. Furthermore, these security classes extend beyond the traditional security labels of access control, allowing the definition of various classification dimensions, such as the dimension of the data category (*military*, *HADR*). Moreover, the defined classes form a lattice structure with a partial order that delineates allowed flows, where the enforcement invariant states that data flows may only occur toward objects of equal or dominating classes.

As a supplement to the description of the data policy, it is important to emphasize that such a policy can encompass complex dissemination scenarios wherein data undergoes modifications throughout their flow. Consequently, we identify two primary categories of data alteration: data transformation and data fusion. The former is relatively straightforward, involving the processing of a single data flow, while the latter entails the integration of multiple data sources to create enhanced IOs (or AIOs), ultimately improving overall SA. Notably, data fusion is a multidisciplinary area without a clear and strict classification. In our work, we proposed the incorporation of the probabilistic Durrant-Whyte model [6,17], which distinguishes fusion approaches based on relationships among sources, identifying the following classes of data fusion strategies:

1. Complementary class denotes approaches where a combination of data from multiple different sources leads to complete system representation;
2. Redundant class reduces uncertainty of the information, where two different entities achieve the same assumption based on the source of information;
3. Cooperative class involves two different entities achieving convergent results, leading to more complex information than the original one.

Consequently, the fusion as well as transformation operations may lead to change (re-assignment) in relevant data attributes. In reference to Denning's work, a dynamic binding mechanism is employed, whereby the security class assigned to an object varies according to its actual content. Therefore, the class-combining mechanism determines the combined security class when data from multiple sources undergoes the fusion process.

For instance, consider the ordered security lattice: $Public \prec_{\lambda} Confidential \prec_{\lambda} Secret \prec_{\lambda} TopSecret$, and two data sources with statically bound classes: a CCTV camera (*Public*) and a GPS sensor (*Secret*). These data sources combined create a new information source by providing a log of the location where a photo was taken. When data from these sources is fused, the resulting data inherits the highest classification level, *Secret*, from its inputs. Notably, this represents an invariant where the security classes of dynamically bound objects consistently remain monotonically non-decreasing throughout processing operations that involve data fusion.

Furthermore, one of Denning's conclusions highlighted that a practical system must incorporate both access and dissemination controls to fulfill all security requirements. This distinction makes IFC models significantly more robust than access control alone; a system may be correct in its access controls yet still unintentionally reveal protected data through implicit flows. Importantly, by employing the Denning model elements in conjunction with

the invariants of the BLP model, we can effectively define an access control mechanism similar to the MAC model. For instance, the security class of a process serves as a constraint (bound) dictating what class of object it may read or where it may write its output.

The practical implementation of these controls is exemplified by IFC systems. In general, these systems facilitate data protection, ensure appropriate data dissemination (routing), and limit implicit flows. In essence, the IFC system varies on data-relevant attributes that are tied to data, where the dedicated structure defines allowed label relations between data and entities that process these data. Simultaneously, the IFC continuously tracks data flows from the source to the destination, ensuring that both the data and its labels comply with the constraints imposed in a strict and holistic manner.

In particular, we have referenced research on IFC systems that, in our opinion, have significantly influenced the formal model and solution presented in this article. Bacon et al. [18] conducted a comparative analysis of various cloud-relevant IFC frameworks, evaluating aspects such as model tracking granularity, re-engineering effort, and label expressiveness. Here, we focus on these models that enable granularity at the data message level. Table 1 provides a comparison of these models alongside our *FedM* model. SafeWeb [19] supports the use of security and integrity labels, with data and sticky labels encapsulated in an event format for processing by a centralized event backend. DEFCon [20] employs parallel labels for message parts and was specifically designed for the financial trading domain, where low latency is a critical KPI. DStar [21] incorporates cryptographic label structures to enable decentralized verification and delegation within untrusted environments. All the frameworks mentioned are characterized by a significant re-engineering effort necessary for their integration with existing systems. However, the architecture of the SafeWeb system indicates that it can be deployed with a lower operational burden than the others mentioned and is closest to our proposed solution.

Table 1. Comparison of different cloud-relevant distributed IFC systems.

Property	SafeWeb [19]	DEFCon [20]	DStar [21]	FedM
Domain	Web services (cloud)	Financial trading	Distributed systems within ZT environments	Distributed systems within ZT environments
Tracking Granularity	Data Message (Event)	Parts of Data Message	Data Message	Parts of Data Message
Label Expressiveness	Confidentiality and integrity	Confidentiality and integrity	Confidentiality and integrity	Various dimensions over discrete or continuous (bounded) value spaces, interpreted structurally and semantically by data schemas
Label Encapsulation	Sticky labels encapsulated in an event format	Labels attached to message parts	Labels attached to message	Labels attached to message headers
Federation Support	No	No	No	Yes
Verification (Enforcement)	Centralized enforcement via event processing backend	Distributed middleware enforcement	Decentralized enforcement on each machine OS-level	Decentralized enforcement via microservices
Data Policy Storage	Single policy service accessed by all enforcement points	Single policy service accessed by all enforcement points	Local per-machine	Distributed ledger or locally cached

Finally, to the best of our knowledge, there is currently no comprehensive formal model that simultaneously integrates the following elements: (a) access and dissemination

controls, emphasizing the protection of data flows and the reliability of their delivery; (b) multi-dimensional, evolving data policies that outline data dissemination pipelines in alignment with the concepts of Ubiquitous and Continuum Computing; (c) a Data Quality Assessment process that continuously measures bounded dimensions, such as trustworthiness, to reduce the uncertainty indicator prevalent in modern Situational Awareness environments; (d) a data-centric and history-aware policy-enforcing decision procedure that facilitates message-part level granularity and resolves conflicting actions; and (e) native mapping to a distributed ledger and message-oriented processing systems within federated IoT environments, where agreed-upon policies can be secured using blockchain technology to ensure accountability while simultaneously mitigating the risks associated with deny, delay, disrupt, destroy, or manipulate (D4M) results.

3. Information Flow Control System

In the following section, we have presented the conceptual architecture of the Information Flow Control system by distinguishing its components functions [18]. These components are then described by our formal model. Furthermore, we identified several classes of malicious vectors that impose the risk of occurrence of the D4M results within IFC solutions. In addition, we pointed out and explained the significance of four IFC design questions for which answers should be determined prior to the specification of these components.

3.1. System Architecture

The IFC architecture is built upon several components (modules) that operate within the Communication and Data Management Layers of IoT architecture. Therefore, these layers manage policy-defined data channels that are established between these components, facilitating the routing (dissemination) of data flows to mitigate potential implicit flows. Below, we outline the roles of the IFC components:

- Classification (Labeling) Endpoint component enables the assignment of defined and approved data-relevant attributes (labels) to entities and data flows;
- Labeling Management module dynamically controls and examines requests for attribute change (their re-assignment);
- Policy Enforcement Point is a tamper-resistant oracle that states actions on the data flows, e.g., data write (approve), data discard (reject), or data modification;
- Policy Specification module is the core of the PEP, where applicable policy labels, invariants, constraints, and actions about data flows are stored;
- Tracking Engine component continuously intercepts and inspects each data flow to provide insight for the PEP;
- Audit and Logging Component ensures accountability of monitored data flows and the PEP decisions. Furthermore, it enables policy tuning capabilities through analysis of policy-related events, such as constraint failure or success.

It is important to note that there are scenarios in which it becomes necessary to re-assign the labeling of data flows, specifically changing the information's secrecy classification from *Confidential* to *Public*. This re-assignment may occur due to the timeliness of data (as it may become outdated) or the need to share certain information with a broader group of consumers. This process of privilege relaxation is known as declassification. Conversely, the narrowing of a privilege is referred to as strengthening. However, the declassification approach contradicts the invariants proposed by Bell–LaPadula—“no read up, no write down”—as well as Denning’s class-combining mechanism. Therefore, the IFC architecture should enable controlled declassification, which requires implementing and

managing additional components regarding data anonymization and sanitization within the secure enclave of the IFC system, known as the trusted computing base.

3.2. Attack Surface

The design of robust IFC systems necessitates the identification of potential D4M threats and the consideration of suitable countermeasures to mitigate the risk of such threats. However, certain security controls may impose limitations on achieving the established environmental KPIs. This highlights the trade-offs between security and efficiency during the design specification phase of IFC solutions. Below, we outline some well known threats:

- Benevolent developer [22] refers to a trusted, non-malicious, and privileged entity, such as a system developer, administrator, or data policy author. However, this individual may inadvertently introduce vulnerabilities into the system due to factors such as environmental complexity, dynamic or ambiguous labeling schemes, or misunderstandings of the intended meaning of the policy.
- Implicit Flow [23] characterizes with indirect data flows due to the existence of conditional branches (data paths), which are unverified and unspecified in policy. This can lead to different execution paths and, at the same time, reveal the information that must be protected;
- Label Creep [24] occurs due to progressively layering of labels of processed (fused) data meaning that over time access to the data is more limited than necessary. Therefore, the appropriate conglomeration and cardinality rules must be implied. Failing to do so can result in system performance delay and ultimately its disruption.

3.3. Design Challenges

Modern distributed and context-aware environments are characterized by their dynamic evolvability. The appropriate design of the IFC system depends on the target environment and general security posture that needs to be fulfilled. Additionally, the lifecycle of IFC components should extend independently and be longer than that of the entities that produce and consume data. Consequently, significant questions must be addressed during the specification phase of IFC components.

How is data isolated?

The indicated question contributes to determining the required visibility level of data across its various states, including in-transit, in-process, and at-rest. Therefore, it aids in selecting the appropriate data isolation technologies and methods, which may prevent the propagation of data to unauthorized entities. Notably, separation architectures differ in their KPIs and data visibility levels. Programmatic isolation leverages built-in language and library features, but requires developers to have a solid understanding of specific APIs, increasing the risk of the benevolent developer threat. Alternatively, system component isolation—such as OS separation for different data flow labels, system processes isolation, or data buses isolation—imposes management and scalability burdens, while allowing comprehensive configuration and hardening of networking, storage, processing, and virtualization components.

Therefore, in performance- and agility-critical environments, containers alongside microservices are preferable. Compared to virtualization, containers incur lower resource overhead, offer better CPU performance due to reduced hypervisor demands, and achieve near-native I/O and network performance through direct OS kernel sharing. For isolation, containers use process-level separation, while virtualization provides full OS and hypervisor separation. Notably, while all microservices qualify as services, not all services are

microservices—the key distinction being scope: microservices are task-specific, whereas traditional services tend to be more coarse-grained.

How data is tracked?

The Tracking Engine continuously intercepts and inspects all incoming and outgoing data flows. Additionally, the tracking module can function in either inline or outline (mirrored) mode. Essentially, inline mode occurs within the data path and enables immediate policy enforcement (detection and prevention). Conversely, outline mode entails a delay in detection and cannot prevent violations in near-real-time.

Furthermore, the highlighted question pertains to the granularity of tracking classes supported by IFC solutions, which may be as follows: group of messages (queues), single messages (events), and message parts. Notably, as the granularity becomes finer, the overhead on IFC components increases proportionally, resulting in more precise tracking data. One effective strategy to mitigate this burden while preserving the desired granularity level is to identify and reduce the number of data flow (communication) channels, thereby simplifying the IFC architecture.

How is policy enforced and implied on data?

On the one hand, the provided question focuses on the Policy Enforcement Point and Logging components. This allows to determine actionability and timeliness of policy enforcement. Considering both passive and active enforcement strategies (modes). In the passive mode, data flow is inspected, but any policy violations are only flagged (alerted) by the Logging component. Conversely, the active strategy initiates policy related actions when data flow is inconsistent with its constraints.

On the other hand, the question addresses the Classification Endpoints and Labeling Management components. Essentially, this addresses how to define an appropriate structure for data labels and how to assign these data-relevant attributes to data flows, all while minimizing additional network traffic and computational burden.

When the data flow control check occurs?

This question allows to determine the various approaches to policy constraint checks that must be met for data flow before it can be accessed (read) by entities. These controls can be implemented statically, at runtime (dynamically), or through a hybrid approach.

In the static approach, data-flow control checks occur prior to execution—typically during formal analysis or compile time—to detect any illicit use of untrusted data, such as in Taint Analysis [25] or Security-Typed Languages. While this approach is effective for verifying compliance and ensuring the correctness of components, it lacks runtime information analysis, which is essential in complex and distributed environments. In contrast, the dynamic approach evaluates data-flow constraints during execution: the Tracking Engine monitors labels, and the Policy Enforcement Point determines flow permissions at runtime. When combined with the Audit and Logging component, this approach offers increased flexibility, although it does introduce some execution overhead. Lastly, the hybrid approach integrates both—static analysis reduces runtime processing demands, while dynamic checks address execution-time dependencies, particularly for label dimensions whose values cannot be assigned at production time and are influenced by environmental factors beyond the data producer's control.

4. Formal Model

In this section, we define our formal model for Secure and Context-Based Data Dissemination in Federated Special IoT Environments, denoted as *FedM*:

$$FedM = \langle N, D, S, \Pi, \mathcal{D}_A, DL, E, Caps, T, PF \rangle \quad (1)$$

To clarify our model, we will utilize an example that aligns well with the adopted high-level special IoT architecture depicted in Figure 1:

An analyst (operator) in the command and control post of Organization 1 (Org1) has been assigned the task of enhancing knowledge regarding environmental (terrain) terms and conditions in a specific area that falls under the jurisdiction of Organization 2 (Org2). In particular, both organizations form a federation structure. According to the cooperation agreement between these organizations, Org2 is allowed to share raw data from LiDAR laser scanner mounted on its unmanned aerial vehicle (UAV), but only for certain layers of orthophotos. Org2 is prohibited from sharing raw data from the Thermal Infrared spectrum, which captures temperature variations, as well as from the Shortwave Infrared layer, used for identifying mineral compositions and soil types. Therefore, Org2 may share fused and aggregated information derived from the Thermal Infrared spectrum, such as the total number of active vehicles identified. In contrast, Org1 is allowed to share data from its CCTV infrastructure, excluding footage that consists of any metadata about its belonging to enforcement agencies. Furthermore, Org1 can access Org2's GIS resources to enhance its data.

Our formal model, as outlined in Formula (1), is a complex structure of primitive elements and the dependencies that exist among them, which facilitates contextual, policy-driven data dissemination in distributed environments while emphasizing the protection of data flows and the reliability of their delivery. Later we will provide a detailed description of the individual components of our model, while Table 2 summarizes the elements of *FedM*.

A data network, N , serves as an equivalent representation of a federated environment, $N := FederatedEnv$, which comprises a finite set of federated organizations, $N = \{Org_1, Org_2, \dots, Org_n\}$. Furthermore, the mapping function, $N_{Org} : N \rightarrow \mathcal{P}(N)$, defines the federation partners for each organization. Here, the notation, $\mathcal{P}(N)$, denotes the power set (set of all subsets) of N . Specifically, $N_{Org}(Org_1) = \{Org_2, Org_n\}$, indicates that the first organization, Org_1 , can actively disseminate data to both Org_2 and Org_n . However, it is crucial to note that this dissemination is regulated by data policies, Π , established through mutual consensus between these organizations. In other words, the mapping function does not suffice for data dissemination; one of the necessary prerequisites is the establishment of a data policy, $\pi \in \Pi$ that clearly delineates which entities within the organization are allowed to access and process data, and to what extent, considering the context of the data.

Before providing a precise definition of the data policy, we would like to clarify that throughout the remainder of this article, we will refer to data as a primitive within a data set, $d \in D$. Each data element is structured as a three-element tuple, $d = \langle payload, headers, timestamp \rangle$. The *payload* can take on any structured or unstructured form that can later be processed and composed into IOs or AIOs. The *headers* contain relevant attributes that facilitate mechanisms for security, reliability, and contextual dissemination, while the *timestamp* reflects the time recorded by the producer entity, which can be employed to detect Time-based Replay Attacks.

Table 2. Formal model elements.

Name/Definition	Description
$N := FederatedEnv$ $N = \{Org_i, \dots, Org_j\}$ $N_{Org} : N \rightarrow \mathcal{P}(N)$	Federated data network that comprises a finite set of federated organizations. Furthermore, the function, N_{Org} , defines the federation partners for each organization.
$D := DataSet$	Set of all data flows, where element of this set, $d \in D$, is structured with payload, headers, and timestamp, $d = \langle payload, headers, timestamp \rangle$.

Table 2. Cont.

Name/Definition	Description
$\mathcal{S} = \{s_{id,v} \mid id \in ID_{\mathcal{S}}, v \in V_{id}\}$	Set of all data schemas that outline the structural and semantic prerequisites for the accurate interpretation and policy-driven dissemination of the data element, d , within the federated environment, N . In particular, each data schema is represented by a subset of various field types selected from a finite collection of all typed fields, $\mathcal{F}$, where $f_i = \langle name_i, type_i, range_i, norm_i \rangle \in \mathcal{F}$.
$\Pi = \{\pi_{id,v} \mid id \in ID, v \in V_{id}\}$	Set of all data policies, which are one of the necessary prerequisites of data dissemination within federated environments. Essentially, the policy element includes constraints, $\pi.\phi \subseteq \Phi$, and data actions, $\pi.\alpha \subseteq \mathcal{A}$, such as write, discard, or data processing.
$\mathcal{D}_{\mathcal{A}} : D \rightarrow (D \times \mathcal{P}(\mathcal{A}))$	Policy decision procedure (the PEP component) that states the dissemination actions, $\mathcal{P}(\mathcal{A})$, on the given data flow, d , according to the data-related policy, $\pi_{id,v} \in \Pi$.
$E := \text{Entities}$	Set of all entities responsible for the production, processing, and consuming of data in accordance with context-aware data policies constraints, Φ .
$DL := \text{DistributedLedger}$ $DL = \langle P, CH, CS, TX \rangle$	Organized decentralized structure, DL , which securely and reliably maintains a comprehensive dataset (channels), CH , of all registered data policies, Π , data schemas, $\mathcal{S}$, and an entity's unique characteristic (their fingerprints), FID . Moreover, access to these channels is regulated through dedicated programs, CS , where each operation on a channel is subject to auditing that provides tamper-resistant events chained together, TX , thereby enabling non-repudiation property.
$E_{\Pi} : E \rightarrow \mathcal{P}(\Pi)$	A mapping that assigns the policies to entities. Specifically, $E_{\Pi}(e) = \{\pi_n, \dots, \pi_m\}$, denotes that a subset of policies has been assigned to the entity, establishing the constraints for produced data.
$PF = TF \cup FF \cup AF$	Set of all processing functions, PF , which include transformation, TF , fusion, FF , and data quality assessment functions, AF . The first two result in modified data, and the last one yields new assessed values for the relevant data attributes.
$E_{PF} : E \rightarrow \mathcal{P}(PF)$	A mapping between entities, E , and processing functions, PF , where we propose that each entity is limited to employing a single function, $pf \in PF$.
$Caps = \{canWrite, canRead, canVerify, canEnforce, canProcess, canReassign\}$	Set of capabilities that can be utilized by entities: <i>canWrite</i> and <i>canRead</i> for data writing and reading; <i>canVerify</i> for verifying the authenticity and provenance of the data; <i>canEnforce</i> for utilizing the policy decision procedure; <i>canReassign</i> for re-assigning of data-relevant labels (Labeling Endpoint); and <i>canProcess</i> for facilitating various processing functions, PF .
$E_{Caps} : E \rightarrow \mathcal{P}(Caps)$	Function that specifies the actual subset of capabilities assigned to entities.
$T := \text{Topics}$	Set of all topics that isolate data at rest and enable the communication mechanism utilized by entities from the moment of data production, through provenance verification, policy enforcement, and any subsequent processing, until the delivery moment.
$E_T : E \rightarrow \mathcal{P}(T)$	The mapping function that specifies which topics can be utilized by which entities.
$T_{\delta_{active}, L} : T \rightarrow (\delta_{active} \times L)$	Function to delineate the scope of what data can be stored (isolated) on topics based on a data policy's active dimensions (data-attributes), $\delta_{active} \subseteq \Delta_{dims}$, and corresponding values, $\lambda \in L$.

Delving deeper, the primitive set $\mathcal{S} = \{s_{id,v} \mid id \in ID_{\mathcal{S}}, v \in V_{id}\}$ encompasses all data schemas that outline the essential structural and semantic prerequisites for the accurate interpretation and policy-driven dissemination of the data element, d , within the federated environment, N . Formally, each schema element $s_{id,v} = \{f_1, f_2, \dots, f_n \mid f_i = \langle name_i, type_i, range_i, norm_i \rangle \in \mathcal{F}\}$ is uniquely identified by a schema identifier $id \in ID_{\mathcal{S}}$ and a version $v \in V_{id}$. The data schema is represented by a subset of various field types selected from a finite collection of all field types, $f_i \in \mathcal{F}$. Essentially, the data schema defines the fields that any entity within the federation may encounter while processing a record generated under this particular schema. As such, each field is characterized as a four-element tuple, where $name_i$ serves as a unique field identifier, $type_i \in \{bytes, bool, enum, \dots, string\} = \mathcal{T}$ denotes the datatype, $range_i$ specifies the set of valid values for that type (with a wildcard range indicating any value), and $norm_i$ defines a normalization function that translates each encoded value back to its canonical representation, such as $norm_i : \{0, \dots, 100\} \rightarrow [0, 1]$.

For example, the sub-schema for a timestamp may be represented as $s_{id,v}.\mathcal{F}_t = \{\langle timestamp, long, \mathbb{Z}^+, \perp \rangle\}$, where the value $\perp$ indicates the absence of a normalization function. Furthermore, Formula (2) delineates the sub-schema $s_{id,v}.\mathcal{F}_p \in \mathcal{F}$ for the *payload* element, where $\langle frame_raw, bytes, \perp \rangle$ specifies that the content of the CCTV frame is of binary type. Conversely, the field $\langle direction, float, [0^\circ, 360^\circ], degree \rightarrow rad \rangle$ represents the horizontal direction of the camera relative to the true north, where the entity encodes it as a value expressed in degrees, and the normalization function subsequently converts degrees to radians.

Data Schema $s_{CCTV_{Org1},1}$: utilized to define CCTV footage payload structure

$$s_{CCTV,1}.\mathcal{F}_p = \left\{ \begin{array}{l} \langle frame_raw, bytes, \perp, \perp \rangle, \\ \langle geo_lat, float, [-90^\circ, 90^\circ], \perp \rangle, \\ \langle geo_lon, float, [-180^\circ, 180^\circ], \perp \rangle, \\ \langle angle, float, [0^\circ, 360^\circ], degree \rightarrow rad \rangle, \\ \langle direction, float, [0^\circ, 360^\circ], degree \rightarrow rad \rangle \end{array} \right\} \quad (2)$$

Unlike the timestamp and payload sub-schemas, which may be domain-specific, the header sub-schemas represent federation-level metadata. Their semantic heterogeneity across the federation, N , must remain domain-agnostic and formally regulated. This includes varying label tuple representations for the same label dimension. Further details regarding the header sub-schema will be elaborated alongside the definition of data policy.

Importantly, there is another property related to the heterogeneity of data among federated networks: a compatibility relation $\prec_S$. This relation is defined as a partial order across schema versions of the same identifier. Formula (3) states that the older schema $s_{id,v}$ is backward-compatible with the newly defined schema $s_{id,v'}$.

Backward Compatibility of Data Schema

$$s_{id,v} \prec_S s_{id,v'} \equiv s_{id,v}.\mathcal{F} \subseteq s_{id,v'}.\mathcal{F} \wedge \forall f \in s_{id,v}.\mathcal{F} : f \in s_{id,v'}.\mathcal{F} \quad (3)$$

This compatibility ensures that a consumer operating under the newer version can accurately interpret records generated by the older version, thus allowing federated organizations to negotiate schema compatibility without necessitating strict version synchronization.

In detail, the set notation, $\Pi = \{\pi_{id,v} \mid id \in ID, v \in V_{id}\}$, represents all structured policy elements, $\pi_{id,v} = \langle id, v, \mu_{id,v}, \delta_{active_{id,v}}, \lambda_{id,v}, \theta_{id,v}, \phi_{id,v}, \rho_{\lambda_{id,v}, \phi_{id,v}}, \alpha_{id,v}, \prec_{\alpha_{id,v}}, \gamma_{\alpha_{id,v}, \phi_{id,v}} \rangle$, where:

- $id \in ID$ is the unique policy identifier that enables multiple policy instances with the same label tuple element;
- $v \in \mathbb{N}_1$ represents the policy version number, enabling the evolution (redefinition) of policies over time;
- $\mu = \langle issuer, status, validity_{window}, description \rangle$ is a policy element that represents the structured metadata, which governs the lifecycle of the policy itself;
- $\delta_{active} \subseteq \Delta_{dims} = \{secrecy, category_{obj}, category_{colors}, integrity, \dots, trust\}$ is the subset of active dimensions names, δ_{active} , selected from the set of all possible dimensions names, Δ_{dims} . Furthermore, L_{dim} defines the space of values for the specific dimension name, $dim \in \Delta_{dims}$. Apart from a fixed (discrete) format, $L_{dim} = \{v_1, v_2, \dots, v_n\}$, the active dimension can be defined over a continuous, $L_{dim'} = \mathbb{R}$, or within a bounded range, $L_{dim''} = \{x \in \mathbb{R} \mid a < x < b\}$. Notably, for federated organizations wishing to actively disseminate data using these dimensions and their respective values, it is imperative to establish a federation-level semantic agreement (contract). For instance, let us assume that both organizations mentioned above have agreed on three active dimensions: secrecy, object category, and trust,

$\delta_{active} = \{secrecy, category_{obj}, trust\}$. The secrecy dimension comprises two values: public and confidential, $L_{secrecy} = \{public, confidential\}$. Conversely, the category dimension includes labels that describe LiDAR laser scanner data and CCTV data, $L_{category_{obj}} = \{LiDAR, CCTV\}$. Finally, a bounded dimension, is expressed as $L_{trust} = [0, 1]$. However, even with a semantic contract in place, syntactic heterogeneity in dimension encoding may emerge across organizations. For example, both organizations might encode trust labels with differing numeric precisions or use distinct controlled vocabularies for category labels. Consequently, the header sub-scheme $s_{id,v} \cdot \mathcal{F}_h$ must implement a normalization function. Essentially, the headers sub-schema provides a formal syntactic contract that addresses encoding discrepancies, whereas the federation-level dimension space agreement ensures the semantic equivalence of these dimensions, $\mathcal{F}_h^{\delta_{active}}$. In contrast, the organization-level dimension governance, $\mathcal{F}_h^{Org}$, outlines the semantic requirements for local security, reliability, and contextual dissemination mechanisms that are not included in the federation-wide dimension agreement, $s_{id,v} \cdot \mathcal{F}_h = \mathcal{F}_h^{\delta_{active}} \cup \mathcal{F}_h^{Org}$.

- λ is an element of the Cartesian product, $L = \prod_{dim \in \delta_{active}} (L_{dim} \cup \{*\} \cup \{\perp\})$, which defines the space of all possible values of the active dimensions, δ_{active} . Additionally, $\{*\}$ denotes a wildcard value that accepts any value from the specified dimension space, while $\{\perp\}$ indicates an unset value. Consequently, considering the aforementioned secrecy and category dimensions, the Cartesian product of these two dimensions is represented in the following manner: $L = \{(public, CCTV), (public, LiDAR), (confidential, CCTV), (confidential, LiDAR), (public, *), (confidential, *), (*, CCTV), (*, LiDAR), \dots, (*, *), (\perp, \perp)\}$. Furthermore, $Label_{\mathcal{E}}(\lambda, dim) \in L_{dim} \cup \{*\} \cup \{\perp\}$ is the label lookup function that extracts from the λ element the value assigned to each active dimension by name. Therefore, the expression, $Label_{\mathcal{E}}(\lambda, secrecy) = confidential \wedge Label_{\mathcal{E}}(\lambda, category_{obj}) = CCTV$, for brevity, can be represented as a tuple of per-dimension values, one per active dimension, $\lambda = (confidential, CCTV) \in L$. For clarity, $\lambda_{\pi} \in L$ denotes the tuple within a policy instance, while $\lambda_d \in L$ indicates the specific tuple that was assigned to the data headers, $d.headers.policy.labels$. Notably, for each active dimension, $dim \in \delta_{active}$, we define a partial order, $\preceq_{dim}$, over, $L_{dim} \cup \{*\} \cup \{\perp\}$, which induces a comparability relation without restricting the underlying value space. This yields a product order over labels, $\lambda \in L$, defined as $\lambda_1 \preceq \lambda_2 \equiv \forall dim \in \delta_{active} : Label_{\mathcal{E}}(\lambda_1, dim) \preceq_{dim} Label_{\mathcal{E}}(\lambda_2, dim)$, where Comparability is enforced solely for dimensions with a defined ordering, while other cases are managed through dimension-specific matching semantics. A total order is not a requirement; thus, certain elements may be incomparable, especially in non-ordinal domains. Additionally, the ordering semantics are established for each type of dimension, with total orders applied where appropriate, such as with $L_{secrecy}$ and L_{trust} . In contrast, categorical dimensions, such as $L_{category_{obj}}$, rely on equality or wildcard-based matching. Consequently, special elements are consistently treated as $\perp \preceq_{dim} \lambda \preceq_{dim} *$;
- $\theta \subseteq \Theta$ determines a finite subset of all parameter contexts, Θ , that define the quantitative and qualitative thresholds governing constraint evaluation. Each parameter context is modeled as, $\vartheta = \langle name, value, validity_{window} \rangle$. The *value* element may be drawn either from the canonical federation value space, L_{dim} , or from a non-dimensional value space determined within the data schema, $s_{id,v}$. Additionally, each parameter context may be classified as either static or time-indexed. For instance, a static Shannon entropy threshold can be expressed as $\langle entropy_max, 6.5, \perp \rangle \in \theta$, where $\perp$ denotes the absence of a validity window. In contrast, a time-dependent trust threshold applicable during nighttime hours can be defined as $\langle trust_min, trust_{val}, [\tau_{dusk}, \tau_{sunrise}] \rangle \in \theta$, where $trust_{val} = 0.75 \in L_{trust}$.

- $\phi \subseteq \Phi$ denotes the data constraint subset, where Φ is the set of all abstract context-aware constraint predicates (functions), $\Phi = \{\varphi \mid \varphi : D \rightarrow \{true, false\}\}$, without reference to any specific field vocabulary or schema context. Notably, the set Φ constitutes the federation-wide reusable vocabulary of constraint logic. This allows to define the abstract constraint once, which may be instantiated across multiple schema contexts and policy deployments without redefinition. For example, $\phi = \{\varphi_0, \varphi_1, \varphi_2, \varphi_3\} \subset \Phi$ defines the constraint subset, where φ_0 is equivalent to the constraint of transformed (enhanced) data with identified objects detected by their thermal signatures (heat emissions); φ_1 denotes fused (aggregated) data derived from the Thermal Infrared spectrum; φ_2 specifies that data obtained from video surveillance cameras must not contain any metadata regarding its law enforcement source; and φ_3 requires that the data payload must exceed a predetermined entropy threshold, indicating that the flow is encrypted. However, in order to evaluate a concrete data primitive, such as data produced by a CCTV entity or LiDAR scanner taking into consideration structural heterogeneity of these data, the process denoted as schema binding must be employed. It binds an abstract constraint, φ , to a specific schema, $s_{id,v}$, producing a schema-bound predicate, φ_S , that knows exactly which fields to read, how to deserialize them, and what value ranges are valid. Additionally, this schema-bound ensures that constraints cannot access fields outside the declared vocabulary. Notably, along with schema binding, constraints can be parametrized in a dynamic and evolvable manner by imposing thresholds regulated through a policy parameter contexts subset, $\theta \subseteq \Theta$. Therefore, we define the schema- and context-bounded constraint set as $\Phi_S = \{\varphi_S \mid \varphi_S : (D \times \mathcal{S} \times \Theta) \rightarrow \{true, false\}\}$. Each element is a boolean-valued predicate that takes a data element $d \in D$, a schema element $s_{id,v} \in \mathcal{S}$, and a parameter context subset $\theta \subseteq \Theta$, as arguments, resolving both the accessible field vocabulary and the constraint threshold values at evaluation time. Consequently, the same abstract entropy constraint, φ_3 , may be bound into two distinct data schema, such as for CCTV and LiDAR data, where the predicate logic is identical but the field resolution differs according to the respective payload sub-schemas, $s_{CCTV,1} \cdot \mathcal{F}_p \neq s_{LiDAR,1} \cdot \mathcal{F}_p$.
- $\rho_{\lambda,\phi}$ delineates the relationship between each active label dimension value and the corresponding subset of data constraints associated with that value. Although it can be stated that there is an implicit mapping between these two policy structure elements, we propose this explicit relation to enhance governance and auditability. For instance, the implicit mapping may be represented as follows: the label tuple, $(confidential, CCTV)$, requires that the data complies with a specific subset of constraints, such as $\{\varphi_2, \varphi_3\} \subset \Phi$. This implies that CCTV data must be encrypted (φ_3) and must not contain any metadata regarding its law enforcement source (φ_2). However, this relation is quite general and fails to define the direct relationship; specifically, it does not clarify that the value of the secrecy dimension label ($confidential \in L_{secrecy}$) necessitates the encryption of the data, while the category dimension ($CCTV \in L_{category_{obj}}$) demands the sanitization of metadata regarding the provenance from law enforcement agencies;
- $\alpha \subseteq \mathcal{A}$ specifies the subset of policy actions (operations), where the function $Type_{\mathcal{A}} : \mathcal{A} \rightarrow Category_{\mathcal{A}}$ returns the action type, which typically falls into the following categories:
 - *write*: actions for writing (disseminating) data that delineate how data at-rest is stored, allowing for later access by consumers, e.g., the operation, $write_{enc}$, that specifies to disseminate encrypted data to the secure store;
 - *discard*: operations for discarding data along with the data removal reason, such as operation, $discard_{enc}$, which removes the entirety of data because it was en-

- encrypted. Notably, we will adhere to the operation, $discard_{payload}$, as the general deny (discard) invariant, which is equivalent to an empty action set, $\alpha = \emptyset$;
- *log*: data flow logging actions that track the occurrence of data, e.g., monitoring and alerting operations that ensure accountability and identified policy violations—specifically, compliance with data constraints ($log_{\varphi_i^+}$) or failures ($log_{\varphi_i^-}$);
 - *fusion*: data modification operations, where multiple data payloads are combined to obtain enhanced insights, e.g., the aggregation, $fusion_{LiDAR}$, that merges data from the Thermal Infrared spectrum obtained through LiDAR sensors to determine the number of vehicles turned on within a specified time interval;
 - *transform*: data modification operations on a single data payload, e.g., the enrichment operation, $transform_{GIS}$, that enhances each CCTV frame by appending a data mapping from the GIS database to identified objects;
 - *assess*: data quality assessment operations, which evaluate the quality of data payloads in relation to the Data Quality Requirements—expressed through the attributes of the Data Quality Dimensions, e.g., the consistency ratio function, which checks if data is compliant with a pattern and to what extent, or the uniqueness ratio function that trails for duplicated data;
 - *compose*: chain operations from various processing categories in a sequenced manner, e.g., the chained action, $compose_{tf_{enc} \circ ff_{obj}} \triangleq transform_{enc} \circ fusion_{objects}$, which first applies the fusion operation, $fusion_{objects}$, then proceeds to encrypt (transform) the outcome, $transform_{enc}$.
- $\prec_\alpha$ is a set of irreflexive, asymmetric, and transitive binary (precedence) relations defined over the action subset, $\alpha \subseteq \mathcal{A}$. In this context, $action_i \prec action_j$, signifies that $action_i$ must precede $action_j$. For instance, the set of precedence relations, $\prec_\alpha = \{(discard_i, write_j), (discard_i, transform_k)\}$, indicates that the discard action takes precedence over both the write and transform operations. Furthermore, the absence of a precedence relation, $\neg(action_i \prec action_j) \wedge \neg(action_j \prec action_i)$, implies that these actions may occur concurrently, such as discarding and logging. However, certain actions are semantically incompatible, meaning they cannot be executed concurrently on the same data flow without risk, such as the write and discard action pair. To safeguard against unsafe data flows, we propose an invariant that ensures the precedence relation is definitive for all incompatible pairs. In other words, for any incompatible actions $action_i, action_j \in \alpha$, it must hold that either $action_i \prec_\alpha action_j \vee action_j \prec_\alpha action_i$. Consequently, the absence of a precedence relation is permitted only for semantically compatible actions.
 - $\gamma_{\alpha, \Phi}$ defines the relationship between the subset of data constraints and the subset of actions. Specifically, $\gamma : \Phi \rightarrow (\mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A}))$ is a function that maps each constraint, $\varphi \in \Phi$, to a pair of action subsets. The first subset of actions in this pair, $\gamma^+(\varphi) \subseteq \alpha$, indicates the actions that are invoked when the constraint predicate is satisfied, $\varphi(d) = true$. Conversely, $\gamma^-(\varphi) \subseteq \alpha$, specifies the subset of actions that should be executed if the constraint function returns false, $\varphi(d) = false$.

In light of the proposed data policy elements and the aforementioned use case scenario, let us provide concrete examples involving different policies. Consider that there exists a unique set of identifiers, $ID = \{UAV_{LiDAR}, CCTV_{glare}\}$, and that the policy for CCTV surveillance, $CCTV_{glare}$, has evolved over time, leading to the proposal of its second version, $V_{CCTV_{glare}} = \{1, 2\}$. The Formula (4) defines the example data policy, $\pi_{UAV_{LiDAR}, 1}$, which applies to the LiDAR laser scanner data used for detecting thermal signatures.

The producer entity labels its data with the tuple (*confidential*, *LiDAR*). As a result, it must comprise encrypted aggregated data of the detected objects from the Thermal Infrared spectrum, $\{\varphi_0, \varphi_1, \varphi_3\}$. Notably, an entropy threshold, $entropy_{max}$, for the con-

straint φ_3 is set at 7.5. Therefore, the applicable actions subset on this data encompass writing the encrypted data ($write_{enc}$), detecting thermal objects, ($transform_{detect}$), executing a chained action ($compose_{tf_{enc} \circ ff_{thermal}}$)—such as fusion ($fusion_{thermal}$) combined with an encryption transformation ($transform_{enc}$)—along with discarding unencrypted data ($discard_{plain}$) and logging the outcomes of constraints (log_{φ}). Moreover, the mapping $\rho_{\lambda_{id,v}, \varphi_{id,v}}(confidential) = \{\varphi_3\}$ indicates that the secrecy dimension label value necessitates data encryption, while the category label $\rho_{\lambda_{id,v}, \varphi_{id,v}}(LiDAR) = \{\varphi_0, \varphi_1\}$ denotes that the data must aggregate identified objects detected by their heat emission. Furthermore, the relation between actions and constraints highlights that the subset of actions corresponding to the satisfied constraint φ_0 is limited to successful constraint logging, $\gamma^+(\varphi_0) = \{log_{\varphi_0^+}\}$. The same action applies to constraint φ_1 , $\gamma^+(\varphi_1) = \{log_{\varphi_1^+}\}$. In contrast, for constraint φ_3 , the action subset is restricted to encrypted data writing, $\gamma^+(\varphi_3) = \{write_{enc}\}$. Conversely, if the evaluation function returns a false value, the actions for constraint φ_0 would then include transformation for object detection along with logging, $\gamma^-(\varphi_0) = \{transform_{detect}, log_{\varphi_0^-}\}$. For constraint φ_1 , the action set would consist of the chained operation and logging, $\gamma^-(\varphi_1) = \{compose_{tf_{enc} \circ ff_{thermal}}, log_{\varphi_1^-}\}$. Lastly, for constraint φ_3 , if the data is not encrypted, the applicable action set would include the unencrypted data discard operation and log the constraint failure, $\gamma^-(\varphi_3) = \{discard_{plain}, log_{\varphi_3^-}\}$.

Policy $\pi_{UAV_{LiDAR},1}$: Thermal signature detection using LiDAR laser scanner

$$\begin{aligned}
 id &= UAV_{LiDAR} \in ID, v = 1 \in V_{UAV_{LiDAR}}, \delta_{active} = \{secrecy, category_{obj}\} \\
 \lambda_{id,v} &= (confidential, LiDAR), \varphi_{id,v} = \{\varphi_0, \varphi_1, \varphi_3\} \subset \Phi, \theta_{id,v} = \{\langle entropy_{max}, 7.5, \perp \rangle\} \\
 \alpha_{id,v} &= \{write_{enc}, transform_{detect}, compose_{tf_{enc} \circ ff_{thermal}}, discard_{plain}, \\
 &\quad log_{\varphi_0^+}, log_{\varphi_0^-}, log_{\varphi_1^+}, log_{\varphi_1^-}, log_{\varphi_3^-}\} \subset \mathcal{A}, \text{ where } compose_{tf_{enc} \circ ff_{thermal}} \triangleq \\
 &\quad \triangleq (transform_{enc} \circ fusion_{thermal}) \\
 \rho_{\lambda_{id,v}, \varphi_{id,v}}(confidential) &= \{\varphi_3\} \wedge \rho_{\lambda_{id,v}, \varphi_{id,v}}(LiDAR) = \{\varphi_0, \varphi_1\} \\
 \gamma_{\alpha_{id,v}, \varphi_{id,v}}(\varphi_0) &= (\gamma^+(\varphi_0), \gamma^-(\varphi_0)) = (\{log_{\varphi_0^+}\}, \{transform_{detect}, log_{\varphi_0^-}\}) \wedge \\
 \gamma_{\alpha_{id,v}, \varphi_{id,v}}(\varphi_1) &= (\gamma^+(\varphi_1), \gamma^-(\varphi_1)) = (\{log_{\varphi_1^+}\}, \{compose_{tf_{enc} \circ ff_{thermal}}, log_{\varphi_1^-}\}) \wedge \\
 \gamma_{\alpha_{id,v}, \varphi_{id,v}}(\varphi_3) &= (\gamma^+(\varphi_3), \gamma^-(\varphi_3)) = (\{write_{enc}\}, \{discard_{plain}, log_{\varphi_3^-}\}) \\
 \prec_{\alpha_{id,v}} &= \{(transform_{detect}, discard_{plain}), (transform_{detect}, compose_{tf_{enc} \circ ff_{thermal}}), \\
 &\quad (compose_{tf_{enc} \circ ff_{thermal}}, discard_{plain}), (compose_{tf_{enc} \circ ff_{thermal}}, log_{\varphi_0^+}), (log_{\varphi_0^+}, discard_{plain}), \\
 &\quad (write_{enc}, compose_{tf_{enc} \circ ff_{thermal}}), (write_{enc}, transform_{detect})\} \\
 \mu_{id,v} &= \langle Org2, active, [1 \text{ May } 2026 \text{ 12:00 UTC}, 1 \text{ June } 2026 \text{ 12:00 UTC}], description \rangle
 \end{aligned}
 \tag{4}$$

An examination of the aforementioned data policy—particularly the subset of resulting actions—reveals a significant concern regarding the deconfliction of these operations. Specifically, when data is only transformed by the thermal object detection function ($transform_{detect}$), the outcome actions, such as the composed operation ($compose_{tf_{enc} \circ ff_{thermal}}$) and data removal ($discard_{plain}$), become mutually exclusive. To address this, we proposed a deconfliction resolution process as part of the state-transition decision procedure, $\mathcal{D}_{\mathcal{A}} : D \rightarrow (D \times \mathcal{P}(\mathcal{A}))$ (defined by Formula (5)), which incorporates a set of precedence relations, $\prec_{\alpha}$, while also considering the history of actions already applied, $\mathcal{H}_d$.

Our procedure, $\mathcal{D}_{\mathcal{A}}$, acts as the Policy Enforcement Point, which is a tamper-resistant oracle that delineates the subset of policy actions, $\gamma(\varphi) \in \mathcal{P}(\mathcal{A})$, that will be performed on the given data, $d \in D$, in accordance with the current data policy, $\pi_{id,v} \in \Pi$. In general, it involves three main steps: (1) selection of applicable policies; (2) evaluation of constraint

predicates; and (3) resolution of conflicting actions. Notably, at any given time, τ , the procedure operates under a single active policy, $\pi_{id,v} \in \Pi$, uniquely identified by the data flow. For clarity, we will write the resulting action subset before the deconfliction process as $\alpha_{constraints} \triangleq \alpha^+ \cup \alpha^-$, which reflects the union of both the actions permitted by satisfied constraints, $\alpha^+ \triangleq \bigcup \{\pi_{id,v} \cdot \gamma^+(\varphi) \mid \varphi \in \pi_{id,v} \cdot \Phi, \text{ where } \varphi(d) = true\}$, and those enforced due to constraint violations, $\alpha^- \triangleq \bigcup \{\pi_{id,v} \cdot \gamma^-(\varphi) \mid \varphi \in \pi_{id,v} \cdot \Phi, \text{ where } \varphi(d) = false\}$. Conversely, a subset of active actions after deconfliction will be denoted as α_{active} . Consequently, the procedure $\mathcal{D}_A$ consistently produces a pair $\mathcal{D}_A(d) \triangleq (d', \alpha_{active})$. If the condition $\mathcal{C}(d, \pi_{id,v})$ is satisfied, then α_{active} represents the subset of active actions after deconfliction, and data flow, d' , indicates the data, d , with its history field updated. Otherwise, we define $\alpha_{active} \triangleq \emptyset$ and $d' \triangleq d$, reflecting the general discard invariant where the history field remains unchanged.

Moreover, through our Lemmas 1–4, we have demonstrated that our procedure is both deterministic (Lemma 3) and terminating (Lemma 4). Additionally, it facilitates the evolution of the resulting policy actions, α_{active} , through historical awareness, $\mathcal{H}_d$ (Lemma 1), and a complete deconfliction process (Lemma 2). Furthermore, by combining these lemmas, the Policy-based Data Flow Preservation Theorem 1 can be established. This theorem ensures that all possible executions (data paths) comply with the active policy action set while simultaneously excluding implicit data paths, thereby guaranteeing that data is accurately delivered.

Policy Decision (Enforce) State-Transition Procedure— $\mathcal{D}_A$

$$\begin{aligned} &\mathcal{D}_A : D \rightarrow (D \times \mathcal{P}(\mathcal{A})) \\ &\forall d \in D : \mathcal{D}_A(d) \triangleq (d', \alpha_{active}) \\ &\quad \text{Let: } \pi_{id,v} \triangleq \text{Policy}_{\mathcal{E}}(d.headers.policy.id, d.headers.policy.version) \\ &\quad \text{Condition: } \mathcal{C}(d, \pi_{id,v}) \triangleq (d.timestamp \in \pi_{id,v} \cdot \mu.validity_{window}) \wedge \\ &\quad \quad \quad \wedge \pi_{id,v} \cdot \mu.status = active \wedge |d.headers.policy.labels| = |\pi_{id,v} \cdot \delta_{active}| \\ &\quad \text{If } \mathcal{C}(d, \pi_{id,v}) \text{ holds, then evaluate predicates and resolve conflicts:} \\ &\quad \quad \alpha_{constraints} \triangleq \bigcup \{\pi_{id,v} \cdot \gamma^+(\varphi) \mid \varphi \in \pi_{id,v} \cdot \Phi, \text{ where } \varphi(d) = true\} \cup \\ &\quad \quad \quad \cup \bigcup \{\pi_{id,v} \cdot \gamma^-(\varphi) \mid \varphi \in \pi_{id,v} \cdot \Phi, \text{ where } \varphi(d) = false\} \\ &\quad \quad \mathcal{H}_d \triangleq d.headers.policy.history \\ &\quad \quad \alpha_{\mathcal{H}_d} \triangleq \text{ActionHistory}_{\mathcal{E}}(\mathcal{H}_d, \pi_{id,v}.id, \pi_{id,v}.v) \\ &\quad \quad \alpha_{active} \triangleq \text{Deconfliction}(\alpha_{constraints}, \alpha_{\mathcal{H}_d}, \pi_{id,v} \prec_{\alpha}) \\ &\quad \text{Update the data element history:} \\ &\quad \quad d' \triangleq d[headers.policy.history \mapsto \mathcal{H}_d \bullet \langle \pi_{id,v}.id, \pi_{id,v}.v, \alpha_{active} \rangle] \\ &\quad \text{If } \mathcal{C}(d, \pi_{id,v}) \text{ does not hold, then general discard invariant:} \\ &\quad \quad \quad \alpha_{active} \triangleq \emptyset, d' \triangleq d \\ &\quad \text{Finally, for all } d \in D \text{ return:} \\ &\quad \quad \mathcal{D}_A(d) \triangleq (d', \alpha_{active}) \end{aligned} \tag{5}$$

Lemma 1 (History-awareness of $\mathcal{D}_A$). Given a single active policy, $\pi_{id,v} \in \Pi$, assigned to data, $d \in D$, an irreflexive, transitive, and asymmetric precedence relation, $\prec_{\alpha}$, over action subset, $\alpha \subseteq \mathcal{A}$, and deterministic context-aware constraints evaluation function, $\Phi = \{\varphi \mid \varphi : D \rightarrow \{true, false\}\}$, if one or more resulting actions, $\alpha_{constraints}$, reference history, $\mathcal{H}_d$, then for the same,

$d \in D$, two executions of the deconfliction process with distinct histories, $\mathcal{H}_d \neq \mathcal{H}'_d$, may yield different active action sets, $\alpha_{active} \neq \alpha'_{active}$. Consequently, this enables history-aware evolution of resulting actions within the scope of utilized data policy without modifying its definition.

Proof. By construction, consider that one of the resulting actions of predicate function, $\alpha_{constraints}$, references $\mathcal{H}_d$. Let $\mathcal{H}_d$ and $\mathcal{H}'_d$ be two distinct histories of already applied actions. Since $\alpha_{constraints}$ is evaluated (deconflicted) against $\mathcal{H}_d$ in the first procedure execution and $\mathcal{H}'_d$ in the second, and since $\mathcal{H}_d \neq \mathcal{H}'_d$, it is possible that outcome α_{active} is different under $\mathcal{H}_d$ than α'_{active} under $\mathcal{H}'_d$. Crucially, neither $\pi_{id,v}.\lambda$ nor $\pi_{id,v}.\phi$ is modified between executions—the behavioral evolution of the $\mathcal{D}_A$ arises only from the change in history without structural modification of the active policy. Notably, if both executions yield empty set, $\alpha_{active} = \emptyset$, the discard invariant applies uniformly, preserving the general deny guarantee regardless of history. $\square$

Lemma 2 (Deconfliction Step Completeness). *For any two actions, $action_i$ and $action_j$ belonging to $\alpha_{constraints}$, the deconfliction step is complete. Conflicting actions are resolved uniquely through the precedence relation $\prec_\alpha$, while incomparable and compatible action pairs, $\neg(action_i \prec action_j) \wedge \neg(action_j \prec action_i)$, are admitted as concurrent within the outcome action subset, α_{active} .*

Proof. Through exhaustive case analysis of all action pairs within $\alpha_{constraints}$ —with all actions originated exclusively from the single active policy, $\pi_{id,v} \in \Pi$ —any potential inter-policy conflicts are eliminated. Moreover, this analysis reveals three distinct cases. The asymmetry of $\prec_\alpha$ guarantees uniqueness within the first two cases. In the third case, the absence of a relationship between pairs indicates no conflict—both actions are independent and may occur concurrently. Since these three cases are mutually exclusive and exhaustive across all pairs in $\alpha_{constraints}$, the deconfliction step is complete—every pair is either resolved through $\prec_\alpha$ or admitted as concurrent. $\square$

Lemma 3 (Determinism of $\mathcal{D}_A$). *Given a single active policy, $\pi_{id,v} \in \Pi$, which is associated with data, $d \in D$ —consisting of an action history sequence, $\mathcal{H}_d$ (Lemma 1)—a precedence relation $\prec_\alpha$ (Lemma 2), and a deterministic constraint evaluation function, $\Phi = \{\varphi \mid \varphi : D \rightarrow \{\text{true}, \text{false}\}\}$. Under these conditions, $\mathcal{D}_A(d)$ states a unique subset of actions from the policy's action set, $\alpha_{active} \subseteq \pi_{id,v}.\alpha$.*

Proof. (1) Policy selection is unambiguous, since is uniquely bound to data flow, d . (2) The deterministic evaluation of predicates against the data, d , results in a unique action subset, $\alpha_{constraints}$. Finally, (3) the deconfliction step uniquely resolves $\alpha_{constraints}$ against history, $\mathcal{H}_d$ (Lemma 1), and the precedence relation, $\prec_\alpha$ (Lemma 2), ensures that no conflicting actions are permitted to occur concurrently. Notably, the empty active action subset denotes the discard invariant, thus addressing the only remaining undefined scenario. Consequently, $\mathcal{D}_A$ is a deterministic procedure. $\square$

Lemma 4 (Termination of $\mathcal{D}_A$). *Given a single active policy, $\pi_{id,v} \in \Pi$, which includes a finite policy action set $\pi_{id,v}.\alpha \subseteq \mathcal{A}$, a subset of constraints, $\pi_{id,v}.\phi \subseteq \Phi$, a precedence relation, $\pi_{id,v}.\prec_\alpha$, and a finite action history sequence, $\mathcal{H}_d$. Under these conditions, $\mathcal{D}_A(d)$ terminates in finite time, τ , for any data, $d \in D$.*

Proof. (1) Policy selection requires only a single lookup, thus terminating trivially. (2) The constraint evaluation iterates over the predicates, $\pi_{id,v}.\phi$, bounding the number of evaluations to those concerning data only, $d \in D$. Moreover, the outcome action subset is

finite $\alpha_{constraints} \subseteq \pi_{id,v}.\alpha$. (3) The deconfliction step operates over the finite subset of actions and against the finite history sequence, $\mathcal{H}_d$, following the precedence relation, $\prec_\alpha$. By considering a naive (exhaustive) upper bound of precedence pairs ($|\alpha_{constraints}|^2$), each resolution step is a single constant-time lookup. Ultimately, the general discard invariant check entails a single evaluation after the deconfliction process, which also trivially terminates. Therefore, $\mathcal{D}_A(d)$ terminates in finite time. $\square$

Theorem 1 (Policy-based Data Flow Preservation). *Let $d \in D$ be data processed under a single active policy, $\pi_{id,v} \in \Pi$, which includes a finite action subset, $\pi_{id,v}.\alpha \subseteq \mathcal{A}$, a subset of constraints, $\pi_{id,v}.\phi \subseteq \Phi$, a precedence relation, $\pi_{id,v}.\prec_\alpha$, and a finite action history sequence, $\mathcal{H}_d$, bounded to that data flow. The decision procedure, $\mathcal{D}_A(d)$ preserves policy-based data flow by enforcing only those actions specified by data policy, $\alpha_{active} \subseteq \pi_{id,v}.\alpha$, ensuring that no implicit operation occur and that no behavioral changes beyond the scope of $\mathcal{H}_d$ are taken into account.*

Proof. By Lemma 4 (Termination), $\mathcal{D}_A$ is guaranteed to terminate in finite time for any finite set of data policies, Π , ensuring that the procedure consistently yields results. Furthermore, as established by Lemma 2 (Deconfliction Step Completeness) and Lemma 3 (Determinism), all enforced actions remain contained within the action subset exclusively defined by the given data policy, $\pi_{id,v}.\alpha$. Notably, the general deny guarantee (discard invariant) ensures that no data action is executed unless explicitly defined by the corresponding data policy. Finally, according to Lemma 1 (history-awareness), the behavioral evolution of $\mathcal{D}_A(d)$ is solely driven by changes in history, without any structural alterations to the active policy. Consequently, $\mathcal{D}_A$ preserves policy-based data flow under all conditions—actions are exclusively delineated through $\pi_{id,v}$, the deny invariant is guaranteed in the absence of explicit actions, and behavioral evolution is strictly bounded by $\mathcal{H}_d$ without alteration of the data policy structure. $\square$

At the beginning of the execution of the decision procedure, policy-related elements are extracted from the data headers, represented as $d.headers.policy = \langle id, version, labels, history \rangle$. These elements include the unique identifier for the policy, $d.headers.policy.id$, and its version number, $d.headers.policy.version$, along with the tuple of per-dimension values, $d.headers.policy.labels$, and a sequenced audit trail of applied actions, $d.headers.policy.history$. The first two elements of the policy header determine the specific versioned policy that the entity utilized to produce its payload, $d.payload$.

Following this, the policy definition is retrieved from the secure store (the Policy Specification Module) using these elements, $\pi_{id,v} = Policy_{\mathcal{E}}(id, version)$. As the secure store, we proposed a dedicated structure, referred to as a Distributed Ledger, $DL := DistributedLedger$, which reliably maintains a comprehensive dataset of all registered data policies. We model this store as organized structure, $DL = \langle P, CH, CS, TX \rangle$, where $P := Peers$ represents a set of peers with invariant that each peer, $p \in P$, holds an identical dataset, referred to as a channel, $ch \in CH$. This ensures redundancy, accessibility, and resilience of the data stored on channels within the federated environment, N . Furthermore, access to these sensitive channels is regulated through dedicated programs known as smart contracts (chaincodes), CS . Each execution of a chaincode is subject to auditing, which provides tamper-resistant events related to operations on the ledger channel, referred to as ledger transactions, $\{tx_i, \dots, tx_j\}$. This, combined with the mechanism that chains (links) transactions together, creates an auditing trail of events called a transaction log, $TX = [tx_0, tx_1, \dots, tx_k]$, thereby enabling non-repudiation for policy registration. The specific details of the policy registration procedure within the DLT, $registerPolicyDL$, along with the definition of the dedicated chaincode, $retrivePolicy_{CS} \in CS$, that retrieves data policies from channel, are beyond the scope of this article.

After the policy is retrieved, its activeness is verified, $\pi_{id,v}.\mu.status$, and it is checked whether the produced data timestamp, $d.timestamp$, falls in the policy's valid time window, $\pi_{id,v}.\mu.validity_{window}$. Subsequently, a cardinality check is conducted, comparing the number of label values, $|d.headers.policy.labels|$, against the number of policy active dimensions, $|\pi_{id,v}.\delta_{active}|$. This countermeasure reduces the attack surface, where a malicious producer may supply more values than the policy has active dimensions. If the cardinality does not match or if the producer's data policy is obsolete, the procedure execution is terminated, resulting in an empty action set, $\emptyset \equiv \{discard_{payload}\}$.

Conversely, if all checks are successful, the procedure proceeds to the core step of determining the subset of actions to be performed on the data, $\alpha_{active} \subseteq \pi_{id,v}.\alpha$. Ultimately, the deconfliction process is initiated, $Deconfliction(\alpha_{constraints}, \alpha_{\mathcal{H}_d}, \pi_{id,v} \prec_{\alpha})$. This procedure resolves conflicting actions based on established precedence relations, $\prec_{\alpha}$, while also considering the history of actions already applied concerning this specific policy, $\alpha_{\mathcal{H}_d} \triangleq ActionHistory_{\mathcal{E}}(\mathcal{H}_d, \pi_{id,v}.id, \pi_{id,v}.v)$. If, following deconfliction, no additional actions can be applied to the data, the policy enforcement procedure mandates the general discard invariant, thereby addressing potential action cycles. In all other scenarios not previously described, the procedure will also state an entirety data discard action. Subsequent to the deconfliction process, the policy history's data header is updated with a structure that includes the enforced policy identifier, version, and the resulting active action subset, $d' \triangleq d[headers.policy.history \mapsto \mathcal{H}_d \bullet \langle \pi_{id,v}.id, \pi_{id,v}.v, \alpha_{active} \rangle]$.

The Formula (6) outlines the decision procedure resulting action set after the deconfliction process for the example policy, $\pi_{UAV_{LiDAR},1}$. Consider that the data payload, $d.payload$, produced by a LiDAR laser scanner was only transformed by the thermal object detection function, $\varphi_0(d) = true$. Therefore, the constraints $\varphi_1(d)$ and $\varphi_3(d)$ are not satisfied, this implies mutually exclusive actions, necessitating the concurrent aggregation and discarding of data, $\alpha_{constraints} = \{compose_{tfenc \circ fff_{thermal}}, discard_{plain}, log_{\varphi_0^+}, log_{\varphi_1^-}, log_{\varphi_3^-}\}$. However, after resolving this conflict based on established precedence relations and historical actions, the set of active actions, α_{active} , comprises the logging of constraints along with the composed operation that aggregates multiple LiDAR data payloads and encrypts the outcome, $\mathcal{D}_{\mathcal{A}}(d) = (d', \{log_{\varphi_1^-}, log_{\varphi_3^-}, compose_{tfenc \circ fff_{thermal}}\})$.

Example 1: Decision Procedure Output for the data policy $\pi_{UAV_{LiDAR},1}$

$$\begin{aligned}
 id = UAV_{LiDAR} \in ID, v = 1 \in V_{UAV_{LiDAR}} & \quad (6) \\
 \mathcal{H}_d = [\langle id, v, \{transform_{detect}, log_{\varphi_0^-} \} \rangle] \wedge \varphi_0(d) = \top \wedge \varphi_1(d) = \perp \wedge \varphi_3(d) = \perp & \implies \\
 \implies \alpha_{constraints} = \alpha^+ \cup \alpha^- = \bigcup \{ \pi.\gamma_{UAV_{LiDAR},1}^+(\varphi) \} \cup \bigcup \{ \pi.\gamma_{UAV_{LiDAR},1}^-(\varphi) \} = & \\
 = \gamma^+(\varphi_0) \cup (\gamma^-(\varphi_1) \cup \gamma^-(\varphi_3)) = \{ log_{\varphi_0^+} \} \cup & \\
 \cup (\{ compose_{tfenc \circ fff_{thermal}}, log_{\varphi_1^-} \} \cup \{ discard_{plain}, log_{\varphi_3^-} \}) = & \\
 = \{ compose_{tfenc \circ fff_{thermal}}, discard_{plain}, log_{\varphi_0^+}, log_{\varphi_1^-}, log_{\varphi_3^-} \} \implies & \\
 \implies Deconfliction(\alpha_{constraints}, \alpha_{\mathcal{H}_d}, \pi_{id,v} \prec_{\alpha}) \implies & \\
 \implies (compose_{tfenc \circ fff_{thermal}} \prec discard_{plain}) \wedge (compose_{tfenc \circ fff_{thermal}} \prec log_{\varphi_0^+}) \wedge \alpha_{\mathcal{H}_d} \implies & \\
 \implies \mathcal{D}_{\mathcal{A}}(d) = (d', \{ log_{\varphi_1^-}, log_{\varphi_3^-}, compose_{tfenc \circ fff_{thermal}} \}) &
 \end{aligned}$$

Taking into account the previously mentioned definitions, the entity, $e \in E$, serves as the primitive element of our model, responsible for the production and processing of data in accordance with context-aware policy constraints, $\pi_{id,v}.\phi$. Therefore, the assignment of policies to entities is determined by the function, $E_{\Pi} : E \rightarrow \mathcal{P}(\Pi)$, which is redundantly handled by the ledger, DL . Specifically, $E_{\Pi}(e) = \{\pi_n, \dots, \pi_m\}$ denotes that a subset of

data policies has been assigned to the entity, establishing the constraints for produced data. For instance, the function, $E_{\Pi}(e_{UAV}) = \{\pi_{UAV_{LiDAR},1}, \pi_{CCTV_{glare},2}\}$, indicates that a UAV equipped with a LiDAR scanner should produce data that adheres to the constraints of the data policy, $\pi_{UAV_{LiDAR},1}$ (Formula (4)). Moreover, if this vehicle is extended with a high-resolution camera, the data produced from this sensor must comply with the constraints of the second version of the policy, $\pi_{CCTV_{glare},2}$.

Furthermore, it is essential to emphasize that a malicious entity can easily start to produce and disseminate data that bypass policy constraints. Conversely, the producer's computing capabilities may be limited, which can affect their ability to fully adhere to data policy constraints, or certain dimensions cannot have their values assigned by the entity during the data production process, as they require further evaluation or are influenced by factors beyond the control of the data producer. Notably, this strictly aligns with the concepts of Ubiquitous and Continuum Computing.

Consequently, the aforementioned requires us to address another element of our model. A set of capabilities (functions), $Caps = \{canWrite, canRead, \dots\}$, along with the mapping function, $E_{Caps} : E \rightarrow \mathcal{P}(Caps)$, that specifies the actual subset of capabilities assigned to the entity. Therefore, we define entities capable of writing data as producers, $\forall e_p \in E_{prod} \subseteq E : canWrite \in E_{Caps}(e_p)$, e.g., IoT sensors. Conversely, entities that read data are referred to as consumers, $\forall e_c \in E_{cons} \subseteq E : canRead \in E_{Caps}(e_c)$, e.g., IoT actuators. Notably, both core capabilities may occur simultaneously, $E_{prod} \cap E_{cons} \neq \emptyset$. Beyond the capabilities of producing and consuming, our model extends the possible set of capabilities to *canProcess*, *canReassign*, *canEnforce*, and *canVerify*. These capabilities were proposed to reduce the impact of malicious entity activities while considering relevant computing concepts. The subset of entities that possess these specialized capabilities—namely, data processing, data label re-assignment, data policy enforcement, and data origin and authenticity verification—are collectively referred to as the microservices set, $MS \subseteq E$.

Before exploring the indicated capabilities and microservices in greater detail, it is essential to define the primitive elements of our model that isolate data at rest, which we denote as topics, $T := Topics$. Furthermore, these primitives enable the communication mechanism utilized by entities from the moment of data production, through verification of its origin, enforcement of data policies, and any subsequent processing, until the delivery moment when it is consumed by entities. Moreover, the function, $E_T : E \rightarrow \mathcal{P}(T)$, specifies which topics can be utilized by which entities. Additionally, we distinguish four general classes of topics, where elements of these classes are associated with dimensions and corresponding label tuples by function $T_{\delta_{active},L} : T \rightarrow (\delta_{active} \times L)$:

- An unauthenticated class, $T_{unauth} \subseteq T$, where producers write their unverified data;
- An authenticated class, $T_{auth} \subseteq T$, where $T_{auth} \cap T_{unauth} = \emptyset$, that verifier entities utilize to store data that have been approved for authenticity and provenance;
- An intermediate topic class, $T_{inter} \subseteq T$, where $T_{inter} \cap T_{unauth} = \emptyset$, that is used by the processing entities to hold the intermediate outcomes (results) of applying transformation, fusion, and assessment functions to the data;
- A delivery class, $T_{delivery} \subseteq T$, where $T_{delivery} \cap T_{unauth} = \emptyset$, that is employed by policy enforcement microservices to store data for which the decision procedure resulted in a write action. This topic also facilitates later access to the stored data by consumers.

In reference to the previously identified types of entities, we determine those with the capability *canProcess* as processing microservices, $MS_{process} \subseteq MS$. Specifically, these microservices facilitate processing functions, $PF = TF \cup FF \cup AF$, which include transformation, $TF = \{tf_1, tf_2, \dots, tf_k\}$, fusion, $FF = \{ff_1, ff_2, \dots, ff_l\}$, and quality assessment functions, $AF = \{af_1, af_2, \dots, af_n\}$. Two first subsets represent the group of modification functions, that when applied to the data, d , result in modified (altered) data, d' . Thus, we define

the application of transformation functions on the data as $d_{transformed} = \odot_{tf \in TF(\alpha'_{active})} tf(d)$, and for fusion functions as $d_{fused} = \odot_{ff \in FF(\alpha''_{active})} ff(\{d_1, d_2, \dots, d_k\})$. In contrast, the assessment functions when applied to the data, d , labeled as λ_d , may yield new assessed values for the relevant data dimensions, $\lambda'_d = \odot_{af \in AF(\alpha'''_{active})} af(d)$. Notably, $TF(\alpha'_{active})$, $FF(\alpha''_{active})$, $AF(\alpha'''_{active})$, indicate the type of transformation, fusion, or assessment functions that will be utilized based on the stated action subset, α_{active} . Additionally, we propose that each microservice is limited to employing a single processing function. This aligns with the design of microservices, which should execute only a single atomic task. Consequently, $E_{PF} : E \rightarrow \mathcal{P}(PF)$, is a mapping between microservices and processing functions.

It is also important to highlight that processing operations can possess either a stateful or stateless characteristic. For instance, the uniqueness function is stateful, necessitating the retention of metadata about prior processed data or the ability to handle substantial statistical datasets that may exceed the storage capacity of the microservice (its memory). In contrast, the consistency function is stateless. Therefore, we proposed expanding the classes of the topic primitive to store prior (statistical) data used for stateful functions, which we denote as, T_{stats} .

Delving deeper, the capability *canReassign* pertains to the re-assignment of data-relevant attributes, $d' \triangleq d[d.headers.policy.labels \mapsto \lambda'_d]$. This process may only be initiated by a specific group of processing microservices, determined by the functions E_{Caps} and E_{PF} . Furthermore, microservices that identify the need for an attribute change utilize the dedicated chaincode, $assignLabels_{CS} \in DL.CS$. This ensures that any modifications made to the data flow labels are properly audited (transacted) within the ledger, $DL.TX$.

Entities with capability, *canEnforce*, are denoted as enforcement microservices, $MS_{enforce} \subseteq MS$. These microservices handle the policy decision procedure, $\mathcal{D}_A$, which states the active action subset, α_{active} , which must be enforced on the data. Furthermore, the intermediate topic class, T_{inter} , is used for actions related to data processing. Subsequently, any application of processing actions necessitates a corresponding data consistency check against the policy. This implies that the processing microservices, $MS_{process}$, after performing a sequence of processing functions, $[pf_k, \dots, pf_l] \subset PF$, will re-write the data to the corresponding authenticated topic class, T_{auth} . Consequently, the policy enforcement microservice evaluates whether the modified data payload adheres to the established policy constraints, thereby specifying the necessary actions once again. Ultimately, the decision procedure may state that the data should be written to the relevant topic instance of the delivery topic class, $T_{delivery}$, which is utilized by consumer entities, E_{cons} . An example of data dissemination involving the aforementioned microservices is illustrated in Figure 2.

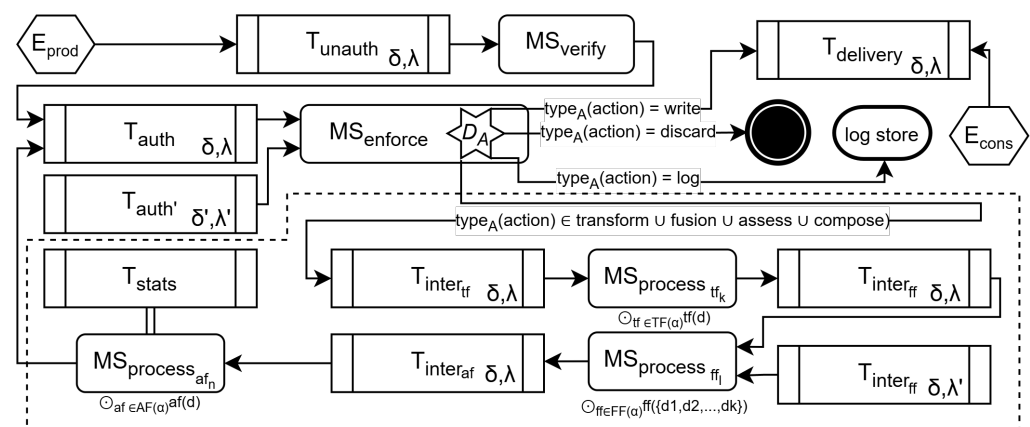


Figure 2. Topic classes and microservice utilization.

Lastly, we proposed entities with the capability, *canVerify*, referred to as verifier microservices, $MS_{verify} \subseteq MS$. These microservices utilize a mechanism that verifies the authenticity and provenance of the data, where we have applied a data-centric approach that leverages properties analogous to those employed in the Zero-Knowledge proofs of identity [26]. Essentially, the proof provided by the prover (the producer entity) does not reveal the prover's secret identity to the verifier, yet it remains verifiable. Therefore, each entity, with the exception of microservices, $e \in E \setminus MS$, is associated with a distinctive set of characteristic samples, denoted as fingerprints, *FID*. These fingerprints are derived by applying specific functions known as fingerprint techniques, $FT = \{ft_0, ft_1, \dots, ft_n\}$, to determine the unique hardware and software features, generated network characteristics, and radio signals related to the entity. Consequently, this indicates that at any given time, τ , there are no two entities defined by the same identity. Additionally, this invariant is reinforced by the distributed ledger structure, which reliably maintains a comprehensive dataset of all fingerprint samples calculated for entities.

In our model, the producers, E_{prod} , before disseminating data to the appropriate unauthenticated topics, $t_{unauth} \in T_{unauth}$, are obliged to seal their data using the features that characterize these entities. Once the sealed data is written to the unauthenticated topic, it becomes available for reading by the verifier microservices, MS_{verify} , which are permitted to either approve or discard the data. To accomplish this, verifiers execute the procedure $verifyData : D_{sealed} \rightarrow (\{true, false\} \times Bytes)$, which takes a sealed data as an input parameter, $d \in D_{sealed} \subset D$, and returns status and reason values. A comprehensive definition of the verification procedure, along with other related functions, is beyond the scope of this article; however, the implementation details of this procedure were discussed in our previous article [9].

Finally, in light of the previously mentioned definitions, we can establish invariants that define the scope of $read : (T \times D) \rightarrow \{true, false\}$ and $write : (T \times D) \rightarrow \{true, false\}$ operations for entities concerning specific topics. It is crucial to highlight that certain label dimensions require strict processing isolation, particularly the secrecy dimension, wherein data classified as confidential must not be processed by the same microservice that handles public data. To address this issue within our formal model, we introduce the global guard mechanism, $\mathcal{G} : (E \times D \times T) \rightarrow \{true, false\}$ (Formula (7)), which encapsulates the capability assignments precondition, E_{Caps} , along with two orthogonal mechanisms that must be satisfied simultaneously: the existence of a mapping between entities and topics, E_T , and label compatibility, $Match_L : ((\Delta_{dims} \times L) \times D) \rightarrow \{true, false\}$, which evaluates whether the label tuple of the data λ_d is compatible with the label scope of the target topic, λ_t . Consequently, the proposed guard $\mathcal{G}$ regulates the allowable dimensions and labels that microservices can process, ensuring that the introduction of future topics or entity types does not unintentionally bypass any of its checks.

Read/Write Guard

$$\mathcal{G}(e, d, t) \triangleq [canRead, canWrite \in E_{Caps}(e)] \wedge [t \in E_T(e)] \wedge [Match_L(T_{\delta_{active}, L}(t), d) = \top] \quad (7)$$

Formula (8) outlines the access control function, $canWrite : (E \times D \times T) \rightarrow \{true, false\}$, which determines whether an entity, e , is permitted to write data, d , on a particular topic, t . This assessment is contingent upon the collective satisfaction of the global guard, $\mathcal{G}(e, d, t)$, which universally enforces capability assignment, E_{Caps} , topic membership, E_T , and label compatibility, $Match_L$. The function returns a *true* value under the following conditions:

- The entity is a producer type, $e \in E_{prod}$, while the topic is classified under unauthenticated topics, $t \in T_{unauth}$;
- The entity is a type of verifier microservice, $e \in MS_{verify}$, and writes to authenticated topic class, $t \in T_{auth}$, after the data has been successfully verified, $verifyData(d) = (true, "Proofs Match.")$.
- The entity is the processing microservice, $e \in MS_{process}$, that writes data to the topics excluding the unauthenticated and delivery topic class, $t \in T \setminus (T_{unauth} \cup T_{delivery})$;
- The entity is a type of enforce microservice, $e \in MS_{enforce}$, and disseminates data to the delivery or intermediate topic, $t \in (T_{delivery} \cup T_{inter})$.

In all other conditions, the function yields a *false* value.

Access control: canWrite function

$canWrite : (E \times D \times T) \rightarrow \{true, false\}$

$\forall e \in E, \forall d \in D, \forall t \in T :$

$$canWrite(e, d, t) \triangleq \begin{cases} true & \text{if } \mathcal{G}(e, d, t) \wedge e \in E_{prod} \wedge t \in T_{unauth} \\ true & \text{if } \mathcal{G}(e, d, t) \wedge e \in MS_{verify} \wedge t \in T_{auth} \wedge \\ & \quad \wedge verifyData(d) = (true, "Proofs Match.") \\ true & \text{if } \mathcal{G}(e, d, t) \wedge e \in MS_{process} \wedge t \in T \setminus (T_{unauth} \cup T_{delivery}) \\ true & \text{if } \mathcal{G}(e, d, t) \wedge e \in MS_{enforce} \wedge t \in (T_{delivery} \cup T_{inter}) \\ false & \text{otherwise} \end{cases} \quad (8)$$

The function defined in Formula (9), *canRead*, is responsible for determining whether an entity, e , has the permission to read data, d , from a specific topic, t . This determination relies on the overall fulfillment of the global guard, $\mathcal{G}(e, d, t)$. The function returns a *true* value under the following conditions:

- The entity is a consumer type, $e \in E_{cons}$, while the topic is classified as a delivery topic, $t \in T_{delivery}$, and data is sealed, $d \in D_{sealed}$;
- The entity is a type of verifier microservice, $e \in MS_{verify}$, and only reads from the unauthenticated class, $t \in T_{unauth}$, where data may potentially be sealed, although this is not an obligatory, $d \in D$;
- The entity is the processing microservice, $e \in MS_{process}$, that reads data from intermediate and statistical topics, $t \in (T_{inter} \cup T_{stats})$;
- The entity is a type of policy-enforcing microservice, $e \in MS_{enforce}$, and reads data from the authenticated topics $t \in T_{auth}$.

In all other conditions, the function yields a *false* value.

Access control: canRead function

$canRead : (E \times D \times T) \rightarrow \{true, false\}$

$\forall e \in E, \forall d \in D, \forall t \in T :$

$$canRead(e, d, t) \triangleq \begin{cases} true & \text{if } \mathcal{G}(e, d, t) \wedge e \in E_{cons} \wedge d \in D_{sealed} \wedge t \in T_{delivery} \\ true & \text{if } \mathcal{G}(e, d, t) \wedge e \in MS_{verify} \wedge t \in T_{unauth} \\ true & \text{if } \mathcal{G}(e, d, t) \wedge e \in MS_{process} \wedge t \in (T_{inter} \cup T_{stats}) \\ true & \text{if } \mathcal{G}(e, d, t) \wedge e \in MS_{enforce} \wedge t \in T_{auth} \\ false & \text{otherwise} \end{cases} \quad (9)$$

5. Policy-Driven Data Dissemination

Concerning the elements of our formal model, *FedM*, in the following we present an example of a data dissemination pipeline, *Pipeline_{CCTV}* (Formula (12)) driven by the data policy outlined in Formula (10). In other words, this pipeline stems from that data policy. In addition, our example illustrates the application of the trustworthiness dimension in conjunction with secrecy and category labels.

The work of Byabazaire et al. [6] introduces a trust model defined over the closed interval, $L_{trust} = [0, 1]$, which is integrated within a three-phase Big Data Cycle: Initial Trust, Result-driven Trust, and Investigation Trust. Each phase generates intermediate trust values, and their aggregation determines the overall trustworthiness $tm_{general_{\tau}} = \sum_{n \in Phases} w_n * tm_n$, where $Phases = \{Init, Driven, Exam\}$.

In the *FedM* model, the Initial Trust phase derives trust from entity metadata, which encompasses factors such as sensor precision, anticipated lifecycle degradation, and contextual subjective assessment. This trust value may be assigned upon entity registration and recorded in the distributed ledger alongside the entity's fingerprints.

Similarly, the Result-driven Trust phase establishes feedback mechanisms based on consumer evaluations, which can be either human-driven or supported by automated methods. These evaluations reflect the producer's influence on the resulting IOs and Situational Awareness.

The Investigation Trust phase, which will be discussed in further detail later in this section, involves analyzing raw or processed data against policy-defined indicators that may evolve over time. Unlike the earlier phases, this trust value contributes to a broader time-dependent aggregation. The overall trust at a given time is computed as a weighted sum of the intermediate trust values. To illustrate this phase, consider a data policy and an evaluation function for estimating trust in an entity, exemplified by a security camera operating under sunlight exposure (glare), which can adversely affect the quality of captured CCTV frames.

This entity, $e_{CCTV} \in E_{prod}$, utilizes data policy, $\pi_{CCTV_{glare},1} \in \Pi$. Here, the data policy was defined for label tuple, where category and trustworthiness dimensions can be wild-cards, $\lambda_{id,v}(public, *, *)$. Furthermore, the entity's operational environment is particularly challenging due to sunlight glare, which can lead to degradation of the produced data (its CCTV frames). Consequently, the policy outlines the acceptable trustworthiness threshold for the producer. Specifically, the constraint ϕ_4 , denotes that the general trust value cannot be lower than 0.85. In contrast, ϕ_5 , evaluates whether the objects identified in the CCTV frames fall within the vehicle category, $CCTV_{car}$. Moreover, the applicable subset of actions related to this data includes: writing the data for detected vehicles ($write_{car}$) or any other object type ($write_{other}$); employing a transformation function that detects objects within a CCTV frame and re-assigns the category label ($transform_{detect}$); and conducting a data quality assessment that calculates trust value and updates the associated dimension label ($assess_{trust}$). Additionally, operations are in place to either discard encrypted data ($discard_{enc}$) or apply a general deny invariant ($discard_{payload}$), along with actions that log the results of constraints (log_{ϕ}). Moreover, the relationship between actions and constraints clarifies that the subset of actions corresponding to the satisfied constraint $\neg\phi_3$ is restricted to the successful logging of constraints, indicating that data is unencrypted, $\gamma^+(\neg\phi_3) = \{log_{\neg\phi_3^+}\}$; otherwise, encrypted data would be discarded. Notably, for constraint ϕ_4 , execution of the assessment action is required regardless of whether the outcome is a success or failure. This serves as a countermeasure against malicious entities that might attempt to pre-determine the trust value themselves and bypass constraint enforcement. Thus, if the decision procedure has previously indicated the need for an assessment action, the subsequent re-evaluation—if trustworthiness falls below the established threshold—

would lead to the discarding of the data flow. Finally, concerning constraint φ_5 , if the data category is not yet classified or does not match the vehicle object, the primary action invoked would be the $transform_{detect}$ action, as it takes precedence over the $write_{other}$ operation.

Policy $\pi_{CCTV_{glare},1}$: Glare effect on the CCTV

$$\begin{aligned}
 id = CCTV_{glare} \in ID, v = 1 \in V_{CCTV_{glare}}, \delta_{active} &= \{secrecy, category_{obj}, trust\} \\
 \lambda_{id,v} &= (public, *, *), \phi_{id,v} = \{\neg\varphi_3, \varphi_4, \varphi_5\} \subset \Phi, \theta_{id,v} = \{\langle trust_{min}, 0.85, [\tau_{noon}, \tau_{dusk}] \rangle\} \\
 \alpha_{id,v} &= \{write_{car}, write_{other}, discard_{payload}, discard_{enc}, assess_{trust}, transform_{detect}, \\
 &\quad \log_{-\varphi_3^+}, \log_{-\varphi_3^-}, \log_{\varphi_4^-}, \log_{\varphi_5^-}\} \subset \mathcal{A} \\
 \rho_{\lambda_{id,v}, \phi_{id,v}}(public) &= \{\neg\varphi_3\} \wedge \rho_{\lambda_{id,v}, \phi_{id,v}}(Label_{\mathcal{E}}(\lambda_{id,v}, category_{obj})) = \{\varphi_5\} \wedge \\
 &\quad \wedge \rho_{\lambda_{id,v}, \phi_{id,v}}(Label_{\mathcal{E}}(\lambda_{id,v}, trust)) = \{\varphi_4\} \\
 \gamma_{\alpha_{id,v}, \phi_{id,v}}(\neg\varphi_3) &= (\gamma^+(\neg\varphi_3), \gamma^-(\neg\varphi_3)) = (\{\log_{-\varphi_3^+}\}, \{\log_{-\varphi_3^-}\}) \wedge \\
 \gamma_{\alpha_{id,v}, \phi_{id,v}}(\varphi_4) &= (\gamma^+(\varphi_4), \gamma^-(\varphi_4)) = (\{assess_{trust}\}, \{discard_{payload}, assess_{trust}, \log_{\varphi_4^-}\}) \wedge \\
 \gamma_{\alpha_{id,v}, \phi_{id,v}}(\varphi_5) &= (\gamma^+(\varphi_5), \gamma^-(\varphi_5)) = (\{write_{car}\}, \{write_{other}, transform_{detect}, \log_{\varphi_5^-}\}) \\
 \prec_{\alpha_{id,v}} &= \{(discard_{enc}, discard_{payload}), (discard_{enc}, write_{car}), (discard_{enc}, write_{other}), \\
 &\quad (discard_{enc}, assess_{trust}), (discard_{enc}, transform_{detect}), (assess_{trust}, write_{car}), \\
 &\quad (assess_{trust}, write_{other}), (assess_{trust}, discard_{payload}), (discard_{payload}, write_{other}), \\
 &\quad (transform_{detect}, write_{other}), (transform_{detect}, assess_{trust}), (discard_{payload}, write_{car}), \\
 &\quad (transform_{detect}, discard_{payload})\} \\
 \mu_{id,v} &= \langle Org_1, active, [1 \text{ June } 2026 \text{ 12:00 UTC}, 1 \text{ July } 2026 \text{ 12:00 UTC}], description \rangle
 \end{aligned}
 \tag{10}$$

In light of the described data policy, consider that the assessment microservice possesses *canReassign* capability and employs the glare error function, $af_{glare} \in AF$, on the data produced by the entity, e_{CCTV} . This function is primarily used to determine the trust metric value. Essentially, this stateful function takes as arguments the timestamp from the data frame, along with the geo-localization and the entity's view angle and direction found in the data frame's metadata (Formula (2)) to calculate the time-varying solar position. Then it correlates that solar position with the statistical dataset, $t_{WDR_{glare}} \in T_{stats}$, which contains loss values for the Wide Dynamic Range (WDR) parameter corresponding to that specific solar position. For clarity, the WDR enables compensation of dark shadows and bright spots without compromising the details in the footage.

Furthermore, assume that entity e_{CCTV} has a WDR parameter set to a nominal value of 120 dB. During the hours of 11:00 AM to 1:00 PM, when the sun is at its highest elevation and directly impacts the sensors of the entity, this can result in an effective WDR loss of 20–30 dB in the brightest zone. Such a loss can negatively affect the visibility threshold, which determines the minimum scene-illuminance value needed for accurate recognition, whether for faces or license plates. Conversely, at dusk, the WDR drops by approximately 15 dB. Therefore, the trust is equal for the noon scenario, $\lambda_{d_{noon}} = (public, CCTV_{car}, 0.835)$, and for the dusk scenario, $\lambda_{d_{dusk}} = (public, CCTV_{car}, 0.87)$. After determining these values, the data trustworthiness label is re-assigned by the assessment microservice, which invokes the dedicated chaincode. Subsequently, the enforcement microservice executes the decision procedure, $\mathcal{D}_{\mathcal{A}}$, to state whether the trustworthiness label meets the specified trust-related constraint, $\varphi_4 \in \phi$.

Finally, according to the policy provided, $\pi_{CCTV_{glare},1}$ (Formula (10)), and the resulting trustworthiness values, Formula (11) represents the result of the decision procedure, $\mathcal{D}_A$, for the noon scenario. This formula pertains to the unencrypted data generated during noon hours, where the constraints $\neg\varphi_3(d_{noon})$ and $\varphi_5(d_{noon})$ are satisfied, while the constraint $\varphi_4(d_{noon})$ is false. This implies the presence of conflicting operations. Consequently, following the deconfliction process—utilizing precedence relations, $\prec_\alpha$, and previously applied actions, $\mathcal{H}_d$, including a transformation function for object detection and trust assessment—the active action set consists of discarding the data flow, $\alpha_{active} = \{discard_{payload}\}$. In contrast, for the dusk scenario, where the constraint $\varphi_4(d_{dusk})$ is satisfied. The active action set includes the writing of data for the detected vehicles, $\alpha_{active} = \{write_{car}\}$.

Example 2: Decision Procedure Output for the data policy $\pi_{CCTV_{glare},1}$ at $\tau_n := noon$

$$\begin{aligned}
 id = CCTV_{glare} \in ID, v = 1 \in V_{CCTV_{glare}}, \lambda_{d_{noon}} &= (public, CCTV_{car}, 0.835) \\
 \mathcal{H}_d &= [\langle id, v, \{log_{\neg\varphi_3^+}, log_{\varphi_4^-}, log_{\varphi_5^-}, transform_{detect}\} \rangle, \langle id, v, \{log_{\varphi_5^+}, assess_{trust}\} \rangle] \wedge \\
 \wedge \neg\varphi_3(d_{noon}) &= \top \wedge \varphi_4(d_{noon}) = \perp \wedge \varphi_5(d_{noon}) = \top \implies \alpha_{constraints} = \alpha^+ \cup \alpha^- = \\
 &= \bigcup \{\pi \cdot \gamma_{CCTV_{glare},1}^+(\varphi)\} \cup \bigcup \{\pi \cdot \gamma_{CCTV_{glare},1}^-(\varphi)\} = (\gamma^+(\neg\varphi_3) \cup \gamma^+(\varphi_5)) \cup \gamma^-(\varphi_4) = \\
 &= (\{log_{\neg\varphi_3^+}\} \cup \{write_{car}\}) \cup \{discard_{payload}, assess_{trust}, log_{\varphi_4^-}\} = \\
 &= \{log_{\neg\varphi_3^+}, discard_{payload}, assess_{trust}, log_{\varphi_4^-}, write_{car}\} \implies \\
 \implies Deconfliction(\alpha_{constraints}, \alpha_{\mathcal{H}_d}, \pi_{id,v}, \prec_\alpha) &\implies \mathcal{D}_A(d_{noon}) = (d'_{noon}, \{discard_{payload}\})
 \end{aligned} \tag{11}$$

Formula (12) models the complete processing pipeline driven (govern) by the proposed data policy, $\pi_{CCTV_{glare},1}$. To enhance clarity, we examine below the dissemination pipeline from the perspective of the IoT Context-Aware architecture (Figure 1), which can be depicted as a system of directed and acyclic processing pipelines. These pipelines begin with data being produced by entities in the Perception Layer, which then progress through the Data Management Layer. Within this layer, data-relevant attributes can be verified, necessary data alterations can be performed, and the data itself can be assessed for consistency with the established data policy. The process ends when the data is consumed by entities within the Intelligent Service Layer. Therefore, we proposed feasible functional blocks that can be utilized to model discrete stages of data dissemination within a federated special IoT environment. We identify five core stages in this process: *Ingestion*, *Verification*, *Enforcement*, *Routing*, and *Delivery*, along with two intermediate stages that pertain to altering operations on data: *Transformation* and *Fusion*, one intermediate stage used to assess data: *Assessment*, and an intermediate routing stage: *Routing_{inter}*.

At the beginning of the core pipeline workflow, a producer entity, $e_p \in E_{prod}$, produces sealed data, $d_{sealed} \in D_{sealed}$, based on the provided payload, $payload \in Bytes$. Following this, the function, $canWrite(e_p, d_{sealed}, t_{unauth})$ (Formula (8)), is used to verify whether the producer is allowed to write data to an unauthenticated topic. The aforementioned establishes the *Ingestion* stage, where a successful outcome from the $canWrite$ function indicates that the data can be written to the topic, $\frac{d_{sealed}}{t_{unauth}}$. Consequently, the write operation initiates the subsequent stage, *Verification*.

During the *Verification* stage, the ingested sealed data is retrieved from the unauthenticated topic by a dedicated microservice capable of verifying this data. More specifically, the verification function, $verifyData$, is utilized by a verifier microservice, $ms_v \in MS_{verify}$, to validate the authenticity and origin of the sealed data. Based on the outcome of the

verification, the microservice either writes the verified data to an authenticated topic with the corresponding dimensions and labels, $\frac{d_{verified}}{t_{auth}}$, or discards it.

Dissemination pipeline driven by the data policy $\pi_{CTTV_{glare,1}}$

$$\begin{aligned}
 Pipeline_{CTTV} &\triangleq Delivery \circ Routing \circ Enforcement \circ Routing_{inter} \circ Assessment \circ \\
 &\circ Routing_{inter} \circ Transformation \circ Routing \circ Enforcement \circ Verification \circ Ingestion \\
 Pipeline_{CTTV}(e_p, e_c, ms_v, ms_e, ms_m, ms_a, t_{unauth}, payload) &\triangleq \\
 &\triangleq Ingestion(e_p, t_{unauth}, payload) \xrightarrow[t_{unauth}]{d_{sealed}} Verification(ms_v, t_{unauth}, d_{sealed}) \xrightarrow[t_{auth}]{d_{verified}} \\
 &\xrightarrow[t_{auth}]{d_{verified}} Enforcement(ms_e, t_{auth}, d_{verified}) \xrightarrow[\alpha_{active'} \subset \pi.\alpha]{d_{enforced'}} Routing(ms_e, \alpha_{active'}, d_{enforced'}) \xrightarrow[t_{inter'}]{d_{routed'}} \\
 &\xrightarrow[t_{inter'}]{d_{routed'}} Transformation(ms_m, t_{inter'}, d_{routed'}) \xrightarrow{d_{transformed}} Routing_{inter}(ms_m, d_{transformed}) \xrightarrow[t_{inter''}]{d_{routed''}} \\
 &\xrightarrow[t_{inter''}]{d_{routed''}} Assessment(ms_a, t_{inter''}, d_{routed''}) \xrightarrow[t_{auth}]{d_{assessed}} Routing_{inter}(ms_a, d_{assessed}) \xrightarrow[t_{auth}]{d_{assessed}} \\
 &\xrightarrow[t_{auth}]{d_{assessed}} Enforcement(ms_e, t_{auth}, d_{assessed}) \xrightarrow[\alpha_{active''} \subset \pi.\alpha]{d_{enforced''}} Routing(ms_e, \alpha_{active''}, d_{enforced''}) \xrightarrow[t_{delivery}]{d_{routed'''}} \\
 &\xrightarrow[t_{delivery}]{d_{routed'''}} Delivery(e_c, t_{delivery}, d_{routed''''})
 \end{aligned} \tag{12}$$

where:

$$\begin{aligned}
 t_{unauth} &\in T_{unauth}, t_{auth} \in T_{auth}, t_{delivery} \in T_{delivery}, t_{inter'}, t_{inter''} \in T_{inter} \\
 e_p &\in E_{prod}, e_c \in E_{cons}, ms_v \in MS_{verify}, ms_e \in MS_{enforce}, ms_m, ms_a \in MS_{process}
 \end{aligned}$$

Once the data have been verified with success, the *Enforcement* stage begins. During this stage, the policy enforcement microservice, $ms_e \in MS_{enforce}$, serves as an oracle and handles the policy decision procedure, $\mathcal{D}_A$. The microservice evaluates the payload of the verified data along with its labels, $d_{verified}.headers.policy.labels$, to ensure compliance with the data-related policy, $\pi_{id,v} \in \Pi$, and its constraints, $\pi.\phi$. As a result of the *Enforcement* stage, a decision regarding the resulting policy action set is made, $\alpha_{active'} \subset \pi.\alpha$. This enables the *Routing* stage to appropriately disseminate verified and policy-enforced data to output topics, $\frac{d_{enforced}}{\alpha_{active'} \subset \pi.\alpha}$.

Consequently, the *Routing* stage concurrently disseminates the enforced data to delivery topics, $T_{delivery}$, and intermediate topics, T_{inter} , based on the stated active action set, $\alpha_{active'}$. When considering dissemination to delivery topics, $\frac{d_{routed''''}}{t_{delivery}}$, this enables consumers, $e_c \in E_{cons}$, to read verified and enforced data, $d_{routed''''}$, which corresponds to the *Delivery* stage. However, prior to this operation, an appropriate access control function, $canRead(e_c, d_{routed''''}, t_{delivery})$ (Formula (9)), is applied to determine whether a consumer is allowed to read data from the delivery topic.

In contrast, the intermediate stages—namely *Fusion*, *Transformation*, and *Assessment*—are managed by dedicated processing microservices along with the intermediate topics utilized by these microservices. In this pipeline, $\frac{d_{routed'}}{t_{inter'}}$, corresponds to the intermediate topic of the first function in the processing sequence, $[tf_{detect}, af_{glare}]$. For clarity, we have reduced the number of functional blocks by combining the actions $transform_{detect}$ and $assess_{trust}$ defined within the data policy into a composed action, $compose_{af_{glare} \circ tf_{detect}}$. Consequently, the intermediate topic, $t_{inter'}$, is utilized by a microservice, $ms_m \in MS_{process}$, that applies object detection, tf_{detect} . Notably, this microservice may possess a *canReassign* ca-

pability, allowing it to imply changes to data-relevant attributes. If necessary, the chaincode $assignLabels_{CS}$ is employed. The aforementioned denotes the intermediate stage named *Transformation*.

After the transformation and re-assignment of attributes, the internal routing procedure, $route_{inter}(ms_m, d_{transformed})$, of the intermediate *Routing_{inter}* stage is invoked. Initially, this procedure updates (increments) the data header value related to the next function index to be applied, $d_{transformed}.headers.policy.action.index$. Subsequently, the intermediate topic extraction function, $ProcessingTopic_{\mathcal{E}}$, is utilized with the updated index. Consequently, the transformed data is written to the extracted intermediate topic with the corresponding dimensions and labels, $\frac{d_{routed''}}{t_{inter''}}$. In our pipeline, the updated index corresponds to the trustworthiness assessment function, af_{glare} , which is applied within the *Assessment* stage.

The intermediate dissemination process continues until all designated processing functions have been applied. Ultimately, the final processing microservice—last in the sequence of functions, $ms_a \in MS_{process}$ —writes the data to the corresponding authenticated topic class, $\frac{d_{assessed}}{t_{auth}}$. At this point, the policy enforcement microservice evaluates whether the modified data payload adheres to the established policy constraints, subsequently determining the necessary actions once again, $\alpha_{active''}$. In our example, the second set of actions indicates a write operation to the delivery topic, $\frac{d_{routed'''}}{t_{delivery}}$.

Core Stage: Enforcement

$$\begin{aligned}
 & Enforcement(ms_e, t_a, d_{verified}) \xrightarrow[\alpha_{active} \subset \pi.\alpha]{d_{enforced} = d_{verified}} Routing(ms_e, \alpha_{active}, d_{enforced}) \\
 & \forall ms_e \in MS_{enforce}, \forall d_{verified} \in D_{sealed}, \forall t_a \in T_{auth} : \\
 & Enforcement(ms_e, t_a, d_{verified}) \triangleq (idle(ms_e) = \top \wedge canRead(ms_e, d_{verified}, t_a) = \top) \implies \\
 & \implies read(t_a, d_{verified}) = \top \implies \mathcal{D}_{\mathcal{A}}(d_{verified}) = (d'_{verified}, \alpha_{active}) \implies \text{let } d_{enforced} = d'_{verified} \text{ in} \\
 & \quad \quad \quad route(ms_e, \alpha_{active}, d_{enforced})
 \end{aligned} \tag{13}$$

The Formula (13) outlines the *Enforcement* stage. Whenever the enforcement microservice, $ms_e \in MS_{enforce}$, is idle and can read verified data from the authenticated topic, $canRead(ms_e, d_{verified}, t_a)$, then data is read, $read(t_a, d_{verified})$. This implies the execution of the decision procedure, $\mathcal{D}_{\mathcal{A}}(d_{verified})$ (Formula (5)), to state the resulting subset of policy actions, α_{active} . Therefore, the microservice evaluates the payload of the verified data, $d_{verified}.payload$, along with its associated labels, $d_{verified}.headers.policy.labels$, to ensure compliance with the data policy, $\pi_{id,v} \in \Pi$, and its constraints, $\pi_{id,v}.\phi$. Successful execution of the decision procedure invokes the route procedure, $route(ms_e, \alpha_{active}, d_{enforced})$, which requires as input the active action subset, α_{active} , along with the verified and sealed data that have already been enforced. Essentially, the route procedure denotes the *Routing* stage.

The Formula (14) outlines the core *Routing* stage, during which the route procedure, $route(ms_e, \alpha_{active}, d_{enforced})$, facilitates a flexible policy enforcement mechanism that allows for multiple concurrent actions, specifically enabling simultaneous dissemination to various output topics. The routing procedure is defined by several conditional statements that consider the action category, $Type_{\mathcal{A}}(action)$. Two primary statements address the *write* and *discard* action categories. If the condition for the former is met, the function $Topics_{\mathcal{E}} : \mathcal{A} \rightarrow \mathcal{P}(T)$, extracts the delivery topics from the action definition, after which data is written to the specified delivery topic or group of topics. Conversely, if the latter condition is satisfied, the data is discarded, $discard(action, t_a, d_{enforced})$. In addition to

these categories, there is also a *log* action category and a set of data processing actions, $\{transform, fusion, assess, compose\}$, that refers to intermediate processing stages.

Notably, we specified that each microservice is limited to employing a single processing function. In addition, the enforced action can represent a chained sequence of processing functions, such as $d_{transformed} = \odot_{tf \in TF(\alpha'_{active})} tf(d)$, which involves the application of multiple processing (transformation) microservices. To effectively manage this chained processing scenario, we proposed an intermediate topic extraction function, $ProcessingTopic_{\mathcal{E}} : \mathcal{P}(PF) \times \mathbb{N}_1 \rightarrow T$. This function accepts an ordered sequence of processing functions, $pf_{chained} = [pf_i, \dots, pf_j] \subset \mathcal{P}(PF)$, along with an index (position), $index \in \mathbb{N}_1$, which denotes the next function to be applied in this specific sequence.

In the *Routing* stage, the index is always set to one, $k = 1$. This indicates the extraction of the first intermediate topic, $t_{inter} \in T_{inter}$, that will be used to process the data. Consequently, the write access control function is invoked, $canWrite(ms_e, d_{routed'}, t_{inter})$. If the enforcement microservice is allowed to write to the derived topic, then the data headers associated with the processing chain are initialized. The processing sequence is assigned to the data, $d_{routed'}.headers.policy.action.pf_{chained} \leftarrow pf_{chained}$, and the actual index of the processing function is set to first function in this sequence, $d_{routed'}.headers.policy.action.index \leftarrow 1$. Subsequently, the data is written to the intermediate topic, $write(t_{inter}, d_{routed'})$.

Core Stage: Routing

$$\begin{aligned}
 & Routing(ms_e, \alpha_{active}, d_{enforced}) \\
 & \forall \alpha_{active} \subset \mathcal{A}, \forall d_{enforced} \in D_{sealed}, \forall t_a \in T_{auth}, \forall ms_e \in MS_{enforce} : \\
 & Routing(ms_e, \alpha_{active}, d_{enforced}) \triangleq route(ms_e, \alpha_{active}, d_{enforced}) \implies \forall action \in \alpha_{active} : \\
 & \left\{ \begin{array}{ll}
 Topics_{\mathcal{E}}(action) = topics \subset T_{delivery} \implies & \text{if } Type_{\mathcal{A}}(action) = write \\
 \implies \forall t_{delivery} \in topics : & \\
 canWrite(ms_e, d_{routed'}, t_{delivery}) = \top \implies & \\
 \implies write(t_{delivery}, d_{routed'}) = \top & \\
 \\
 log(action, d_{enforced}) & \text{if } Type_{\mathcal{A}}(action) = log \\
 \\
 discard(action, t_a, d_{enforced}) & \text{if } Type_{\mathcal{A}}(action) = discard \\
 \\
 \text{let } pf_{chained} = action.pf_{chained} \subseteq \mathcal{P}(PF) \text{ in} & \text{if } Type_{\mathcal{A}}(action) \in \{transform, \\
 ProcessingTopic_{\mathcal{E}}(pf_{chained}, 1) = t_{inter} \implies & fusion, assess, compose\} \\
 canWrite(ms_e, d_{routed'}, t_{inter}) = \top \implies & \\
 \implies d_{routed'}.headers.policy.action.index \leftarrow 1 \wedge & \\
 \wedge d_{routed'}.headers.policy.action.pf_{chained} \leftarrow pf_{chained} \implies & \\
 \implies write(t_{inter}, d_{routed'}) = \top &
 \end{array} \right. \quad (14)
 \end{aligned}$$

where:

$$d_{routed'} = d_{enforced}, Topics_{\mathcal{E}} : \mathcal{A} \rightarrow \mathcal{P}(T), ProcessingTopic_{\mathcal{E}} : \mathcal{P}(PF) \times \mathbb{N}_1 \rightarrow T$$

Formula (15) outlines a generic definition of a functional block applicable to all three intermediate processing stages: *Transformation*, *Fusion*, and *Assessment*. Whenever the processing microservice, $ms_{proc} \in MS_{process}$, is idle and able to read from the intermediate topic, $canRead(ms_{proc}, d_{routed'}, t_{inter})$, it proceeds to read the enforced data. This implies the execution of the processing procedure, $process(d_{routed'}, pf_{actual})$, which applies the actual

processing function, $pf_{actual} \in PF$, to the relevant data, d_{routed} , based on the data's action index, $index_{pf_{actual}} = d_{routed}.headers.policy.action.index$. As a result, the processed data, $d_{processed}$, along with the proposed data-relevant attributes, $\lambda_{d_{processed}}$, are returned. If the label tuple, $\lambda_{d_{routed}}$, prior to processing does not match the proposed tuple, the *assignLabels* procedure is employed to re-assign labels. Moreover, the use of chaincode *assignLabels_{CS}* ensures that any changes to data labels are adequately audited within the DLT. Subsequently, the internal routing procedure, $route_{inter}(ms_{proc}, d_{transformed})$, is initiated, even if the label tuples are the same.

Intermediate Stage: Generic Processing—Transformation, Fusion, Assessment

$$\begin{aligned}
 & \forall ms_{proc} \in MS_{process}, \forall d_{routed} \in D_{sealed}, \forall t_{inter} \in T_{inter} : \\
 & : \begin{cases} Transformation(ms_{proc}, t_{inter}, d_{routed}) \\ Fusion(ms_{proc}, t_{inter}, d_{routed}) \\ Assessment(ms_{proc}, t_{inter}, d_{routed}) \end{cases} \triangleq \begin{matrix} idle(ms_{proc}) = \top \\ \wedge \\ canRead(ms_{proc}, d_{routed}, t_{inter}) = \top \end{matrix} \implies \\
 & \implies read(t_{inter}, d_{routed}) = \top \implies process(d_{routed}, pf_{actual}) = (d_{processed}, \lambda_{d_{processed}}) \implies \\
 & \implies \begin{cases} assignLabels(d_{processed}, \lambda_{d_{processed}}) = d_{processed'} \implies \text{if } \lambda_{d_{routed}} \neq \lambda_{d_{processed}} \\ \implies route_{inter}(ms_{proc}, d_{processed'}) \\ route_{inter}(ms_{proc}, d_{processed}) \end{cases} \quad \text{otherwise} \quad (15)
 \end{aligned}$$

where:

$$index_{pf_{actual}} = d_{routed}.headers.policy.action.index$$

$$pf_{actual} = d_{routed}.headers.policy.action.pf_{chained}[index_{pf_{actual}}]$$

Formula (16) defines the intermediate routing stage, denoted as *Routing_{inter}*, during which the internal routing procedure, $route_{inter}(ms_{proc}, d_{processed})$, is invoked. This procedure controls and coordinates the execution of sequenced processing functions, pf_{chain} , resulting in the dissemination of processed data to either the authenticated topic class, T_{auth} , or to the intermediate class, T_{inter} . Initially, the procedure verifies the cardinality constraint by comparing the number of elements in the sequenced processing set against the actual index value assigned to the processed data, $|pf_{chain}| = index_{pf_{actual}}$. If both values match, it indicates that there are no pending processing functions, allowing the data to be written to the corresponding authenticated topic. Notably, the label dimensions and label tuples are compared, $Match_L(T_{\delta_{active}, L}(t_a), d_{processed})$, and the access control function, $canWrite(ms_{proc}, d_{processed}, t_a)$ is executed. If these conditions are satisfied, processed data is subsequently written, $write(t_a, d_{processed})$. Conversely, if the cardinality constraint is not met, the actual action index for the data is updated (incremented by one), $d_{processed}.headers.policy.action.index = index_{pf_{actual}} + 1$. The intermediate topic extraction function, $ProcessingTopic_{\mathcal{E}}(pf_{chain}, index_{pf_{next}})$, is then employed with the updated index. Consequently, the processed data is disseminated to the intermediate topic extracted, $write(t_{inter}, d_{processed})$.

Intermediate Stage: $Routing_{inter}$

$$\begin{aligned}
 & Routing_{inter}(ms_{proc}, d_{processed}) \\
 & \forall ms_{proc} \in MS_{process}, \forall d_{processed} \in D_{sealed} : \\
 & Routing_{inter}(ms_{proc}, d_{processed}) \triangleq route_{inter}(ms_{proc}, d_{processed}) \implies \\
 & \implies \left\{ \begin{array}{l} \exists t_a \in T_{auth} : Match_L(T_{\delta_{active}, L}(t_a), d_{processed}) = \top \implies \text{if } |pf_{chain}| = index_{pf_{actual}} \\ \implies canWrite(ms_{proc}, d_{processed}, t_a) = \top \implies \\ \implies write(t_a, d_{processed}) = \top \\ \\ d_{processed}.headers.policy.action.index \leftarrow index_{pf_{next}} \implies \text{otherwise} \\ \implies ProcessingTopic_{\mathcal{E}}(pf_{chain}, index_{pf_{next}}) = t_{inter} \implies \\ \implies canWrite(ms_{proc}, d_{processed}, t_{inter}) = \top \implies \\ \implies write(t_{inter}, d_{processed}) = \top \end{array} \right. \quad (16)
 \end{aligned}$$

where:

$$\begin{aligned}
 & ProcessingTopic_{\mathcal{E}} : \mathcal{P}(PF) \times \mathbb{N}_1 \rightarrow T, pf_{chain} = d_{processed}.headers.policy.action.pf_{chained} \\
 & index_{pf_{actual}} = d_{processed}.headers.policy.action.index, index_{pf_{next}} = index_{pf_{actual}} + 1
 \end{aligned}$$

6. Instantiation Example of the IFC Components

In this section, we have refined our novel multi-layered framework [9], where we proposed a deployment of IFC components that facilitate policy-supported contextual dissemination of data. Our improvement mainly utilizes technologies and components that already have been provided within our solution, which requires low re-engineering effort. Notably, a new integrated technology is the Open Policy Agent [11], which functions as a part of the PEP component, and will be the subject of the further experimental studies.

6.1. Overview

Before exploring the indicated extension, we briefly describe the overall architecture of our framework. Figure 3 depicts the system's overall structure for a federation comprising two organizations: *Org1* and *Org2*. At the Data Queue layer, we have deployed the Apache Kafka stream-processing platform [27], while the Data Schema layer utilizes Schema Registry [28]. For the Distributed Ledger layer, we selected the Hyperledger Fabric solution [29]. In the Streams Microservice layer, we have implemented two of the same streams processing logic, employing the Kafka Streams API library in Java [30] and the Sarama library in Go [31]. Finally, at the Policy Enforcement layer, we have deployed a history- and precedence-aware OPA engine that evaluates flows based on data policies written in the Rego language [32]. Essentially, our pluggable architecture consists of the following layers:

- *Data Scheme Layer*: Structurally and semantically determines the fields of data flows (streams) that any entity within the federation environment may encounter while processing streams. It serves as a necessary prerequisite for correct interpretation and policy-driven dissemination of data;
- *Policy Enforcement Layer*: Evaluates data streams and delineates their dissemination, determining the applicable actions for a given stream based on its payload, schema, context, and provenance;
- *Publisher Layer*: Encompasses entities that generate data streams in accordance with established data policies, while also facilitating data-centric protection of these streams through a sealing process;

- *Subscriber Layer*: Comprises authenticated entities that consume verified, policy-enforced data streams from the Data Queue layer;
- *Streams Microservice Layer*: Verifies the sealed data streams originating from the Publisher layer and enforces (routes) policy actions, such as write or discard, as well as executing stream processing tasks, including enrichment;
- *Data Queue Layer*: Consists of distributed queues that acquire, merge, store, and replicate data streams from the Publisher layer, thereby making them accessible to both the Subscriber and Streams Microservice layers;
- *Distributed Ledger Layer*: Enables the interchangeable deployment of various DLTs, which reliably maintain a comprehensive dataset of all registered data policies, data schemas, and entity identities within the federation's organizations. In addition, it may retain information on entity trust, data quality, and environmental context, all of which inform policy-driven data dissemination;
- *Fingerprint Enrichment Layer*: Supports connectionless protocols such as UDP via a protocol forwarder that converts data into connection-oriented formats such as TCP. Furthermore, it enables behavior-based authentication for entities by gathering various features, including network and radio characteristics;
- *Communication Layer*: Connects the various layers of the framework to the distributed ledger via a hardware-software IoT gateway;
- *Device Maintain Layer*: Manages entity registration and establishes entity identity through fingerprinting methods, in addition to uploading the necessary data policies pertinent to each entity.

In light of the aforementioned layers, we now provide a general description of how our framework integrates with the IFC components identified in Section 3. Notably, our deployment supports data message level granularity, utilizing the headers element of the Kafka structure record to associate policy labels with a data message (its payload). Furthermore, the Data Queue, in conjunction with the Streams Microservice layer, facilitates a mechanism for concurrent (multi-topic) data stream dissemination enforced by the outcomes stated by the Policy Enforcement layer. In this context, we have utilized the OPA engine, which operates at the data message-part level, alongside enforcement microservices, to facilitate history-aware decision-making and support policy-driven contextual evaluation of data payloads. Additionally, data policies and the dependencies defined in our formal model are stored reliably and redundantly within the Distributed Ledger layer.

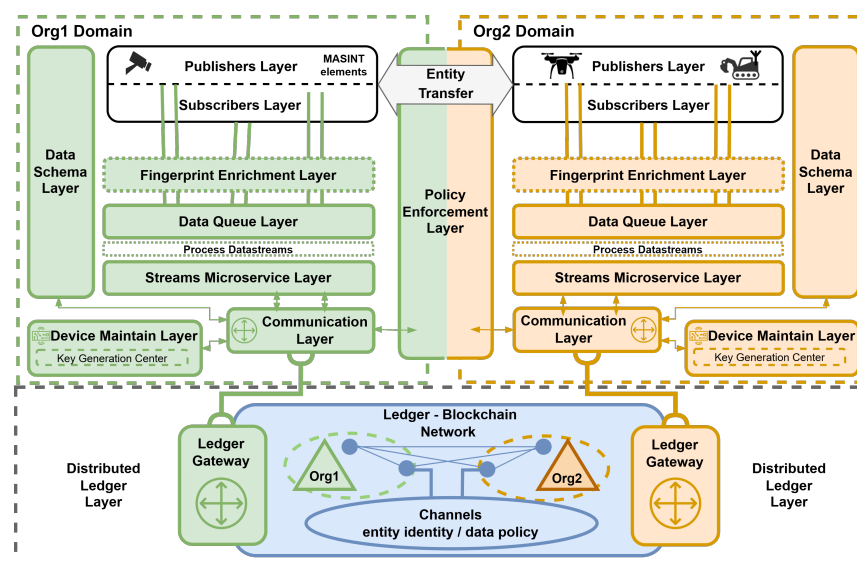


Figure 3. Proposed framework general overview.

Policy Specification Module and Labeling Management Component

The Distributed Ledger layer is incorporated for tasks related to Labeling Management and Policy Specification, allowing for the application and dynamic modification of various label dimensions expressiveness. Furthermore, this layer facilitates the definition and approval of data policy constraints and required actions within heterogeneous and evolving environments. In addition, its decentralized nature enhances resilience against various D4M attacks, while also minimizing the risk of the benevolent developer by separating the data plane and policy logic from the core application functionality. Finally, the use of a consensus protocol enforces that any policy change is approved among all involved parties, with chaincodes enabling these changes to be implemented consistently and procedurally.

Labeling Endpoint

The record (message) is the atomic data unit processed and disseminated by the Data Queue and Streams Microservice layer. Conceptually, a Kafka message is structured with a key, value, timestamp, and optional headers. Consequently, these headers can be utilized to store relevant attributes that facilitate security, reliability, and contextual dissemination mechanisms. Within our framework, the entities of the Publisher layer are responsible for producing data in accordance with context-aware policy constraints, where the appropriate label tuples can be stored in the message headers. Additionally, the role of the Labeling Endpoint component may also be assigned to microservices that possess the *canReassign* capability and operate within the Streams Microservice layer. Notably, any re-assignment of attributes necessitates the use of a dedicated chaincode, which ensures that modifications of record labels are thoroughly audited within the Distributed Ledger layer.

Tracking Engine

Logically, Kafka messages are organized into sequenced structures referred as topics (logs), where the set of messages with identical keys creates a logical subset of a topic named partition. Physically, messages are grouped and stored on disk as log segments, where a subset of segments makes up a logical partition, and a set of related partitions composes a topic. With regard to the Tracking Engine component, our Data Queue layer enables single data message level granularity, where data is isolated on topics, and all topic partitions are distributed over Kafka cluster brokers and can be replicated to enable message resilience.

Furthermore, within our framework, the Kafka topic enables the communication mechanism utilized by entities from the moment of data production, through verification of its origin, enforcement of data policies, and any subsequent processing, until the delivery moment when the data is consumed. To write to or read from these topics, entities must specify unique names of these topics. Therefore, a sensible naming convention must be employed, and we proposed the following convention, *topicClass.(dimension.value)⁺*, and five general topic class prefixes (*topicClass*): *unauth.**, *auth.**, *inter.**, *stats.**, *delivery.**, where the suffix *(dimension.value)⁺* denotes one or more dimension—value pairs determined by the mapping $T_{\delta_{active},L}$. This distinction for the prefix and suffix of the name is essential, in order to define and handle entity access and authorization using Access Control Lists (ACLs). Notably, the Kafka technology has built-in components and mechanisms, where ACLs (mapping E_T) can be defined and enforced to specify which entities can access a specific topic and the operations they can perform on that resource, such as the READ and WRITE operation. Accordingly, the *topicClass* prefix serves as the general prefix-based rule, which governs all topics within a class regardless of their suffix. For example, the topic name *delivery.secrecy.confidential.category.vehicles.trustworthiness.high* comprises three

label dimensions: secrecy, category, and trustworthiness, and below, the Kafka ACL rule can be defined for a whole *delivery* class:

```
kafka-acls --bootstrap-server localhost:9092 --allow-principal User:*
--add --operation Read --topic delivery. --resource-pattern-type prefixed
```

Notably, for bounded (or continuous) dimensions, we applied a fixed, human-readable value name that encodes a bounded range; for instance, *high* for bound $(0.75, 1.00]$, or *any* for bound $[0.00, 1.00]$. However, this discretization does not reduce the active dimension space—for dimensions that are continuous or bounded, a precise float value may still be carried in the message header. Simultaneously, a discrete suffix reduces the number of topics that must be managed and acts as a coarse sorting mechanism over the bounded dimension space, enabling consumers to subscribe selectively to topics by range without inspecting individual messages.

Lastly, to implement a mechanism for data dissemination (routing) to multiple concurrent topics based on the defined policy active action subset, α_{active} , we employed the Kafka Streams Processor API's *context.forward()* method in conjunction with a functional interface *TopicNameExtractor*. This method is executed for each specified action using the input record (evaluated data), where each invocation implies the extraction of the destination topic's name (*TopicNameExtractor*), subsequently allowing the data to be routed.

Policy Enforcement Point

The Open Policy Agent is an open-source, domain-agnostic engine that works at the data message-part level, alongside enforcement microservices. The OPA engine evaluates the input data against context-aware constraints to determine decisions (actions). An example of a policy definition, along with the corresponding input payload and engine result, is illustrated in Figure 4. Policy definitions are written using a high-level declarative language called Rego, which facilitates various fine-grained algebraic and logical expressions. Furthermore, it supports built-in functions suited for miscellaneous use cases. In addition to the native built-in categories—such as comparison, regex, and encoding—the OPA architecture allows for the registration of custom-built functions developed in Go, including a built-in that computes Shannon entropy. Moreover, the evaluation process can be context-aware and enhanced with external data that provides up-to-date insights into specific environmental attributes, e.g., dynamic cardinality of data flows within specific categories used as the threshold. Delving deeper, the outcomes of evaluations can be represented as comprehensive data structures that extend beyond simple allow or deny actions.

It is essential to highlight that although the OPA engine does not feature a built-in deconfliction primitive, our Rego policies have been thoughtfully defined to incorporate a deconfliction process that is both history- and precedence-aware. As a result, based on our generic policy, the engine is able to accurately state the union of actions permitted by satisfied constraints and those required due to constraint violations, denoted as $\alpha_{constraints} = \alpha^+ \cup \alpha^-$. Any overlap or conflict within $\alpha_{constraints}$ is deliberately left unresolved at this stage. Instead, conflicts are addressed through a dedicated deconfliction process defined within the Rego policy itself, executed by the OPA engine to exclude actions that have already been satisfied according to the recorded history and that conflict due to precedence relations. Therefore, there is no necessity to implement a custom unified actions deconfliction mechanism within the *process()* method of the Kafka Streams Processor API; the Kafka Streams microservice simply utilizes the α_{active} returned by the OPA engine to enforce dissemination actions.

```

1 package data.pipeline
2
3 default is_encrypted := false
4
5 is_encrypted if {
6   input.headers.entropy_score >= data.entropy_threshold
7 }
8
9 allow_write if {
10   not is_encrypted
11 }
12
13 next_state := "write_enc" if { allow_write }
14 next_state := "discard" if { not allow_write }
15
16 reason := "Payload is unencrypted" if { allow_write }
17
18 decision := {
19   "allow_write": allow_write,
20   "is_encrypted": is_encrypted,
21   "entropy_score": input.headers.entropy_score,
22   "next_state": next_state,
23   "reason": reason,
24 }
25
26
27

```

INPUT

```

1 {
2   "headers": {
3     "entropy_score": 4.1823,
4     "payload_size_bytes": 46,
5     "timestamp": "2026-05-24T10:00:00Z"
6   },
7   "data": "Hello world of entropy!"
8 }

```

DATA

```

1 {
2   "entropy_threshold": 7.5
3 }

```

OUTPUT

```

1 {
2   "allow_write": true,
3   "decision": {
4     "allow_write": true,
5     "entropy_score": 4.1823,
6     "is_encrypted": false,
7     "next_state": "write_enc",
8     "reason": "Payload is unencrypted"
9   },
10  "is_encrypted": false,
11  "next_state": "write_enc",
12  "reason": "Payload is unencrypted"
13 }

```

© 2026 Open Policy Agent contributors. OPA Slack OPA v1.16.0, Regal v0.39.0

Figure 4. Rego policy example.

Audit and Logging Component

The Kafka State Stores within the Streams Microservice layer have the capability to collect IFC-relevant events, providing valuable insights for the Audit and Logging component. These analytics can be processed by non-human interfaces, such as machine learning engines, and integrated with the Distributed Ledger layer for the purpose of policy tuning. Essentially, the State Store is an in-memory or persistent disk-resident key-value table backed by a dedicated Kafka topic, while ensuring Kafka's security and resiliency properties include a durability guarantee. Moreover, it supports microservices for the storage and retrieval of their intermediate results, allows for interactive read-only queries by the external applications, and facilitates the microservice state restoration mechanism.

6.2. Resource-Constrained Setup Benchmarking

In our previous research [9], we conducted performance studies (benchmarks) on various processing systems (Streams Microservice layer), data dissemination systems (Data Queue layer), and distributed ledger systems (Ledger layer). Moreover, we explored the Computing Continuum concept, which involves relocating the ledger peer from the Cloud to the Fog location. In these benchmarks, we assessed several key metrics: latency, throughput, resource usage (both CPU and RAM), and power consumption. To facilitate this evaluation, we established two configurations: an AWS cloud setup where we measured average data consumption times within a Kafka Streams-based microservice pipeline, emphasizing audiovisual stream processing and overall system stability; and a configuration on resource-constrained devices (Raspberry Pi), where we focused on measuring the duration of the most time-consuming operations within the microservices. Furthermore, we conducted a comparative analysis of latency performance between Java (Kafka Streams) and Go (Sarama) implementations, concentrating on two distinct latency metrics: event-time latency and processing-time latency. Event-time latency refers to the interval between the timestamp assigned to an input message by the producer and the point at which the system under test (SUT) generates an output message. Conversely, processing-time latency indicates the duration between the SUT ingesting (reading) an input message and the generation of an output message.

In this article, we benchmark the processing-time latency for the OPA decision engine, a crucial component of the Policy Enforcement layer. To achieve this, we made slight

modifications to our resource-constrained setup (Table 3 and Figure 5) and proposed two deployment configurations for the OPA engine: (i) co-locating it with the enforcement microservice within the Fog Processing layer, and (ii) deploying it on separate devices, with the OPA engine relocated to the Cloud Processing layer to simulate network overhead and communication through an IoT gateway (Comms layer).

Notably, our benchmarks focused solely on the enforcement microservice implemented in Java, which communicates with the decision engine via the HTTP API. Future work will include benchmarking a Go-based microservice that integrates the OPA engine as a library. Additionally, we take into consideration our earlier findings related to Java's just-in-time dynamic compilation, which can lead to variability in internal operation latencies during the initial execution phase, referred to as the warm-up state.

Table 3. An overview of the Raspberry Pi device-based environment components.

Framework Layer	Hardware/Software Component	Description
Communication	Gigabit switch and an IEEE 802.11 router	At the Tactical level (unit area of responsibility, AoR), three zones (subunit command posts) were designated to simulate geographical distances.
Publishers/Subscribers	Three RPi3 (CPU: 1.4 GHz Arm Cortex A53 and Memory: 1 GiB)	Devices were deployed at each subunit AoR to be used as producers and consumers.
Data Queue	Cluster of the Apache Kafka that consist three brokers: RPi5 devices (CPU: 2.4 GHz Arm Cortex A76 and Memory: 8 GiB)	Cluster was deployed in the edge processing location. Furthermore, each broker was configured with a single partition and a replication factor of three.
Streams Microservice	Two different devices: RPi5 (CPU: 2.4 GHz Arm Cortex A76 and Memory: 8 GiB), and SLap (CPU: 3.3 GHz i5-12450H and Memory: 16 GB)	Resource-constrained device was used to test Java microservices. The SLap platform was deployed as a reference benchmark.
Policy Enforcement	Same resources as for Streams Microservice layer.	Resource-constrained device was used to test OPA decision engine. The SLap platform was deployed as a reference benchmark.
Distributed Ledger	Hyperledger Fabric Blockchain Network (ver. 2.2) with two nodes (peers) per organization: VMs (vCPU: 2 and Memory: 2 GiB)	Ledger peers were deployed on a server using virtualization technology (data centers processing location).

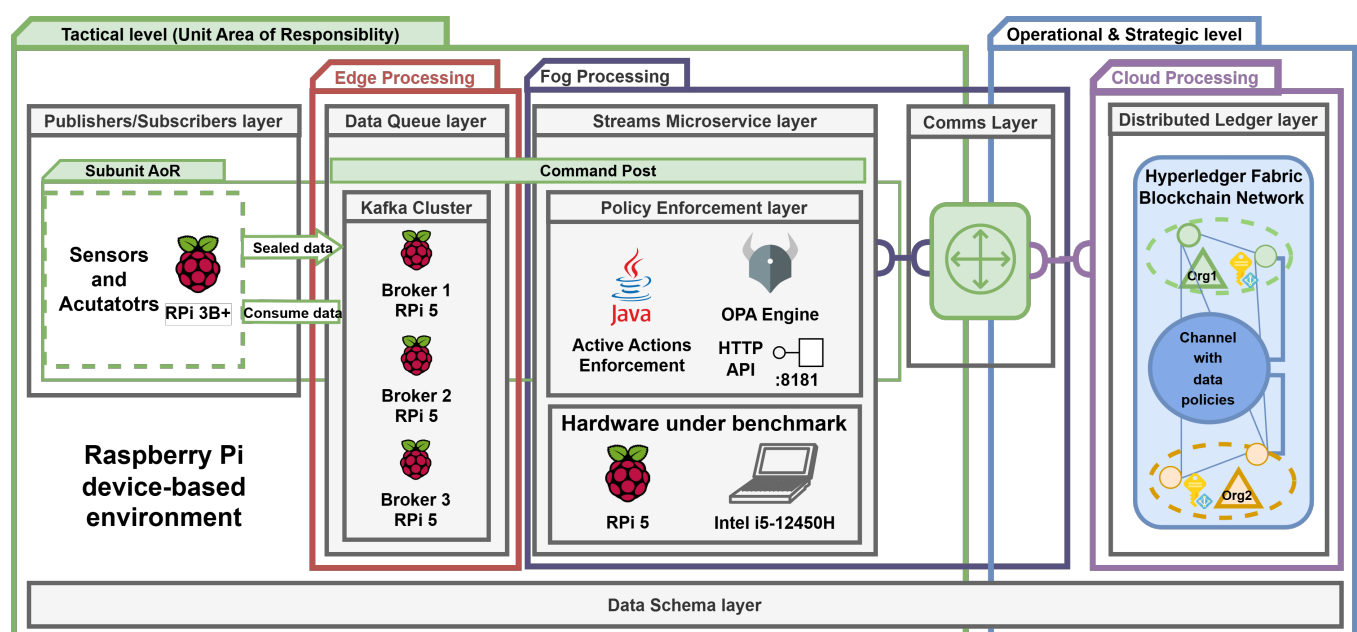


Figure 5. An overview of the Raspberry Pi device-based environment.

To assess the performance of the OPA engine and the enforcement microservice, we implemented a real-time monitoring stack that leverages the node exporter service, Prometheus, and Grafana. This setup facilitates the tracking of device resource utilization during benchmark tests and helps identify potential anomalies. The stack accumulates a comprehensive array of hardware- and OS-level metrics, including RAM, I/O disk, and CPU usage. Additionally, we utilized the revision-agnostic tool *vcgencmd* on the RPi5 platform to monitor throttling events, temperature, and power consumption. This monitoring is essential for detecting issues that may impair CPU instructions per clock, which could negatively impact the execution of both the microservice and the OPA engine.

Latency results presented below are averaged from 10 launches of the microservice and OPA engine. We employed the following metrics: average, average absolute deviation, minimum, population standard deviation, and percentiles p90, p95, and p99. These metrics were derived from HTTP request-response latencies collected from four distinct sources: the OPA engine service logs; the OPA engine's HTTP endpoint exposed for Prometheus, specifically the `http_request_duration_seconds_bucket` endpoint; the enforcement microservice itself, using the `Java System.nanoTime()` method to measure the time interval required to receive a decision from the OPA engine, `opaClient.evaluate(policyId, opaInput)`; and from a curl-based load-generation tool. We planned the following benchmark scenarios:

- Scenario I (Isolated): involves a single-threaded, isolated evaluation of our generic deconfliction policy's cost, excluding any overhead from the OPA server engine. We utilize native measurement tools such as `opa bench` and `opa eval --profile`, which are detailed in the official documentation [11] regarding the high performance of OPA policy decisions. These tools enable us to characterize the algorithmic behavior of the policy and identify latency bottlenecks, particularly concerning early exits and data- and size-dependent rules, such as `some h in input.history`. We acknowledge that these factors may impact OPA's decision-making latency. To address this, we plan to parametrize the history element by its size ($n \in \{0, 1, 100, 1000, 10,000, 100,000\}$), shape (array $O(n)$ versus object $O(1)$), and composition, including both representative and adversarial (worst-case) scenarios, where every history element fails to match the policy action set, thus requiring traversal of the entire history.
- Scenario II (Shared): introduces the OPA decision engine server and the enforcement microservice, alongside the associated network and data serialization overhead. Both services operate on the same hardware under test conditions.

6.3. Results

Scenario I (Isolated)

Our primary objective was to assess how the structure of the history representation influences the processing-time latency. The results clearly demonstrate a significant architectural distinction: the key-object (map) representation of history, which allows for $O(1)$ lookup, remains unaffected by the size of adversarial history. In contrast, the array representation, characterized by an $O(n)$ traversal cost, shows a nearly linear increase in latency as history size escalates. For the array implementation under adversarial conditions, latency soared to 5.83 ms with 1000 elements and reached 47.6 ms with 10,000 elements—a growth trend that, if unaddressed, could leave the decision engine susceptible to denial-of-service attacks from either crafted or organically large inputs. In light of these findings, we have adopted the key-object history structure as the default representation for all subsequent benchmarks.

Using the `opa eval --profile` command on the Raspberry Pi 5, we focused on the `timer_rego_query_eval_ns` metric, which directly relates to policy decision latency, as it measures only the query evaluation against the in-memory data store. With the key-object

structure, evaluation latency remained largely unaffected by the size of the history. We observed a minimum latency of 0.66 ms, an average latency of 0.86 ms, and a 90th percentile latency of 1.62 ms. To further distinguish the steady-state evaluation cost from one-time preparation overhead—which includes data loading, parsing, and compilation—we also employed `opa bench`, which repeatedly evaluates a single query along its most efficient execution path. In this benchmark, latency again proved insensitive to history size, yielding a minimum latency of 0.25 ms, an average latency of 0.32 ms, a p90 of 0.35 ms, and a standard deviation of approximately 0.013 ms. Taken together, these two tools indicate that, once compiled, the key-object-based deconfliction policy imposes a small and highly predictable per-decision evaluation cost, largely decoupled from the volume of historical data being reasoned over.

Scenario II (Shared)

Building on the previously mentioned findings, we focused Scenario II on the key-object history structure and limited the adversarial history size to 10 elements, which we deemed representative of realistic operational deconfliction contexts. For each test iteration, 1000 HTTP requests containing this adversarial history were sent to the OPA engine. First, we established reference latencies using a `curl`-based load-generation tool that concurrently captured three latency sources outlined in our methodology: tool-observed latency, OPA server logs, and Prometheus histogram data.

With up to three concurrent queries, the `curl` tool recorded a p90 latency of 2.69 ms, while the OPA server logs indicated a p90 of 1.72 ms. Prometheus-based resource monitoring confirmed that the OPA engine remained stable even with up to six concurrent requests (Figure 6). At this increased concurrency level, latency experienced a corresponding rise: the server-log p90 increased to 4.05 ms, while the `curl`-tool p90 escalated to 7.74 ms.



Figure 6. Grafana monitoring—OPA engine benchmarking (* denotes a non-null field value).

Subsequently, we conducted measurements using our Java-based enforcement microservice as the OPA client, operating under a single-producer workload. The microservice recorded an average HTTP API response latency of 1.67 ms (with an average deviation of 0.26 ms), a minimum latency of 1.23 ms, and a p90 latency of 2.30 ms. These results align closely with, and are slightly higher than, the `curl`-based reference measurements taken under similar load conditions (p90: 1.62 ms via the tool, 1.09 ms via server logs, and 2.03 ms

via Prometheus), indicating that the additional latency introduced by the Java HTTP client and the serialization process is modest.

Regarding power and thermal behavior, the RPi 5 exhibited stable power consumption during OPA decision processing, averaging 4.15 W with a p90 of 5.46 W. The device's temperature remained at or below 60 °C, with no instances of thermal throttling observed throughout any of the benchmark runs. This indicates that the platform operated comfortably within its safe operating range, even under sustained decision-evaluation loads.

7. Threat Model

In this section, we present a threat model for the proposed *FedM* structure. The policy-driven data dissemination framework operates within federated environments in which processed data often include sensitive information. Accordingly, unauthorized disclosure or manipulation of such data may expose network participants to significant material, operational, or reputational risks.

The primary objectives of attackers in federated IoT ecosystems generally include stealing or forging data through compromised IoT devices, disrupting the communication layer, or utilizing these devices to launch D4M attacks that hinder access to federated services. While the types of attacks relevant to federated IoT networks largely align with those found in non-federated networks, it is crucial to differentiate adversaries that originate from organizations other than the owner of the targeted devices or services. These adversaries typically have access rights that are greater than those of a fully external attacker but fewer than those of a member of the owning organization.

In our earlier work [9], we conducted a high-level security risk assessment that addressed multiple security and reliability threats spanning the application, network, and perception layers of the dissemination framework. In this article, we focus specifically on three threats: implicit flows, the benevolent developer, and the behavior of the distributed ledger under degraded network conditions. Our threat model delineates the adversary's capabilities, the trust boundaries between entities, and the assumptions under which each security guarantee is valid. Consequently, we identify the following key groups of actors within our threat model:

- Authenticated, policy-compliant entities, E , operating within their assigned capabilities, E_{Caps} ;
- Enforcement microservices that concurrently execute the decision procedure, $\mathcal{D}_A$, treated as a trusted, tamper-resistant oracle, whose output is exclusively bounded by selected policy $\alpha_{active} \subseteq \pi_{id,p} \cdot \alpha$;
- Peers (nodes) of the distributed ledger, $DL.P$, are assumed to be Byzantine fault-tolerant under the underlying consensus protocol, provided that fewer than one-third of the federation nodes are compromised, $p < |DL.P|/3$. Additionally, the data channels, $DL.CH$, are responsible for maintaining the policies, Π , data schemas, $\mathcal{S}$, parameter contexts, Θ , and entity fingerprints, FID , as well as the audited transactions, $DL.TX$;
- The policy author, instantiated as the benevolent developer, who is assumed honest but potentially careless in policy authorship;
- External adversary who cannot break the cryptographic primitives underlying entity authentication;
- Internal attacker—such as a compromised device—who cannot forge valid entity fingerprints, FID , under the assumption that both the ledger channels, $DL.CH$ and transaction logs, $DL.TX$, remain uncompromised.

The aforementioned delineates the boundaries between trusted and potentially adversarial components and provides the foundation for the subsequent analysis of implicit flows, policy authoring risks, and ledger resilience under degraded connectivity.

Implicit Flows

We define an implicit flow as any transfer of a data element, $d \in D$, to an entity, $e \in E$, that occurs without a corresponding action in α_{active} that explicitly enforces it. The Policy-based Data Flow Preservation Theorem 1 directly addresses this threat: because our decision procedure, $\mathcal{D}_A$, is deterministic (Lemma 3) and terminates (Lemma 4), and restricts its output to the action subset defined by the selected policy, $\alpha_{active} \subseteq \pi_{id,v} \cdot \alpha$ (Lemma 2), any flow of d to e that cannot be traced back to a specific action in α_{active} is, by definition, excluded from the model's execution space. This guarantee holds under the assumption that all data movement between entities occurs solely through the topic primitive, $t \in T$, which is structured across the four topic classes T_{unauth} , T_{auth} , T_{inter} , and $T_{delivery}$. Access to these topics is governed by the global guard mechanism, $\mathcal{G}$, and function, $canRead(e, d, t)$, which are contingent upon the existence of a mapping E_T . By design, access to topics outside this mapping is excluded from the model, and any information flows through such paths do not qualify as implicit flows within the model's scope but rather represent boundary conditions for Theorem 1. Finally, we explicitly assert that the guarantee against implicit flows does not extend to covert channels E_T .

Benevolent Developer

Unlike the malicious insider, the benevolent developer threat stems from unintentional policy misconfigurations rather than deliberate misuse. This is based on the assumption that the developer has legitimate authority to author and version policies $\pi_{id,v} \in \Pi$ along with all associated structural elements, including the schema binding, the parameter contexts $\pi_{id,v} \cdot \theta \subseteq \Theta$, the label tuple scope $\pi_{id,v} \cdot \lambda \in L$, and the constraint set, $\pi_{id,v} \cdot \phi_S \subseteq \Phi_S$. Consequently, the attack surface is limited to structurally or semantically flawed definitions of these elements, rather than to unauthorized system access.

Our model addresses this threat through three complementary mechanisms. First, the properties of precedence relations, the general discard invariant, and deconfliction completeness collectively reject structurally invalid policies prior to deployment. This includes the detection of malformed precedence relations and unresolved action conflicts, regardless of whether the flaw was intentional. Second, the context parameter that is admissibility invariant ensures that parameters selected from the canonical federation dimension value space, L_{dim} , are formally constrained by this dimension's value space. Third, the schema-policy binding relation is registered within a distributed ledger and is subject to auditing, $DL.TX$. This registration enables the detection of incorrect schema bindings—such as those that misalign constraints against an unintended field vocabulary or misencoded label tuples—during ledger audits before enforcement takes place. Additionally, explicit policy versioning allows for the identification and suppression of erroneous policy definitions without disrupting the historical record of enforcement decisions made under those definitions.

Notably, these mechanisms effectively mitigate both detectable structural errors and a subset of semantic misconfigurations; they do not resolve cases where semantically valid policies misrepresent the author's intended meaning within the formal boundaries. For example, a misspecified label tuple value in parameter contexts could inadvertently permit unintended data elements to be written, even while the data policy remains structurally well-formed.

Distributed ledger under degraded conditions

The distributed ledger DL serves as the authoritative registry for all structural elements utilized by the enforcement microservice, such as policies, Π , data schemas, $\mathcal{S}$, parameter contexts, Θ . We assume that under degraded network conditions—such as

network partitions or temporary disconnects—the enforcement microservice continues to operate with data policies and schemas stored locally (cached). This cache is populated upon the first successful retrieval of the selected data policy from a ledger node. We consider that this caching mechanism may also be integrated with ledger chaincode execution, for instance, during the registration of a new data policy.

Importantly, this design embodies a deliberate trade-off between availability and consistency. In essence, the model prioritizes continued enforcement—even based on potentially outdated data policies—over halting the dissemination process during periods of network degradation, rather than ensuring strong consistency of the policy state across the federation at all times.

A related threat arises within the verification mechanism, where entity identities must be safeguarded while remaining enforceable. In our previous work, we introduced the Computing Continuum Concept, positioning a ledger node at the tactical level [9]. This node must be secured and, under degraded conditions, should only contain the identities of entities operating within its vicinity. Consequently, asset transfers between federated organizations may be restricted under such circumstances, as the necessary identity and policy information might not be accessible beyond the local tactical domain. This limitation aligns with the availability-oriented design of the ledger: while the system remains operational locally, cross-domain operations reliant on current identity and policy data may face temporary constraints.

8. Formal Verification and Assessment

In Section 4, we outlined our formal model for Secure and Context-Based Data Dissemination in Federated Special IoT Environments (referred to as *FedM*), along with the Policy-based Data Flow Preservation Theorem proofed by four lemmas. We established that our policy enforcement procedure (Formula (5)) is both deterministic and terminating, facilitating the evolution of the resulting policy actions, α_{active} , through awareness of history and precedence. To strengthen confidence in the correctness of the *FedM* model and the Flow Preservation Theorem, we conducted formal verification using the *P framework* [12]. We represented the primitive elements of our model as communicating state machines that disseminate asynchronous events illustrated in Figure 7, thereby framing the *FedM* as a distributed system (P-model) that executes each stage of the policy enforcement procedure shown in Figure 8, including steps such as policy fetching (FetchingPolicyStep), union of constraint actions (UnionActionsGathering), and the deconfliction step (Deconfliction). At the same time, we observed and ensured the preservation of the properties established by Lemmas 1–4 (Table 4).

Table 4. Liveness and safety properties.

Type	Name	Monitors	Description
liveness	AllMessagesAreHandledbyPEP	Implicit flows	Every produced and verified message is eventually acknowledged by PEP.
liveness	AllProducedMessagesTerminate	Lemmas 2 and 4, implicit flows	Every produced and verified message eventually reaches delivery or discard stage.
liveness	PolicyDBAlwaysResponds	Lemma 3, policy compliance	Every policy request eventually receives a matching unambiguous response.
safety	DisseminationPipelineOrderSafety	All lemmas	Produced messages progress through pipeline stages only in valid order.

Table 4. Cont.

Type	Name	Monitors	Description
safety	DiscardIsTerminal	Lemma 4, discard invariant	Discarded message can never re-enter processing or be delivered.
safety	PrecedencePropsPreserved	Lemma 2, benevolent developer	Precedence relation associated with each policy is verified to remain irreflexive, asymmetric, and transitive before being used in deconfliction step.

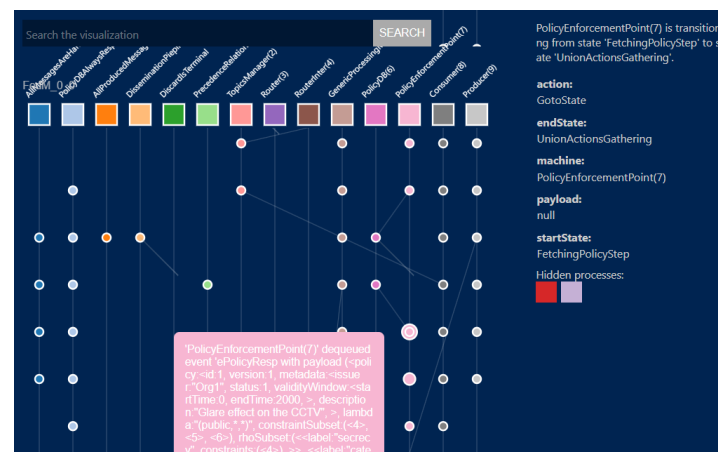


Figure 7. FedM model primitives represented as state machines (* denotes a wildcard value).



Figure 8. Policy enforcement state machine.

The correctness of the P-model was evaluated through a set of monitor (observer) machines that verify whether the system adheres to both safety and liveness properties. Collectively, these properties ensure that the system reaches a cold state upon completion of execution—implying that every event is eventually handled (liveness) and that every response complies with the model’s invariants (safety). These properties, summarized in Table 4, support the Policy-based Data Flow Preservation Theorem and primarily focus on threat model assumptions regarding implicit data paths, as well as the necessity to maintain the order of functional blocks within the dissemination pipeline (Section 5).

Since the developed P-model represents a continuously operating data dissemination system, the PEP state machine is designed to persistently await and process incoming messages until an explicit termination condition is reached, `param globalMsgCntParam : int; .` This design reflects the semantics of real-world streaming systems, where components are expected to operate indefinitely rather than terminate after a finite computation. Consequently, most verification (checks) executions are inherently long-running and effectively non-terminating within any practical step bound. In this study, we adopt the default maximum step bound of the P checker (`-ms 10000`). Notably, reaching this bound is not treated as an error condition; rather, it is consistent with the intended always-on operational semantics of our model.

Each schedule represents a distinct interleaving of asynchronous events, message deliveries, and internal state transitions across concurrently executing machines. To en-

sure coverage of different behaviors, the checker employs pseudo-random seeds, which influence scheduling decisions and enable reproducibility of explored traces. Under seed 901282872 (`--seed 901282872`), the checker explored up to 10,000 schedules for the single-producer configuration (`-tc tcSingleProducer`), yielding 37 distinct execution timelines. These timelines are equivalent classes of schedules with different observable behaviors. No property violations (bugs) were detected, and all schedules exceeded the bound. For configurations involving multiple producers (`globalProducerCountParam in [2,3,4]`) and multiple consumers (`globalConsumerCountParam in [2,3,4]`), the checker generated 72 distinct timelines—for two producers and consumers—again with zero reported bugs. Across both experimental scenarios, no violations of the specified liveness or safety properties were observed within the explored execution horizon.

Despite these results, it is important to acknowledge a fundamental limitation of bounded exploration when applied to non-terminating systems. Because most schedules do not terminate within the specified step bound, verification of liveness properties is necessarily partial: such properties are evaluated only over finite execution prefixes rather than complete executions. Consequently, while the absence of counterexamples provides empirical confidence in the correctness of the model, it does not constitute a formal proof of liveness under unbounded execution. This limitation is inherent to simulation-based and bounded verification approaches and should be considered when interpreting the results.

Furthermore, within the P-model, we instantiate example messages and data policies whose structures are consistent with the definition of the *FedM* policy element. To exhaustively explore possible constraint outcomes, the policy enforcement state machine evaluates each policy constraint non-deterministically—the dollar sign for branch in Formula (17). This construction enables the model checker to explore system behavior under all combinations of satisfied and violated constraints, thereby exercising the full space of constraint outcomes and ensuring that policy enforcement logic is validated across all feasible enforcement scenarios.

Non-deterministic function for policy constraint outcomes in P framework

```

alphaConstrains = CollectRawActions(resp.policy);                                     (17)
fun CollectRawActions(policy : tPolicy) : tActionSet{
  var raw : tActionSet;
  var i : int;
  var branch : bool;
  i = 0;
  while(i < sizeof(policy.rules)){
    branch = $;
    if(branch){
      raw = MergeActions(raw, policy.rules[i].pos);
    } else {
      raw = MergeActions(raw, policy.rules[i].neg);
    }
    i = i + 1;
  }
  return raw;
}

```

From the perspective of the security-by-design concept [33]—which advocates for integrating security into a system from the outset, rather than as an afterthought—the developed P-model facilitates formal verification and validation of data policies prior to their deployment in a production system. Specifically, through the *PrecedencePropsPreserved* specification, precedence relations can be validated to ensure they retain essential properties such as irreflexivity, asymmetry, and transitivity. This early-stage validation aids in identifying inconsistencies during the modeling phase, thereby preventing costly integration efforts and runtime debugging later on.

Furthermore, the P-model allows for the evaluation of whether the target system will function correctly before deployment, accounting for arbitrary data flows and varying component counts, including different numbers of producers and consumers. By systematically exploring multiple execution schedules and component configurations, the model can assess robustness against scaling effects—such as increased message rates, concurrent producer activities, and heterogeneous consumer behaviors. This capability is particularly beneficial in federated and multi-domain environments, where the number of participating systems may fluctuate dynamically and policy correctness must be maintained under diverse operational conditions.

In summary, our model enables contextual, policy-driven data dissemination in distributed environments while prioritizing the protection of data flows and the reliability of their delivery. This necessitates a precise definition of the participating entities, communication data channels, data elements along with their labeling structures, and the data policy structure that explicitly governs data dissemination and processing pipelines. Consequently, our model:

1. Encapsulates a federated data network, N , alongside a set of evolving and versioned data policies, Π , which determine processing of data flows and their dissemination among authenticated and authorized entities, E . Notably, a comprehensive multi-dimensional, Δ_{dim} , discrete and continuous (bounded) labeling structure, L , can be assigned to data at runtime. Concurrently, various data processing strategies can be applied depending on the outcome actions delineated by the policy constraints, $\gamma_{\alpha, \phi}$.
2. Defines the deterministic policy enforcement decision procedure, $\mathcal{D}_A$, which incorporates a deconfliction process. This process determines both the actions permitted by the satisfied policy constraints and those that must be enforced due to their violations. Essentially, our deconfliction process utilizes a precedence relation, $\prec_{\alpha}$, along with a historical sequence of already enforced actions, $\mathcal{H}_d$, that is tied to the data throughout its flow. In addition, this enables the data policy-enforce mechanism lifecycle to have an independent and longer span than the lifecycle of entities that process data. Notably, if procedure executions yield an empty set $\alpha_{active} = \emptyset$, the discard invariant applies uniformly, preserving the general deny guarantee regardless of history;
3. Formalizes a federated, decentralized, and organized structure of the Distributed Ledger Technology, DL , which redundantly stores unique identities of entities, FID , as well as the registered data policies, Π , and data schemas, $\mathcal{S}$. Furthermore, the DLT enables the seamless transfer (mobility) of entities between organizations within the federation. Simultaneously, this allows to leverage another organization's infrastructure for secure data dissemination. In other words, the operator in the command and control post of *Org1* can enhance Situational Awareness about environmental conditions within a specific area that belongs to the federated constituency of *Org2*.
4. Outlines a comprehensive set of capabilities, $Caps$, for entities, E , such as data production (writing) and consumption (reading). Additionally, to mitigate the effects of malicious activities by entities while considering relevant computing concepts, we

proposed the set of capabilities to enable the utilization of processing functions, PF , such as modification and assessment functions.

5. Utilizes different topic classes, T , that isolate data at rest and enable a communication mechanism incorporated by entities from the moment of data production, through provenance verification, policy enforcement, and any subsequent processing, until the delivery moment when the data is consumed;
6. Establishes invariants that define the scope of read and write operations for entities, E , related to specific topics, T , utilizing the global guard mechanism, $\mathcal{G}$, along with the *canRead* and *canWrite* functions. At the same time, this guard mechanism regulates the allowable dimensions and labels that microservices can handle, ensuring that the addition of future topics or entity types does not unintentionally bypass any of its checks.
7. Enables entity characteristic-based verification of data origin and authenticity by asserting a threat model, where an entity might be captured by a malicious organization and potentially manipulated (forged) while still retaining the ability to produce valid proofs. Notably, our solution enables the detection of such a specific attack vector, as the entity's fingerprint samples, FID , can be evaluated at any time to ensure they correspond with the characteristics stored in the distributed ledger, DL . Consequently, this attack vector would necessitate manipulation of both the ledger channels and transaction logs, making such an attack unlikely to succeed.

9. Conclusions and Future Work

At the beginning of this article, we introduced the high-level architecture of the intelligence platform (Figure 1), which presents several challenges related to security, reliability, and contextual data dissemination. In response, we have endeavored to formalize our contributions to addressing these challenges, with a particular focus on contextual dissemination. We proposed a multi-level, flexible formal model for the Secure and Context-Based Dissemination of Data within Federated Special IoT Environments (Section 4). Notably, our model is built upon various access control models, Denning's research on information flow control (Section 2), and the requirements for the IFC architecture discussed in Section 3.

In summary, our model systematically incorporates both access and dissemination controls, where we prioritize the protection of data flows and the reliability of their delivery. For this purpose, we defined robust policy-driven enforcement elements, which enable tracking and management of data dissemination in accordance with evolvable environmental conditions. To mitigate the uncertainty indicator, simultaneously achieving a clearer understanding of the environment around the entity, we defined processing primitives and essentially determined the quality assessment elements that are fundamental to the continuous measurement of data quality—whether objective or subjective—known as the Data Quality Assessment process. Moreover, our model strictly aligns with the concepts of Ubiquitous and Continuum Computing. To reinforce the above statements, in Section 5, we presented an example of a data dissemination pipeline driven by the data policy. In addition, we explained the application of a bounded trustworthiness dimension—a significant aspect of modern Situational Awareness environments—along with the data quality assessment microservice through the utilization of our model elements.

Furthermore, our model places a significant emphasis on the detailed management of actions that can be implied on the data and their appropriate assignment to the processing entities that interact with that data. This is crucial in environments where stringent accountability is required at every stage of the system's lifecycle. Therefore, we utilized a structured component denoted as the Distributed Ledger, which facilitates the application and dynamic modification of the diverse dimensions of label expressiveness. Notably,

the decentralized nature of the ledger enhances resilience against various D4M attacks while also mitigating the risks posed by the benevolent developer by separating the policy logic from functionality of context-aware applications. Moreover, the use of a consensus protocol ensures that any policy change receives approval from all involved parties, with chaincodes enabling these changes to be executed consistently and procedurally. Additionally, we have thoughtfully integrated the verification of data authenticity and provenance into the data dissemination process, which facilitates a novel cybersecurity posture of Zero-Trust architectures. A comprehensive formal definition of the Zero-Knowledge-like verification procedure, along with its formal analysis, will be addressed in our forthcoming article. Notably, we proved that our policy enforcement procedure (Formula (5)) is both deterministic and terminating, facilitating the evolution of the resulting policy actions, α_{active} , through awareness of history and precedence.

To enhance confidence in the correctness of the *FedM* model and the Flow Preservation Theorem, we undertook formal verification utilizing the *P framework* [12]. By applying our model within the P verifier, we identified three key advantages from a security-by-design perspective: (i) policies are verified against both safety and liveness properties prior to integration; (ii) structural defects in precedence relations are detected before deployment; and (iii) the system's response to scaling and variability is comprehensively understood in a formal, reproducible manner. Consequently, our P model lays a robust foundation for embedding security guarantees directly into the design of data dissemination systems, rather than relying solely on empirical testing or post-deployment monitoring.

In addition to the formal model, we have refined our novel, multi-layered framework, which proposes a deployment of IFC components aimed at supporting policy-driven contextual data dissemination. Overall, our solution emphasizes a data-centric paradigm that enhances message-level granularity and improves information flow control, thereby enabling effective sharing of data and information among authenticated and authorized federated consumers. This improvement requires minimal re-engineering effort and primarily depends on existing technologies and components that have already been deployed within our multi-layered framework. A significant enhancement is the inclusion of the Open Policy Agent, which operates alongside enforcement microservices to facilitate decision-making as part of the Policy Enforcement Point component.

The deployment of the architecture described in Section 6, which integrates OPA, Kafka Streams, and Hyperledger Fabric, has been only preliminarily evaluated in this article. The experimental validation of the Policy Enforcement layer was limited to the OPA decision engine, and more extensive benchmarking of this engine is planned for future work. Specifically, the benchmarking of the distributed configuration—where the OPA engine is moved to the Cloud Processing layer and interacts with the enforcement microservice through the Communication layer and IoT gateway—was not completed in this study and will be explored in a separate forthcoming article. In that future work, we aim to utilize external packet-level analysis tools such as Wireshark to examine network transport overhead (including TCP/TLS handshake costs and round-trip times), in conjunction with our current HTTP- and application-level latency measurements. Furthermore, comprehensive end-to-end benchmarking of the integrated stack—including Kafka topic scalability, blockchain synchronization costs, and the overhead associated with ACL or label reassignment—represents a crucial direction for future research. We intend to assess the entire pipeline under increasing numbers of policies, topics, entities, and data flows in a dedicated experimental study.

In our opinion, the proposed model aligns well with the dynamic Observe–Interpret–Predict–Act cycle of Situational Awareness. Its practical deployment, which is our novel framework, holds significant promise as a foundation for various fixed and ad-hoc ap-

plications. For instance, an ad-hoc system for coordinating international efforts focused on HADR operations. In this scenario, our system would facilitate reliable data dissemination from CCTV cameras and health devices, such as SOS wristbands. This capability would significantly reduce response times for individuals needing assistance and enhance decision-making through improved information sharing.

Another potential application for a context-aware data dissemination system functioning within a federated IoT environment is border surveillance. This system would primarily rely on distributed multispectral sensors, including thermal imaging, daylight cameras, radar, and acoustic sensors. These devices would be stationed along the border and deployed on mobile platforms, such as UAVs. Data gathered from these sensors would be sent to the nearest distribution center, where it would be processed and forwarded to five coordination centers corresponding to various services within the federation: the army, police, border guards, fire department, and emergency medical services. The data processing might include filters to identify specific events, such as the detection of individuals at the border or unidentified objects crossing it. Subsequently, information about these events could be disseminated to patrol vehicles and individual soldiers, and if necessary, to medical services. Concurrently, the data might also be disseminated to a separate analytical server that aggregates operational and technical data for further analysis by the border guard commander.

Author Contributions: Conceptualization, J.S. and Z.Z.; methodology, Z.Z.; software, J.S.; validation, J.S.; formal analysis, J.S. and Z.Z.; investigation, J.S. and Z.Z.; resources, Z.Z.; data curation, J.S.; writing—original draft preparation, J.S.; writing—review and editing, J.S. and Z.Z.; visualization, J.S.; supervision, Z.Z.; project administration, Z.Z.; funding acquisition, Z.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACL	Access Control List
AIOs	Aggregated Information Objects
API	Application Programming Interface
BLP	Bella–LaPadula
CCTV	Closed-circuit television (video surveillance)
D4M	Deny, Delay, Disrupt, Destroy, or Manipulate
DAC	Discretionary Access Control
DQA	Data Quality Assessment
DLT	Distributed Ledger Technology
HADR	Humanitarian Assistance and Disaster Relief
IFC	Information Flow Control
IoT	Internet of Things
KPIs	Key Performance Indicators
LiDAR	Light Detection and Ranging
MAC	Mandatory Access Control
OPA	Open Policy Agent
Org	Organization
PEP	Policy Enforcement Point

RPi	Raspberry Pi
SA	Situational Awareness
UAV	Unmanned Aerial Vehicle
WDR	Wide Dynamic Range
ZT	Zero-Trust

References

- Weiser, M. The Computer for the 21st Century. *Sci. Am. Mag.* **1991**, *265*, 94. <https://doi.org/10.1038/scientificamerican0991-94>
- Zaccarini, M.; Cantelli, B.; Fazio, M.; Fornaciari, W.; Poltronieri, F.; Stefanelli, C.; Tortonesi, M. VOICE: Value-of-Information for Compute Continuum Ecosystems. In Proceedings of the 27th Conference on Innovation in Clouds, Internet and Networks, Paris, France, 11–14 March 2024; pp. 73–80. <https://doi.org/10.1109/ICIN60470.2024.10494500>
- Suri, N.; Wrona, K.; Zieliński, Z. *Military Applications of Internet of Things: Architectures, Security, Reliability, and Interoperability for MIIoT*; Springer Nature: Cham, Switzerland, 2026. <https://doi.org/10.1007/978-3-032-03378-9>
- Almuntasheri, S.; Alrumaih, T.; Alablani, I.; Jabr, B.; Alenazi, M. A survey on Battlefield Management Systems: Architecture, challenges, and future research directions. *Alex. Eng. J.* **2026**, *147*, 93–116. <https://doi.org/10.1016/j.aej.2026.05.038>
- U.S. Army Cyber Command. Zero Trust: What Is Zero Trust? 2023. Available online: https://www.army.mil/article/264818/zero_trust (accessed on 27 July 2026).
- Byabazaire, J.; O'Hare, G.; Collier, R.; Delaney, D. IoT Data Quality Assessment Framework Using Adaptive Weighted Estimation Fusion. *Sensors* **2023**, *23*, 5993. <https://doi.org/10.3390/s23135993>
- Sychowiec, J.; Zieliński, Z. An Experimental Framework for Secure and Reliable Data Streams Distribution in Federated IoT Environments. In Proceedings of the 18th Conference on Computer Science and Intelligence Systems (FedCSIS), Warsaw, Poland, 17–20 September 2023; pp. 769–780. <https://doi.org/10.15439/2023F3882>
- Pióro, L.; Sychowiec, J.; Kanciak, K.; Zieliński, Z. Application of Attribute-Based Encryption in Military Internet of Things Environment. *Sensors* **2024**, *24*, 5863. <https://doi.org/10.3390/s24185863>
- Sychowiec, J.; Zieliński, Z. A Blockchain-Based Framework for Secure Data Stream Dissemination in Federated IoT Environments. *Electronics* **2025**, *14*, 2067. <https://doi.org/10.3390/electronics14102067>
- Denning, D.E. A lattice model of secure information flow. *Commun. ACM* **1976**, *19*, 236–243. <https://doi.org/10.1145/360051.360056>
- Cloud Native Computing Foundation. Open Policy Agent—Documentation. Available online: <https://www.openpolicyagent.org/docs> (accessed on 27 July 2026).
- P Programming Framework—Publications. Available online: <https://p-org.github.io/P/publications/> (accessed on 27 July 2026).
- Penelova, M. Access Control Models. *Cybern. Inf. Technol.* **2021**, *21*, 77–104. <https://doi.org/10.2478/cait-2021-0044>
- Bell, D.E.; LaPadula, L.J. *Secure Computer Systems: Mathematical Foundations*; MTR-2547; MITRE Corporation: Bedford, MA, USA, 1973. Available online: https://archive.org/details/DTIC_AD0770768 (accessed on 27 July 2026).
- Biba, K.J. *Integrity Considerations for Secure Computer Systems*; MTR-3153; MITRE Corporation: Bedford, MA, USA, 1977. Available online: https://archive.org/details/DTIC_ADA039324 (accessed on 27 July 2026).
- Denning, D.E. *Cryptography and Data Security*; Addison-Wesley Longman: Reading, MA, USA, 1982. Available online: <https://faculty.nps.edu/dedennin/publications/Denning-CryptographyDataSecurity.pdf> (accessed on 27 July 2026).
- Grime, S.; Durrant-Whyte, H.F. Data fusion in decentralized sensor networks. *Control Eng. Pract.* **1994**, *2*, 849–863. [https://doi.org/10.1016/0967-0661\(94\)90349-2](https://doi.org/10.1016/0967-0661(94)90349-2)
- Bacon, J.; Eysers, D.; Pasquier, T.; Singh, J.; Papagiannis, I.; Pietzuch P. Information Flow Control for Secure Cloud Computing. *IEEE Trans. Netw. Serv. Manag.* **2014**, *11*, 76–89. <https://doi.org/10.1109/TNSM.2013.122313.130423>
- Hosek, P.; Miglavacca, M.; Papagiannis, I.; Pietzuch, P.R. *SafeWeb: A Middleware for Securing Ruby-Based Web Applications*; Springer: Berlin/Heidelberg, Germany, 2011; Volume 7049, pp. 491–511. https://doi.org/10.1007/978-3-642-25821-3_25
- Miglavacca, M.; Papagiannis, I.; Eysers, D.; Shand, B.; Bacon, J.; Pietzuch, P.R. DEFCon: High-Performance Event Processing with Information Security. In *USENIX Conference on Web Application Development*; USENIX Association: Berkeley, CA, USA, 2010. Available online: <https://api.semanticscholar.org/CorpusID:43859> (accessed on 27 July 2026).
- Zeldovich, N.; Boyd-Wickizer, S.; Mazières, D. Securing Distributed Systems with Information Flow Control. In *Symposium on Networked Systems Design and Implementation*; USENIX Association: Berkeley, CA, USA, 2008. Available online: <https://api.semanticscholar.org/CorpusID:8839666> (accessed on 27 July 2026).
- Burket, J.; Mutchler, P.; Weaver, M.; Zaveri, M.; Evans, D. GuardRails: A Data-Centric Web Application Security Framework. In *USENIX Conference on Web Application Development*; USENIX Association: Berkeley, CA, USA, 2011. Available online: <https://api.semanticscholar.org/CorpusID:8280509> (accessed on 27 July 2026).
- Denning, D.E.; Denning, P.J. Certification of programs for secure information flow. *Commun. ACM* **1977**, *20*, 504–513. <https://dl.acm.org/doi/10.1145/359636.359712>

24. Stefan, D.; Russo, A.; Mitchell, J.C.; Mazières, D. Flexible dynamic information flow control in Haskell. In *Haskell '11: Proceedings of the 4th ACM symposium on Haskell*; Association for Computing Machinery: New York, NY, USA, 2011; pp. 95–106. <https://doi.org/10.1145/2034675.2034688>
25. Davis, B.; Chen, H. DBTaint: Cross-Application Information Flow Tracking via Databases. In *USENIX Conference on Web Application Development*; USENIX Association: Berkeley, CA, USA, 2010. Available online: <https://api.semanticscholar.org/CorpusID:5441924> (accessed on 27 July 2026).
26. Feige, U.; Fiat, A.; Shamir, A. Zero-knowledge proofs of identity. *J. Cryptol.* **1988**, *1*, 77–94. <https://doi.org/10.1007/BF02351717>
27. Apache Kafka Documentation. Available online: <https://kafka.apache.org/43/getting-started/introduction/> (accessed on 27 July 2026).
28. Schema Registry for Confluent Platform. Available online: <https://docs.confluent.io/platform/current/schema-registry/index.html> (accessed on 27 July 2026).
29. Hyperledger Fabric Documentation. Available online: <https://hyperledger-fabric.readthedocs.io/> (accessed on 27 July 2026).
30. Apache Kafka Streams API Documentation. Available online: <https://kafka.apache.org/documentation/streams/> (accessed on 27 July 2026).
31. Sarama Go Library. Available online: <https://pkg.go.dev/github.com/shopify/sarama> (accessed on 27 July 2026).
32. Rego Policy Language. Available online: <https://www.openpolicyagent.org/docs/policy-language> (accessed on 27 July 2026).
33. EU: Common Assessment Method for Standards and Specifications, Security by Design. Available online: <https://interoperable-europe.ec.europa.eu/collection/common-assessment-method-standards-and-specifications-camss/solution/elap/security-design> (accessed on 27 July 2026).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.