

# Multi-Stakeholder Access Control in Data Ecosystems

Aditya Sissodiya

Cyber-Physical Systems



---

# Multi-Stakeholder Access Control in Data Ecosystems

**Aditya Sissodiya**

Department of Computer Science, Electrical and Space Engineering  
Luleå University of Technology  
Luleå, Sweden

---

**Supervisors:**

Ulf Bodin, Eric Chiquito and Johan Kristiansson



*to my grandmother, प्रेमलता ...*



---

# ABSTRACT

---

In multi-stakeholder data ecosystems, digital resources (e.g. shared datasets) are often co-owned by independent organizations, making access governance a collaborative challenge. Each stakeholder brings its own security and business constraints, so agreeing on concrete access rules across organizational boundaries is hard. For example, in an industrial IoT supply chain, machine data can be valuable to the factory operator, the equipment supplier, and a maintenance provider, each must consent to who can access what. Manual agreement is slow and contentious, motivating an automated negotiation mechanism to help stakeholders efficiently converge on a shared set of acceptable rules.

Consensus is only half the problem, once access rules are agreed upon, they must be enforced reliably in a distributed system with no single authority to push updates. Nodes go offline, networks partition, and rule updates arrive out of order. Without careful design, different parts of the system will enforce different decisions. Ensuring that all parties can consistently uphold the agreed rules under stated assumptions requires robust, fault-tolerant mechanisms. This licentiate thesis tackles both agreement and enforcement, providing methods to reach common ground on access rules and to apply them securely despite distributed-systems challenges.

The thesis makes three contributions. First, it introduces a utility-based negotiation method that lets multiple stakeholders collaboratively arrive at a common access-rule set by quantifying preferences and using optimization to automate consensus, supporting more structured and reproducible agreement compared to informal negotiation. Second, it develops a formal verification toolchain for cloud-native infrastructures (shown on Kubernetes) to prevent misconfiguration and privilege escalation; RBAC and admission rules are translated into logical constraints, an SMT solver checks for unsafe conditions, and only verified rules are deployed; an integrated deny-overrides enforcement path then applies them consistently at runtime. Third, it outlines EQuack (to be detailed in forthcoming work), an offline-capable access control model for distributed ecosystems where continuous connectivity cannot be assumed. It ensures that even if nodes diverge while offline, they eventually converge to the same authorized state via deterministic deny-wins replay over update logs and tamper-evident audit trails, without relying on blockchains or other heavy consensus mechanisms.

Taken together, the results suggest a path toward more secure collaboration by supporting structured agreement over access rules and providing enforcement mechanisms that can remain consistent in distributed settings, within the limits of the evaluated scenarios and stated assumptions.



---

# CONTENTS

---

<b>Part I</b>	<b>1</b>
CHAPTER 1 – INTRODUCTION	3
1.1 Problem Formulation and Research Questions . . . . .	4
1.2 Research Methodology . . . . .	7
1.3 Thesis Outline . . . . .	8
CHAPTER 2 – RESEARCH BACKGROUND	9
2.1 Distributed, Decentralised and Federated Systems . . . . .	9
2.2 Multi-Stakeholder Data Ecosystems . . . . .	10
2.3 Co-Owned Resources . . . . .	11
2.4 Access Control Models in Multi-Stakeholder Settings . . . . .	12
2.5 Policy Negotiation and Decision Support . . . . .	13
2.6 Enforcement and Formal Verification in Cloud-Native Systems . . . . .	14
2.7 Offline Enforcement in Edge and Cyber-Physical Systems . . . . .	17
CHAPTER 3 – CONTRIBUTIONS	19
3.1 Objective- and Utility-Based Negotiation for Access Control . . . . .	19
3.2 Formal Verification for Preventing Misconfigured Access Policies in Kubernetes Clusters . . . . .	19
3.3 Eventually Consistent Access Control with Deterministic Deny-Wins Replay for Multi-Stakeholder Offline Systems . . . . .	20
CHAPTER 4 – IMPLEMENTATION AND CODE ARTIFACTS	21
4.1 AccessControlPolicyNegotiationAlgorithm (Paper A) . . . . .	21
4.2 k8s-abac-verification-demo (Paper B) . . . . .	23
4.3 equack-core (Paper C) . . . . .	25
CHAPTER 5 – CONCLUSION	29
5.1 Discussion on Assumptions and Limitations . . . . .	30
5.2 Future Work . . . . .	31
5.3 Closing Remarks . . . . .	32
REFERENCES	33

<b>Part II</b>	<b>43</b>
<b>PAPER A</b>	<b>45</b>
1 Introduction . . . . .	47
2 Challenges in Access Control . . . . .	48
3 Criteria for Negotiation . . . . .	48
4 Access Control Negotiation Algorithm . . . . .	50
5 Real-World Scenario Evaluation . . . . .	53
6 Algorithm Complexity Analysis . . . . .	57
7 Related Work . . . . .	58
8 Conclusion . . . . .	59
<b>PAPER B</b>	<b>63</b>
1 Introduction . . . . .	65
2 Background and Related Work . . . . .	68
3 Formal Analysis & Policy Verification . . . . .	74
4 Case Study of Real World Kubernetes Security Incidents . . . . .	82
5 Experimental Validation . . . . .	87
6 Discussion . . . . .	90
7 Conclusion . . . . .	91
<b>PAPER C</b>	<b>97</b>
1 Introduction . . . . .	99
2 Problem Analysis . . . . .	101
3 System and Threat Model . . . . .	105
4 EQuack Model . . . . .	106
5 Implementation . . . . .	115
6 Evaluation . . . . .	116
7 Other Related Work . . . . .	122
8 Discussion and Future Work . . . . .	123
9 Conclusion . . . . .	126

---

## ACKNOWLEDGMENTS

---

First and foremost, I owe my deepest gratitude to Professor Ulf Bodin, along with Dr. Eric Chiquito and Dr. Johan Kristiansson, for their guidance and supervision. I would also like to acknowledge Professor Olov Schelén, whose early insights and direction were instrumental in setting my research trajectory.

To my wife, Rupal Sirohi, none of this would have happened without you. You reshaped your plans and priorities to walk beside me, carried an almost ridiculous amount of belief in me while I was too busy second-guessing everything, and somehow kept this whole thing from unraveling, often through nothing but the sheer force of will. You're my anchor, my partner, and the reason any of this holds together.

A huge thank you to Parul Khanna, without you I might never have found myself in this chapter of my life, your belief in me was the push I didn't know I needed.

To my colleagues, Malte Kerl, whose immense technical expertise and rigorous attention to detail made both research and life in Luleå far more navigable, and Christoffer Fink, thank you for your steady support and insight.

My gratitude also extends to my bandmates Joakim Nilsson, William Laumann, Karl Löwenmark and Alex Chiquito. Playing music with you all has been the best form of therapy, creative, cathartic, and significantly cheaper than the licensed kind.

Sonam Sharma, the intellectual sparring partner I never quite deserved. For over two decades, you've continually challenged how I think, question, and build, shaping not only how I reason and argue but also how I approach the world itself. Your influence on my growth is immeasurable; equal parts mentor and friend, I wouldn't have had it any other way.

Yajat Bakshi, thank you for quite literally growing up with me and being there through every phase of life. Few friendships survive time and change the way ours has, and I'm grateful for every bit of it.

Somesh Tiwari, who immediately set the bar for what genuine friendship looks like, thank you for your loyalty and that inconvenient habit of making everyone around you better, I am unsurprisingly, no exception.

Finally, to my parents Vijay & Ekta Sissodiya, thank you for your endless love and patience. Your support has been the quiet, steady foundation beneath everything I've done in my life, including this academic journey.

Aditya Sissodiya,  
Luleå, January 2026.



# Part I



---

# CHAPTER 1

---

## Introduction

*“You can change friends but not neighbours.”*

*Atal Bihari Vajpayee*

Many systems we build and operate today no longer sit under a single administrative roof. Cloud services, data spaces, and edge or cyber-physical deployments routinely span multiple organizations, each with its own operators, policies, and incentives. In that kind of environment, access to shared resources is the outcome of several stakeholders’ rules interacting rather than a single authority’s decision. The upside is obvious, data can move and services can be composed across boundaries, but it also makes access control harder to get right, because the rules have to hold under distribution, local autonomy, and mismatched governance expectations.

Traditional access control models assumed a single authority or a homogeneous environment, but in today’s multi-actor systems the governance and enforcement of security rules are spread across trust boundaries. Each participant may enforce its own rules and assumptions, and these can vary widely or even conflict with each other. In such an environment, the central problem is not defining a single global policy, but ensuring that heterogeneous, independently chosen access rules interact in a way that is coherent and predictable. While existing access control mechanisms address expression or enforcement within single domains, they provide limited support for jointly reconciling and enforcing rules across organizational boundaries, motivating the mechanisms developed in this thesis. This challenge is compounded when stakeholders have conflicting interests or objectives, since no single party can unilaterally dictate the security rules for everyone else.

Before proceeding to the formal problem statement and research questions, it is useful to clarify the setting this thesis operates in. We are concerned with *multi-stakeholder data ecosystems*: environments where multiple independent parties, each with their own governance, policies, and infrastructure must collaborate around shared or co-owned resources. Examples include cross-organizational data spaces (such as those envisioned by initiatives like Gaia-X or the International Data Spaces), multi-tenant cloud platforms, and cyber-physical systems at the edge where devices from different owners interact. In such settings, the term *decentralized* refers to the absence of a single central authority that governs all access decisions; *distributed* describes the physical and logical reality that system components (services, data stores, policy engines) run across multiple nodes and locations; *federated* captures the cooperative aspect: independent domains agree to interoperate under shared

rules while retaining local autonomy. Finally, *co-owned resources* are assets (data sets, services, devices) over which multiple stakeholders hold legitimate interests and must jointly determine access. These concepts will be defined more precisely in Chapter 2, but an intuitive grasp of them is sufficient for understanding the research questions that follow.

## 1.1 Problem Formulation and Research Questions

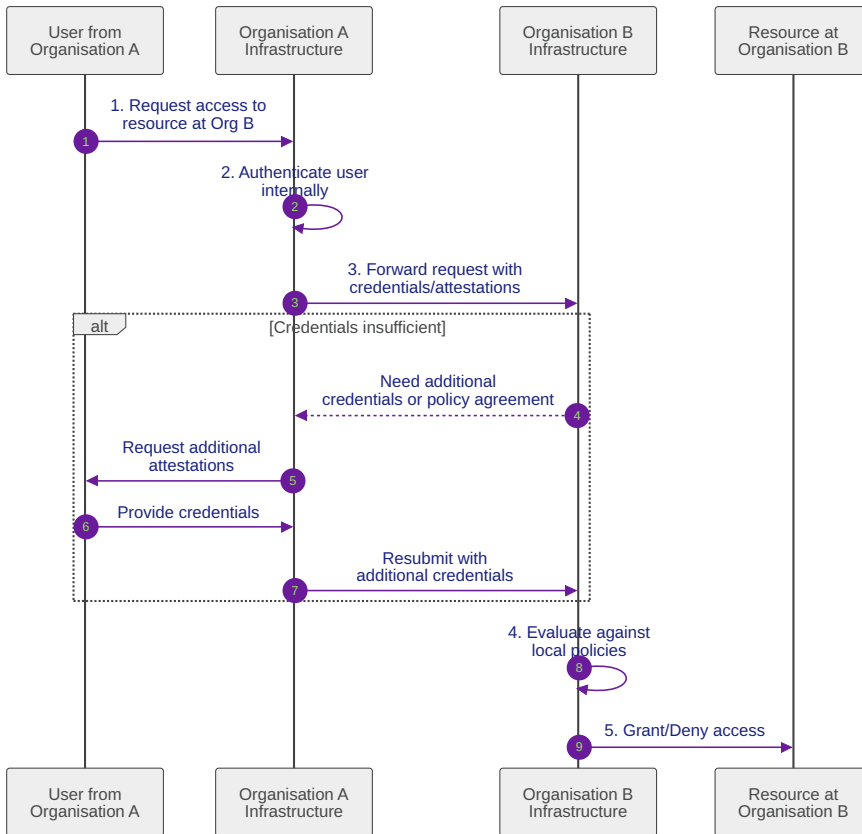


Figure 1.1: Cross-organisation scenario.

Consider a scenario (as shown in Figure 1.1) where a user from Organisation A wants to access a resource at Organisation B. Organisation A might accept B's authentication of its own users but distrust B's judgement on, for example, financial transactions. Organisation B, in turn, may require that any external user (like one from A) present specific credentials or agree to usage policies before

access is granted. Neither side fully trusts the other, neither controls the other's infrastructure, yet they still need to interoperate.

In such settings, two technical problems appear together. First, *how* should access rules be agreed when stakeholders have different and sometimes conflicting priorities (security, utility, privacy, cost)? Relying on informal, case-by-case negotiation easily leads to weakest-link compromises or stalemates. A more principled approach treats access-rule selection as a negotiation problem; stakeholders express preferences, candidate rule sets are compared against those preferences, and a mechanism selects rules that are acceptable to all. Second, once some access rules have been agreed in principle, *how* can we ensure that they are enforced correctly across heterogeneous systems, possibly spanning cloud platforms and intermittently connected edge devices? Here, misconfigurations, semantic mismatches between policy intent and platform configuration, and offline operation can all undermine the agreement.

### 1.1.1 Research Questions

We focus on two main research questions (RQs) that drive our investigation:

**RQ-A:** *How can mutually acceptable access-control agreements be systematically established at the level of joint policy selection for co-owned resources in decentralised systems where governance, policy, and enforcement are distributed across independent actors with conflicting interests?*

This question addresses the fundamental tension between stakeholder autonomy and the need for cooperation. RQ-A is addressed primarily through the utility-based negotiation framework developed in Paper A, which focuses on the problem of agreeing on access rules between independent organisations. It is also informed by the offline enforcement model in Paper C, which must maintain trust guarantees even when parties cannot communicate in real time.

**RQ-B:** *What design principles enable security and consistency in federated access control without sacrificing stakeholder autonomy or system adaptability?*

Once access rules are agreed upon, how do we ensure they are actually upheld across heterogeneous, independently administered systems? Here, the tension is between strong security guarantees (which often rely on centralised control or strict coordination) and the autonomy that stakeholders demand (which resists centralisation). RQ-B is addressed through the formal verification pipeline for Kubernetes access control developed in Paper B, which targets correctness of enforcement on a cloud-native platform. It is further addressed by the EQuack<sup>1</sup> model in Paper C, which provides deterministic, deny-wins enforcement semantics for offline-capable nodes.

Together, these research questions shape the work of this thesis. RQ-A directs us towards mechanisms for establishing and negotiating trust among autonomous parties, while RQ-B guides the design of enforcement architectures that deliver security and consistency without requiring a central authority.

---

<sup>1</sup>EQuack is an intentionally unoriginal play on the acronym ECAC (Eventually Consistent Access Control) and refers to the same class of access-control problem.

The contributions build on prior work in access control, trust negotiation, formal methods, and distributed systems, as detailed in Chapter 2. The research problems were identified collaboratively through discussions with supervisors and exposure to challenges in industrial and academic projects.

### 1.1.2 Assumptions

The contributions in this thesis are not designed for a worst-case adversarial universe. They make a number of simplifying assumptions that reflect the kinds of systems and failures we actually set out to study. When these assumptions are violated, the guarantees claimed for the corresponding contribution no longer hold.

- *Administrators are fallible, not fully malicious.* The work in Papers A and B assumes that system administrators and policy authors are trying to configure access control sensibly, but may misunderstand platform semantics or overlook corner cases. The primary threat is misconfiguration, not an insider with root access who actively tries to bypass every control. If an operator can arbitrarily modify code, kernels, or hardware, none of the mechanisms in this thesis will save the system.
- *Information eventually flows between nodes.* The EQuack model in Paper C assumes that nodes which are temporarily disconnected will eventually be able to exchange operations (access rules, credentials, audit entries) again. Convergence and audit guarantees are proved under this eventual-delivery assumption. Permanent partitions or devices that never rejoin are out of scope.
- *Cryptography behaves as advertised.* The EQuack model relies on standard cryptographic primitives (digital signatures, hash functions) to provide integrity and authenticity for operations and audit logs. We assume these primitives are implemented correctly and are not broken by practical cryptanalytic advances during the system's lifetime.
- *Preferences can be stated explicitly.* The negotiation framework in Paper A assumes that stakeholders are able (or can be supported) to express their priorities over criteria such as security, utility, and privacy in a way that can be fed to a multi-criteria optimiser. Likewise, the verification pipeline in Paper B assumes that the security properties of interest are available as explicit invariants. In both cases, this thesis does *not* tackle the upstream problem of eliciting preferences or discovering good invariants; it takes them as given and asks: *if* we have such inputs, what can we automate?

Chapter 5 returns to these assumptions and discusses which are fundamental (for example, the impossibility of defending against fully compromised hardware) and which are methodological shortcuts (for example, taking security invariants as given) that future work ought to address.

Throughout, I am explicit about the boundary between existing knowledge and contributions. Access control models (RBAC, ABAC, usage control), trust-negotiation concepts, multi-criteria decision theory, formal verification techniques (SMT solving, model checking), distributed consistency models (CRDTs, eventual consistency), and cryptographic primitives (digital signatures, hash chains) are taken from the literature.

The specific combination of these building blocks to address the identified challenges, the formal models tailored to each problem domain, the algorithms and optimisation procedures, the tool implementations, and the empirical evaluations demonstrating that the solutions work in realistic scenarios are the contributions.

## 1.2 Research Methodology

This section describes how I approach research problems in this thesis. It is not meant as a general prescription for how systems security research should be done, but as a personal account of the process I followed and, in hindsight, how it actually played out.

1. *Identify a problem.* In practice, this step is never as clean as the papers suggest. The initial “problem” is often vague, which only later solidifies into a question I can write down.
2. *Design a solution.* Once the problem is clear enough to articulate, I design a candidate solution by combining building blocks from the literature. The design phase is deductive in intent; I ask what properties the solution should have and derive a structure that could deliver them. In reality, this is where biases and shortcuts creep in. For example, in Paper A and Paper B I sometimes let available techniques (multi-criteria optimisation, SMT solving) shape the formulation of the problem more than I would now like to admit.
3. *Build a prototype and testbed.* After developing the design, I implement it and build targeted scenarios to exercise its boundary conditions. These artefacts are prototypes rather than production systems, and their primary role is to generate evidence and expose weaknesses. This phase also makes implicit assumptions visible. For example, in Paper B the dependence on a hand-picked security invariant became difficult to ignore once the tool could convincingly “verify” any invariant I chose to encode.
4. *Evaluate and iterate.* Finally, I evaluate the prototype against the original problem. Does it actually prevent the misconfigurations or deadlocks I started from? Where does it fall short? Evaluation results feed back into the design, and the cycle repeats until the solution is robust enough to report. Sometimes the iteration reveals that the original framing was optimistic; in some cases, there is no satisfying solution under the assumptions I initially made, or the cost of enforcing the desired property is higher than I had expected.

Earlier work in the thesis (particularly the choice of criteria in Paper A and the ad hoc security invariant in Paper B) shows more “solution-driven” thinking than I would now consider ideal. The latter work in Paper C is more explicit about its assumptions and less willing to let tools dictate the shape of the problem.

In that sense, the methodology described above is both a description and an aspiration. It captures the structure I tried to follow, and the loop I would follow more strictly in future work, having seen where taking convenient assumptions “out of thin air” leads and how much effort it takes to repair that later.

### 1.3 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces the technical background and related work in access control, negotiation, verification, and distributed enforcement that frame the research questions. Chapter 3 makes the contributions explicit, stating each claim and which research questions it addresses. Chapter 4 presents the three software artefacts associated to each contribution, focusing on their architecture and implementation. Chapter 5 revisits the research questions, explicitly answering RQ-A and RQ-B, discusses limitations and assumptions, and outlines directions for future work.

Readers may wish to read Chapters 1 and 2 sequentially, to first understand the problem setting and then the technical background. Chapter 3 provides a compact overview of what the thesis claims to contribute and can be read before or after the papers themselves. Chapter 4 can be read selectively; each of its three artefact descriptions (negotiation framework, Kubernetes verification pipeline, EQuack prototype) stands alone and may be skimmed or studied in detail depending on interest. Chapter 5 is best read last, as it explicitly answers the research questions, discusses limitations, and sketches directions for future work.

---

## CHAPTER 2

---

# Research Background

*“The nice thing about standards is that there are so many to choose from.”<sup>1</sup>*

*Andrew S. Tanenbaum*

This chapter sets the thesis in context. It clarifies how this work uses the terms distributed, decentralised, and federated, and what that means for multi-stakeholder data ecosystems and co-owned resources. It then reviews access control models and why they tend to assume a single policy owner, before covering trust negotiation, policy combination, and multi-criteria decision making as background for agreeing on rules. On the enforcement side, it surveys PDP/PEP designs, Kubernetes access control, and formal verification to motivate the pipeline in Paper B, and it reviews offline/edge enforcement, eventual consistency, and deterministic replay as context for EQuack in Paper C.

### 2.1 Distributed, Decentralised and Federated Systems

The terms *distributed*, *decentralised*, and *federated* are often used interchangeably, but they point to different dimensions of a system [1, 2]. A system is *distributed* when its components (processes, services, data stores) are spread across multiple networked nodes and coordinate by passing messages [1]. Distribution is a physical and logical property; state and computation are not confined to a single machine. A cloud provider’s internal infrastructure, a microservice architecture, or a replicated database are all distributed in this sense, even if they are entirely owned and controlled by one organisation.

A system is *decentralised* when decision-making authority is not concentrated in a single entity [3]. Multiple parties can make independent decisions about configuration, policy, or operation, and no single party can unilaterally impose rules on everyone else. Decentralisation is an organisational property; a system can be physically distributed but centrally governed, or conversely centrally specified but physically replicated [4]. In this thesis, “decentralised” refers mainly to the governance of access control. Each stakeholder runs its own infrastructure, sets its own local access policies, and is not compelled to accept others’ access policies wholesale.

A system is *federated* when independent domains agree to interoperate under shared rules while

---

<sup>1</sup> “The nice thing about some quotes is that there are so many authors you can attribute them to.” *Malte Kerl*

retaining local autonomy [5]. Federation is a cooperative arrangement; participants voluntarily join and follow common protocols (for identity, policy exchange, auditing, and so on), but each domain remains self-governing internally [6]. Federation typically implies some degree of distribution and some degree of decentralisation, but emphasises the agreement that enables collaboration. Figure 2.1 sketches these three notions as axes rather than categories; a given ecosystem can be more or less distributed, more or less decentralised, and more or less strongly federated, and where it lands in this space strongly affects what can be expected from its access control.

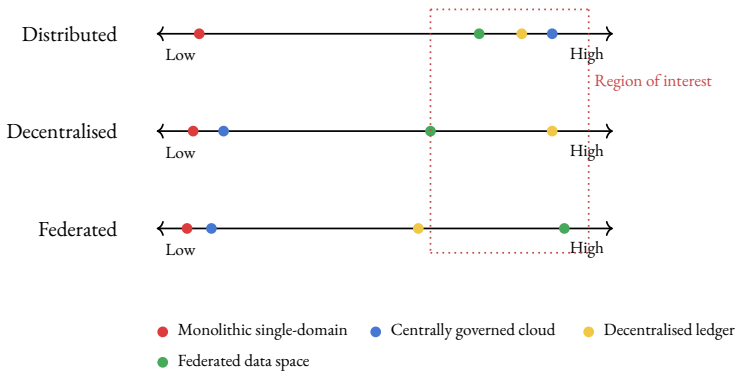


Figure 2.1: Illustrative placement of system types along three axes; degree of distribution, decentralisation, and federation used in this thesis. The markers are not intended as a taxonomy or a precise measurement; they indicate relative position, and the dashed box highlights the region of interest addressed by the thesis.

The systems of interest in this thesis are distributed (components run on multiple nodes), decentralised (no single authority controls all access), and often federated (domains agree to share data or services under negotiated rules). The difficulty is not defining access control in a single domain; it is making coherent sense of access control when all three of these dimensions interact [7].

## 2.2 Multi-Stakeholder Data Ecosystems

A *multi-stakeholder data ecosystem* is an environment in which multiple independent parties collaborate around shared or interdependent data assets [8]. Each party brings its own access policies, legal obligations, and risk tolerance; the ecosystem only works if data flows between them in a controlled way [9].

In cross-organisational data spaces (such as Gaia-X<sup>2</sup> or the International Data Spaces Association<sup>3</sup>), data providers, consumers, and intermediaries participate under common rules for data usage and sovereignty [10]. In research collaborations, multiple institutions pool patient records, sensor

<sup>2</sup><https://gaia-x.eu/>

<sup>3</sup><https://internationaldataspaces.org/>

data, or other sensitive datasets under data-sharing agreements, with each institution still responsible for its own subjects and infrastructure [11, 12]. In industrial supply chains, manufacturers, suppliers, logistics providers, and service companies exchange operational and business data, each under its own regulatory and commercial constraints [13]. Across all these cases, there is no single party that can impose its preferred access policy on everyone else [14]. Rules about who can see or change what are the outcome of agreements between stakeholders, not of a single administrator writing a policy file [15]. This is where the negotiation aspects of RQ-A become concrete; we are not tuning one organisation’s policy; we are trying to get several organisations to settle on acceptable rules for co-owned data [16].

## 2.3 Co-Owned Resources

A *co-owned resource* is any asset where more than one stakeholder has a legitimate say in how it is accessed or used [17]. Co-ownership can arise through joint creation (a dataset, model, or service built collaboratively), through stacked interests (a cloud provider controlling infrastructure, a tenant controlling the application, and end users having rights over their personal data), or through regulatory overlap (multiple legal regimes imposing requirements on the same data) [18].

Figure 2.2 sketches a simple layered view; infrastructure at the bottom, applications in the middle, data subjects at the top. Different stakeholders exert control at different layers, but access decisions often need to respect all those layers at once [19, 20].

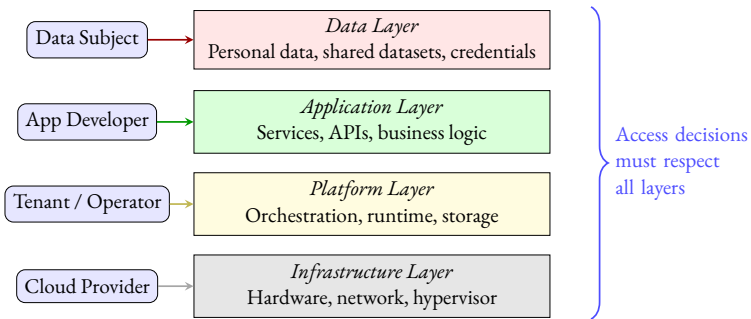


Figure 2.2: Illustrative stacked ownership of a co-owned resource. Different stakeholders control different layers, but access decisions must respect constraints from all layers simultaneously.

For co-owned resources, unilateral access decisions are rarely legitimate [21]. Granting or denying access may require respecting multiple policy viewpoints at once [22]. This is the type of resource all three contributions target, in different operational environments; data-sharing agreements in Paper A, cluster configurations in Paper B, and offline edge devices in Paper C.

## 2.4 Access Control Models in Multi-Stakeholder Settings

Classical access control models were developed for single organisations with central policy authority [23, 24]. It is worth recalling them, if only to see where they start to creak when pulled into multi-stakeholder, decentralised contexts.

### 2.4.1 Discretionary and Mandatory Access Control

*Discretionary Access Control* (DAC) lets resource “owners” decide who can access their resources, typically via access control lists [25, 26]. This is flexible and intuitive, if you own a file, you grant or revoke access. The downside is that global guarantees are weak; once a subject has access, they can often propagate it further, intentionally or not, and there is no single view of the system’s security posture [27]. *Mandatory Access Control* (MAC) flips the picture. System-wide policies, not individual owners, determine who can access what [28, 29]. Subjects and objects are labelled (for example with security classifications), and fixed rules such as Bell–LaPadula for confidentiality or Biba for integrity are enforced uniformly [30, 31]. MAC can provide strong guarantees in high-assurance systems, but it assumes that one authority gets to decide the labelling and rules. In multi-stakeholder ecosystems, both extremes are problematic. DAC aligns with local autonomy but can easily undermine joint agreements; one party can accidentally open up a shared resource beyond what others consider acceptable. MAC provides strong control but presupposes a single security policy and authority, which is exactly what we do not have [32, 33].

### 2.4.2 RBAC, ABAC, and Usage Control

*Role-Based Access Control* (RBAC) assigns permissions to roles, and users to those roles [34]. It is attractive in organisations with relatively stable job functions and is built into many platforms, including Kubernetes [35]. Administrators manage permissions at the level of roles rather than individual users, which scales better. However, RBAC shows its limits when contextual factors (time, location, tenant) matter, when roles need to cross organisational boundaries, or when permissions depend on dynamic properties of the data or environment [36]. Cross-domain RBAC requires manual role mapping or global agreement on role semantics, both of which are brittle in loosely coupled federations [37, 38]. *Attribute-Based Access Control* (ABAC) bases decisions on attributes of subjects, resources, actions, and the environment [39]. In principle this provides more expressiveness and is better suited to dynamic environments than RBAC [40]. In practice it shifts the problem; someone must decide which attributes to trust, which attribute authorities to accept, and how to align attribute vocabularies across domains [41]. As policies grow, the combined behaviour can be hard to predict even for experts [42, 43]. *Usage control and obligations* models (such as  $U_{CON}_{ABC}$ ) extend ABAC by adding obligations and continuous enforcement; not just “may read” but “may read only if condition C holds, and must later do action D” [44]. This looks much closer to the kinds of clauses found in data-sharing agreements (no onward disclosure, delete after date  $X$ , log all access events, and so on) [45, 46]. Enforcing such obligations across organisational boundaries, however, is still largely unsolved; once data leaves one domain, the original controller has limited visibility into what happens [47]. RBAC, ABAC, and usage control provide the vocabulary to express access policies, but they say nothing about how multiple stakeholders should arrive at a policy they can all live

with [21, 48]. They also do not help much with checking that a complex stack like Kubernetes actually enforces what the policy intends [49, 50]. These are precisely the issues that RQ-A and RQ-B will target; Paper A uses a multi-criteria view of “good policy” to support agreement over a fixed catalogue of rules; Paper B tackles the mismatch between high-level intent and low-level configuration on a concrete platform.

## 2.5 Policy Negotiation and Decision Support

Classical access control assumes access policies exist and focuses on enforcing them [23]. In multi-stakeholder ecosystems, the question of *what the policy should be* is itself a technical and socio-technical problem [21]. This section looks at three strands of work that relate to RQ-A and Paper A; trust negotiation, policy conflict resolution, and multi-criteria decision making.

### 2.5.1 Trust Negotiation

Trust negotiation grew out of the need for strangers in open systems to establish enough trust to interact securely [51, 52]. Instead of hard-coded lists of trusted principals, parties use credentials and policies to make decisions on the fly [53]. A typical picture (illustrated in Figure 2.3) is two parties revealing credentials and partial policies in steps, each disclosure driven by a local strategy [54, 55].

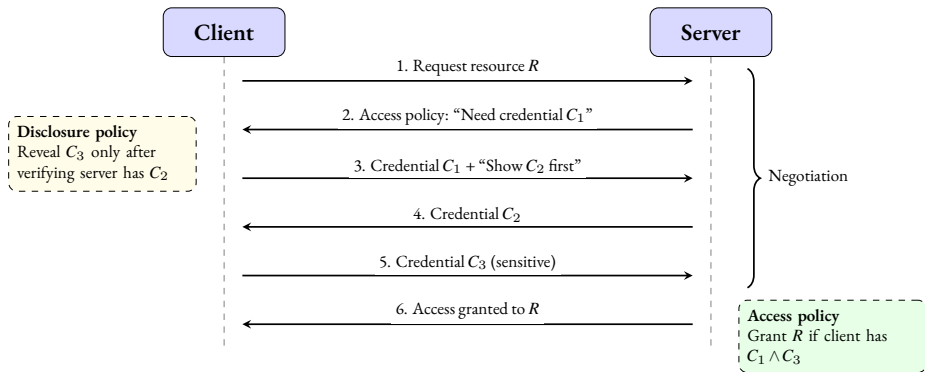


Figure 2.3: Illustrative trust-negotiation interaction. Each party reveals credentials incrementally, governed by local disclosure policies, until enough trust is established to grant access.

The main building blocks are, digitally signed credentials attesting to attributes (membership, roles, certifications) [56, 57], access policies defining which credentials are required for a given resource [32], and disclosure policies constraining when a party is willing to reveal particular credentials or parts of its policy set [52, 58].

Negotiation proceeds as a sequence of requests and disclosures, each party revealing just enough to make progress without oversharing [55, 59]. This work is useful for determining *whether* access

should be granted, given the existing policies and available credentials, but it largely assumes that those policies are already there [60]. It does not tackle the question of how two or more organisations should co-design the underlying rule set that governs a shared resource.

### 2.5.2 Policy Combination

When multiple policies apply to the same request, their decisions must be combined [61, 62]. Policy languages such as XACML make combination strategies explicit; deny-overrides, permit-overrides, first-applicable, or priority-based schemes [63]. These mechanisms matter in federated settings, where different stakeholders' policies may interact [64].

However, policy combination is the last step in the pipeline; it assumes each stakeholder has independently authored a policy and now the system must decide how to reconcile them at run time [65]. It does not help stakeholders understand whether the resulting overall behaviour matches anybody's intent, or whether there might have been a better set of rules to begin with [42].

### 2.5.3 Multi-Criteria Decision Making

Multi-criteria decision making (MCDM) provides a toolbox for choosing between alternatives when multiple, often conflicting, criteria matter [66]. Techniques such as weighted-sum scoring, Pareto optimisation, or Analytic Hierarchy Process are standard in operations research [67]. Choosing access rules for co-owned resources is, fundamentally, a multi-criteria problem; security, utility, privacy, cost, and availability all matter, and different stakeholders weight them differently [68]. While multi-criteria decision making has been applied in security and access control evaluation, it has largely been used for analysing or tuning existing policies rather than for jointly selecting policies among multiple autonomous stakeholders [69], which is the focus of this thesis. Policy design in practice is still mostly manual, informal, and sensitive to whoever is loudest in the room.

Figure 2.4 sketches the connection; stakeholders specify criteria and weights, candidate policies are scored, and a frontier of Pareto-efficient options emerges [70]. Paper A turns this conceptual picture into a concrete negotiation support tool.

Trust negotiation and policy combination answer "Given everyone's access policies, can we grant this request?". Multi-criteria decision making answers "Given alternative options and multiple criteria, which option should we pick?".

## 2.6 Enforcement and Formal Verification in Cloud-Native Systems

RQ-B asks how to keep enforcement secure and consistent without centralising everything. This section zooms in on cloud-native systems, particularly Kubernetes, where the verification pipeline of Paper B lives.

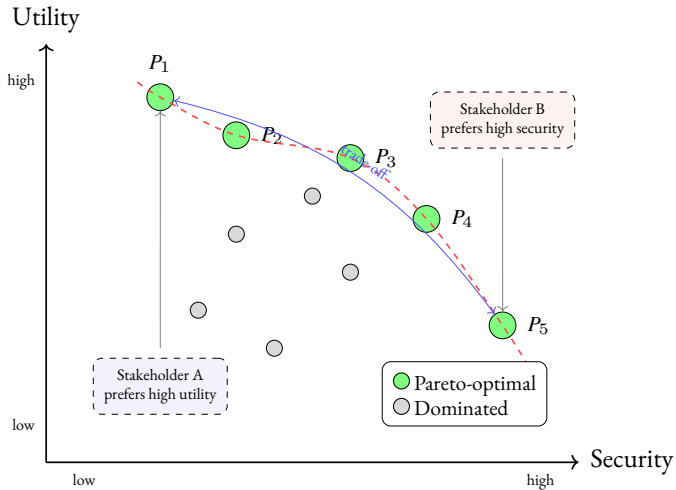


Figure 2.4: Access-rule selection as a multi-criteria decision problem. Candidate policies are plotted by their scores on two criteria; the Pareto frontier (dashed line) contains non-dominated options. Different stakeholders may prefer different points on the frontier.

### 2.6.1 PDPs, PEPs, and Policy Distribution

The classic reference model for access control separates the *Policy Decision Point* (PDP), which evaluates requests against access policies, from the *Policy Enforcement Point* (PEP), which intercepts requests and enforces decisions [7]. In a monolithic system both may live in the same process; in larger systems, PDPs may be centralised services, replicated components, or embedded into each application [71].

In multi-stakeholder or multi-tenant settings, each participant may run its own PDP/PEP stack [72, 73]. Centralising all decisions would make some things easier (single place to verify), but is often politically or operationally unacceptable [74]. The net result is that actual enforcement behaviour is the emergent product of several stacks, each with its own configuration and failure modes [75, 76].

### 2.6.2 Access Control in Kubernetes

Kubernetes is now the default platform for running containerised workloads, and its security story is a good example of why RQ-B is non-trivial [77, 78].

At a high level, Kubernetes enforces access control through a sequence of steps:

1. *Authentication* of API clients (via certificates, bearer tokens, or OIDC) [79].
2. *Authorisation* via RBAC: Roles and ClusterRoles define sets of allowed actions on resources; RoleBindings and ClusterRoleBindings attach those roles to users, groups, or service accounts [35].

3. *Admission control*, implemented through built-in controllers and user-defined webhooks, which can reject or mutate API requests based on additional policies [80, 81].
4. *Network policies* that control traffic between pods and external endpoints at the network layer [82, 83].

A typical request path is shown in Figure 2.5; a client authenticates to the API server, RBAC decides whether the action is allowed in principle, admission controllers enforce additional constraints, and network policies determine the actual data flows once the object exists.

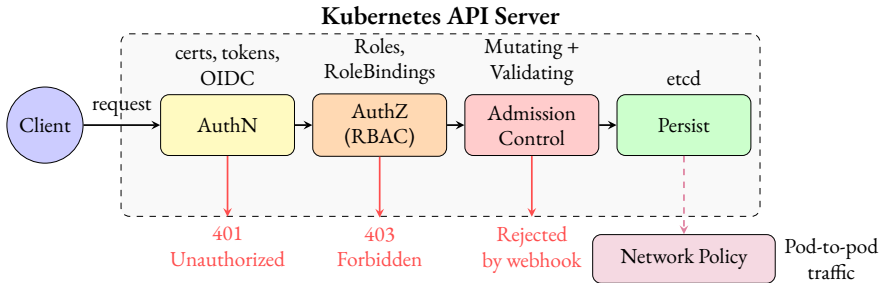


Figure 2.5: Simplified view of access control in Kubernetes. A request passes through authentication, RBAC authorisation, and admission control before being persisted. Each stage can reject the request for different reasons. Network policies additionally constrain runtime traffic.

In practice, these layers interact in messy ways [49, 50]. It is easy to construct configurations where a RoleBinding grants far more permissions than intended, where a user can create or modify RoleBindings and thus escalate privileges [84], where an admission webhook is misconfigured or missing, leaving a policy unenforced [85], or where cross-namespace access is unintentionally allowed [86]. Real incidents have shown that such misconfigurations are both common and hard to spot by inspection [87]. Tools exist to lint manifests or enforce best practices [88], but they rarely provide a formal statement of what is guaranteed *not* to happen.

### 2.6.3 Formal Verification for Access Policies

Formal verification offers a way to link configuration to intent more rigorously [89]. The basic pattern is to model the behaviour of the system (or a relevant subset) in a formal language, express desired properties as logical formulas (invariants), and then use model checking, SAT/SMT solving, or related techniques to see if those properties hold or to find counterexamples [90, 91].

For access control in Kubernetes, this means modelling what requests can be made in a given configuration, what the API server and RBAC logic will decide, and what the admission controllers will accept [92, 93]. If an SMT solver can find a model where a "non-admin" user ends up with cluster-admin powers, then there is a real misconfiguration, not just a theoretical concern [84].

There is a body of work on verifying abstract access control models [42, 43] and on verifying specific security properties of configuration languages [94]. There is less on whole-platform, configuration-driven pipelines for real systems like Kubernetes [93, 95]. Much current practice relies on conventions and testing rather than proofs [87].

The verification pipeline in Paper B is built exactly in this gap between intent and configuration. It targets RQ-B in a specific, concrete setting; given a Kubernetes cluster configuration and a set of security properties (in the form of invariants), it uses SMT solving to detect real misconfigurations. The work takes the invariants as input rather than trying to derive them; the contribution is showing that, once those desired properties are written down, we can check them mechanically for a non-trivial, widely used platform.

## 2.7 Offline Enforcement in Edge and Cyber-Physical Systems

The third context in which RQ-A and RQ-B play out is at the edge; devices and subsystems that are only intermittently connected, yet still must enforce rules issued by multiple authorities.

### 2.7.1 Offline Enforcement Challenges

Edge and cyber-physical systems bring a slightly different set of headaches [96]. Devices cannot always reach a central PDP, so they must make decisions locally [97]. Policies change over time, often while devices are offline; updates need to be propagated and reconciled later [98]. Several authorities (device owner, fleet operator, regulator) may issue rules, and their rules may interact or conflict [99, 100]. For accountability and debugging, it is useful to reconstruct *after the fact* why a device allowed or denied particular actions while it was offline [101, 102].

Most existing access control work either assumes continuous connectivity to some trusted decision service [7], or treats offline behaviour as an exceptional mode without strong guarantees [103, 104].

### 2.7.2 Eventual Consistency, CRDTs, and Ordering

Distributed systems have long wrestled with conflicting updates and partitions [105]. Eventual consistency and Conflict-free Replicated Data Types (CRDTs) are two responses; they allow replicas to accept updates independently and guarantee that, once all updates have been exchanged, the replicas converge to the same state [106].

CRDTs are attractive for offline-capable systems [107, 108], but applying them naively to access control is dangerous [109]. A “last writer wins” policy for permissions, for example, is not obviously what stakeholders want [110]. Deny rules, revocations, and time-bounded permissions make the merge semantics much more delicate [111, 112].

To get useful guarantees, we need a well-defined way to order operations (for example, by hybrid logical clocks and issuers) [113], clear conflict resolution rules (for example, deny-wins over allow) [61, 64], and a mechanism to replay operations deterministically and produce the same decisions everywhere [114]. Figure 2.6 illustrates this replay model; operations arrive in potentially different orders, but once sorted and replayed, all replicas end up in the same logical state.

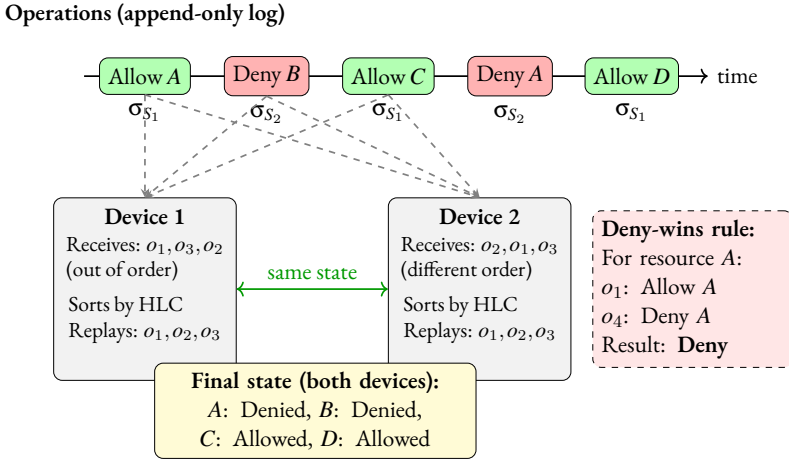


Figure 2.6: Deterministic replay model for offline enforcement. Operations arrive at devices in different orders but are sorted by hybrid logical clock before replay. Deny-wins semantics ensures that conflicting rules resolve conservatively. All devices converge to the same access state.

### 2.7.3 Deny-Wins and Deterministic Replay

A conservative stance in multi-stakeholder access control is that a single stakeholder’s deny should be enough to block access, even if others would allow it [61, 115]. This is the intuition behind deny-overrides semantics [63]. Combining this with deterministic replay over an append-only log of operations gives a promising pattern [114, 116].

Each rule or credential change is an operation appended to a log and signed by its issuer [117, 118], devices maintain local logs and state, accept operations as connectivity allows, and periodically reconcile [119]. Given a set of operations, all devices replay them in the same order and apply the same deny-wins evaluation rules [106], the resulting access state, and the history of decisions, can be reconstructed and audited [101, 114].

This approach fits the “eventual information flow” assumption in Chapter 1, we do not require continuous connectivity, but we do require that updates eventually reach everyone.

# Contributions

*“The purpose of computing is insight, not numbers.”*

*Richard Hamming*

This chapter summarises the contributions of the thesis. Each contribution is positioned against prior work, and linked to the research questions defined in Chapter 1. Table 3.1 provides a compact mapping from contributions to research questions.

### 3.1 Objective- and Utility-Based Negotiation for Access Control

In *Paper A*<sup>1</sup>, we propose a framework that models access-rule selection as a multi-criteria optimisation problem, enabling stakeholders with conflicting priorities to systematically identify mutually acceptable access policies. Prior work on trust negotiation focuses on credential disclosure sequences rather than joint policy design. Policy combination mechanisms assume policies already exist and address runtime conflict resolution. Our framework operates at an earlier stage; it supports the process of *choosing* a policy set by making stakeholder preferences explicit and computing Pareto-efficient candidates. This contributes a documented and reproducible decision-support layer at the stage of joint policy selection, complementing prior work on trust negotiation and policy combination, which primarily address runtime decision-making.

### 3.2 Formal Verification for Preventing Misconfigured Access Policies in Kubernetes Clusters

We develop a two-part assurance mechanism for Kubernetes access control in *Paper B*<sup>2</sup>; an offline, verify-before-deploy pipeline that checks security invariants against RBAC and admission configurations, and a runtime enforcement layer that delegates admission decisions to an external XACML policy decision point. Together, these components can reduce the risk, within the scope of encoded

---

<sup>1</sup>Published in ICISSP, Science and Technology Publications, Lda, 2025

<sup>2</sup>Published in IEEE Access, s. 141798-141813, IEEE, 2025

invariants, that misconfiguration or drift leads to unintended privilege. Unlike existing Kubernetes security tooling, which focuses on pattern-based linting or post-deployment monitoring, this work introduces a configuration-level verification pipeline that checks explicit security invariants against the full authorization semantics of the platform. These approaches can be useful in practice but they do not provide a strong, configuration-level argument that a stated property holds for *all* requests the API server may receive. Our work addresses this gap by demonstrating how selected classes of Kubernetes authorization properties can be translated into solver constraints and mechanically checked. At the same time, the runtime admission integration acknowledges a practical limitation of static verification; not all operational policy intent is conveniently expressed as invariants over configuration alone, and configurations evolve. Delegating admission decisions to a standards-based XACML PDP provides a disciplined enforcement surface that can absorb richer policy logic and act as a conservative backstop when the static model is incomplete or the deployment has drifted.

### 3.3 Eventually Consistent Access Control with Deterministic Deny-Wins Replay for Multi-Stakeholder Offline Systems

In our forthcoming *Paper C*, we design and implement EQuack, an access-control enforcement model for intermittently connected nodes that guarantees convergence to consistent authorisation state through deterministic replay over a signed, append-only operation log with deny-wins semantics. While prior work has explored offline access control and eventual consistency, this work’s novelty lies in combining deterministic replay, deny-wins semantics, and cryptographically verifiable audit into a single enforcement model for multi-authority settings. EQuack combines ideas from CRDTs, hybrid logical clocks, and cryptographic audit into a coherent model that allows nodes to operate independently during disconnection, ensures all nodes converge to the same access decisions once operations are exchanged and produces a tamper-evident audit trail (note that the convergence is ensured only under the assumptions stated in Section 1.1.2). This extends RQ-B into the edge and cyber-physical domain.

Table 3.1: Mapping of contributions to research questions.

Contribution	RQ-A	RQ-B
1. Utility-based negotiation framework	✓	
2. Formal verification + runtime enforcement		✓
3. EQuack	✓	✓

The chapters that follow describe the implementation of these contributions (Chapter 4) and revisit the research questions in light of the results (Chapter 5).

# Implementation and Code Artifacts

*“Talk is cheap. Show me the code.”*

*Linus Torvalds*

## 4.1 AccessControlPolicyNegotiationAlgorithm (Paper A)

This artefact<sup>1</sup> addresses *RQ-A*, how to systematically establish mutually acceptable access-control agreements when stakeholders have conflicting interests. The framework operationalises the idea that access-rule selection can be treated as a multi-criteria optimisation problem, producing documented recommendations rather than ad-hoc compromises. The framework follows a conventional web architecture. Figure 4.1 summarises the main components of the stack: Python 3.8+ with Flask 2 and SQLAlchemy, PostgreSQL with JSONB columns, HTML/CSS/JavaScript for the front-end, and Matplotlib/Seaborn for visualising results.

The client UI allows users to define candidate policies, register stakeholders, assign criterion weights, and trigger the computation of a recommended policy. The Flask layer exposes CRUD endpoints and a dedicated endpoint for running the negotiation algorithm. The database persists all policies, stakeholders, and weight assignments, with JSONB columns used for flexible attribute storage.

The core data model consists of three entities: *Policy*, *Stakeholder*, and *Weight*, implemented as SQLAlchemy models in *app/models.py*. Using JSON/JSONB for both *Policy.details* and *Weight.weights* keeps the schema stable when new criteria are added; the algorithm simply iterates over the JSON keys. The corresponding schema is shown in Figure 4.2.

The core logic, implemented in *app/logic.py*, computes individual utilities, aggregates them, selects a policy, and performs a basic consensus check. For each stakeholder *a* and policy *P<sub>j</sub>*, individual utility is computed as

$$U_a(P_j) = \sum_l w_{a,l} f_l(P_j),$$

where  $w_{a,l}$  is stakeholder *a*'s weight for criterion *l* from *Weight.weights*, and  $f_l(P_j)$  is the score of policy *P<sub>j</sub>* on criterion *l* from *Policy.details*.

---

<sup>1</sup><https://github.com/adityasissodiya/AccessControlPolicyNegotiationAlgorithm>

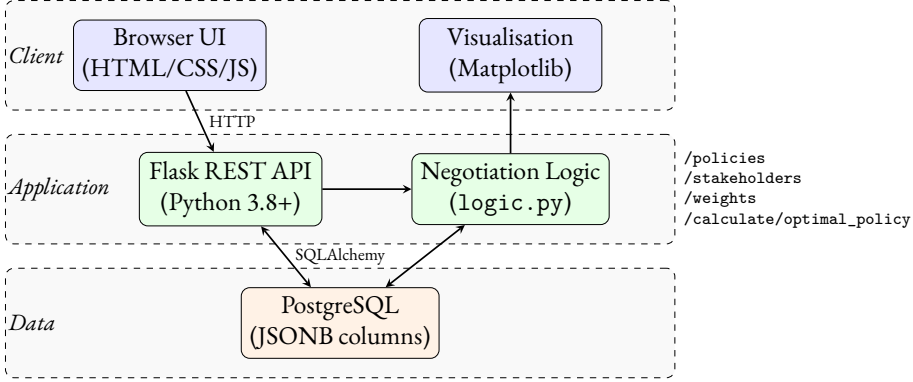


Figure 4.1: High-level architecture of the utility-based negotiation framework: browser UI, Flask REST API, and PostgreSQL storage.

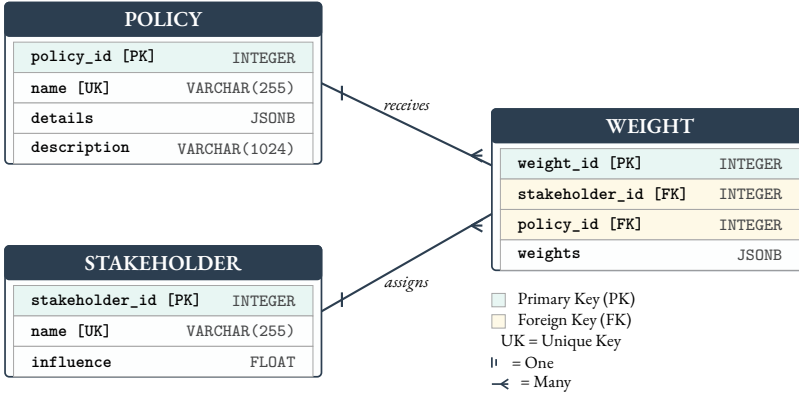


Figure 4.2: Simplified data model of the negotiation framework: policies, stakeholders, and per-stakeholder weight assignments.

Aggregate utility is a weighted sum over stakeholders:

$$U_A(P_j) = \sum_a \alpha_a U_a(P_j),$$

where  $\alpha_a$  is the stakeholder's influence factor. The framework selects a recommended policy  $P^*$  by

$$P^* = \arg \max_j U_A(P_j).$$

A simple consensus predicate then checks whether each stakeholder's utility for  $P^*$  exceeds a configurable threshold  $\theta$ :

$$\forall a : U_a(P^*) \geq \theta.$$

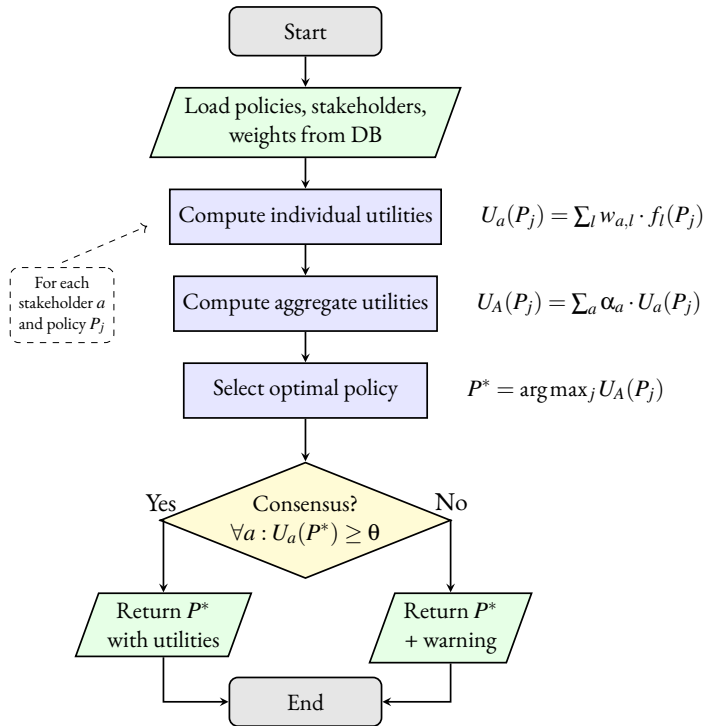


Figure 4.3: Algorithmic flow in the negotiation framework; individual utility computation, aggregation, policy selection, and consensus check.

The `/calculate/optimal_policy` endpoint executes this pipeline and returns the chosen policy together with per-stakeholder utilities, which can be visualised in the UI. The end-to-end algorithmic flow is summarised in Figure 4.3.

## 4.2 k8s-abac-verification-demo (Paper B)

This artefact<sup>2</sup> addresses *RQ-B*, how to achieve secure and consistent enforcement without sacrificing stakeholder autonomy. The testbed demonstrates a verify-before-deploy approach that catches misconfigurations before they reach production, complemented by a runtime enforcement layer that acts as a safety net against configuration drift.

The Kubernetes verification testbed combines an offline CLI verifier with a runtime enforcement module, implemented in the `k8s-abac-verification-demo` repository.

- An *offline verification path*, implemented as a Python CLI, that parses Kubernetes YAML manifests, encodes selected invariants into SMT-LIB2, and calls the Z3 solver.

<sup>2</sup><https://github.com/adityasissodiya/k8s-abac-verification-demo>

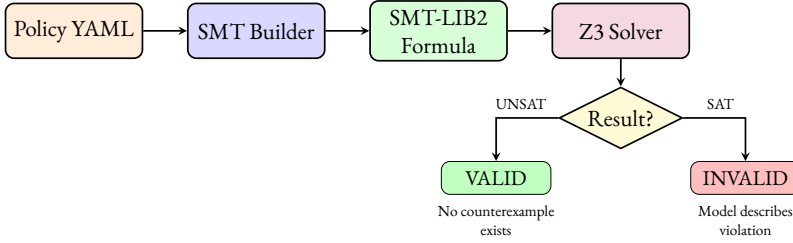


Figure 4.4: Offline SMT-based verification: policy YAML is translated into a constraint system and checked by Z3.

- A *runtime enforcement path* consisting of a Flask-based Validating Admission Webhook and an external XACML PDP (AuthzForce). The webhook turns *AdmissionReview* objects into XACML requests and applies the PDP's *Permit/Deny* decisions.

The implementation uses Z3 (QF\_S) for SMT solving, Kubernetes *ValidatingWebhookConfiguration* for API integration, AuthzForce as an XACML 3.0 PDP, and Python 3.8+ for both the CLI and the webhook.

The CLI verifier translates selected aspects of Kubernetes configuration into SMT-LIB2 formulas. Each invariant is encoded by asserting the negation of the intended property and then asking Z3 whether a counterexample exists. A simplified encoding for a registry check looks as follows:

```

1 (set-logic QF_S)
2 (declare-fun registry () String)
3 (assert (= registry "myregistry.com.attacker.com"))
4 ; ... constraints extracted from policy ...
5 (assert (not allowed))
6 (check-sat)
7 (get-model)
  
```

Specialised builder functions handle different invariant classes, for example:

```

1 build_smt_for_registry(policy_yaml)
2 build_smt_for_wildcard(policy_yaml)
3 build_smt_for_tenant(policy_yaml)
  
```

The CLI wrapper interprets solver output as follows:

- UNSAT  $\Rightarrow$  *VALID* (no counterexample for the encoded violation).
- SAT  $\Rightarrow$  *INVALID* (model describes a violating configuration).

Figure 4.4 summarises the offline SMT-based verification pipeline.

The runtime component reuses the same invariant ideas as XACML policies, for example: *image-registry-policy.xml* registry restrictions for container images. *wildcard-rolebinding-policy.xml* constraints on wildcard subject/resource patterns. *tenant-isolation-policy.xml* namespace isolation constraints.

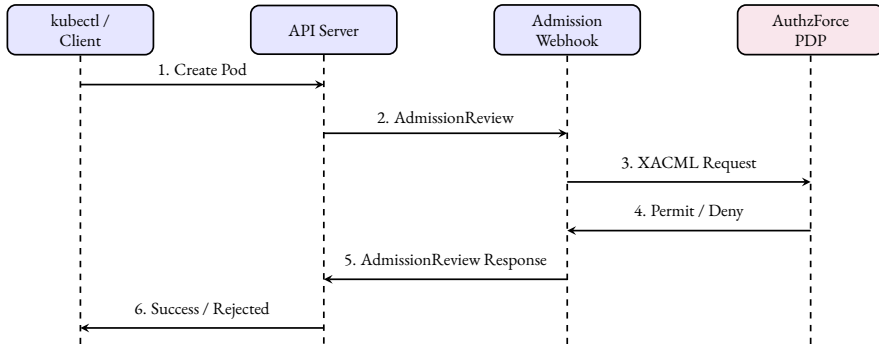


Figure 4.5: Runtime enforcement path: Kubernetes admission webhook delegates authorisation decisions to an XACML PDP.

Figure 4.5 shows the end-to-end runtime enforcement path.

Example Kubernetes manifests are stored under *cli/fixtures/*. For each invariant class, the set includes both “bad” and “fixed” configurations. Running the CLI over the fixtures produces a small but concrete regression suite in which each manifest is labelled as VALID or INVALID for the invariants considered.

### 4.3 equack-core (Paper C)

The artefact is implemented as `equack-core`<sup>3</sup>, a Rust (edition 2021) engine that represents every state change as an operation and derives all materialised state by replay. Figure 4.6 summarises the execution path from ingestion to query, and Figure 4.7 shows the supporting storage, networking, and cryptographic components.

Operations are received over `libp2p` and written to local storage as they arrive (INGEST). The storage layer is built on RocksDB and separates the append-only operation log, materialised state, audit records, and checkpoints into distinct column families (Figure 4.7). This keeps the append path independent of replay and query, and enables checkpoints without coupling the engine to a specific on-disk layout.

Each operation is validated before it is admitted to replay (VALIDATE). Validation verifies the author signature over a canonicalised header and checks credential status against the locally maintained trust view. The implementation uses Ed25519 for signatures, BLAKE3 for operation identifiers and digests, and SHA-256 for status-list chunk verification (Figure 4.7). Performing validation at ingest time allows replay to treat the operation set as well-formed input and remain deterministic.

The log is represented as a directed acyclic graph of operations (EVENT DAG). Operation headers carry causal parent links and an HLC timestamp; the engine accepts out-of-order arrivals by staging operations with missing parents and activating them once dependencies are present. Replay deterministically traverses the activated DAG (REPLAY). The engine computes a topological order and

<sup>3</sup><https://github.com/adityasissodiya/equack-core>

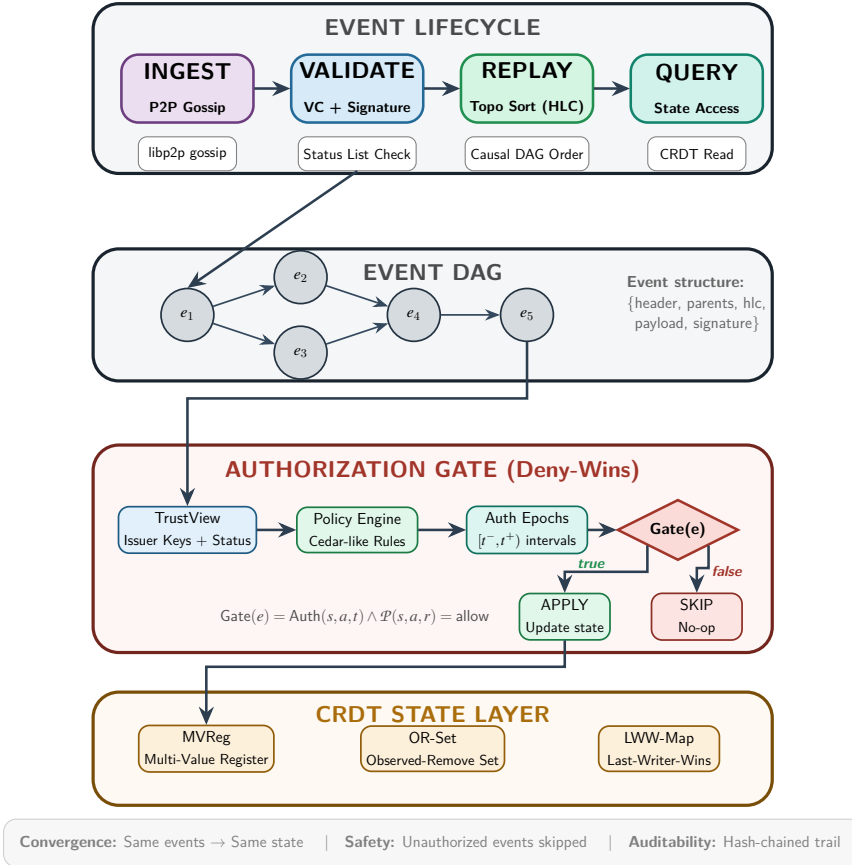


Figure 4.6: EQuack core data-flow. Operations pass through four phases: **INGEST** receives operations via peer-to-peer gossip, **VALIDATE** checks cryptographic signatures and credential status, **REPLAY** deterministically orders and applies operations using a topological traversal of the causal DAG with Hybrid Logical Clocks (HLC), and **QUERY** exposes the materialised state. The replay loop routes each operation through an authorisation gate (TrustView, Policy Engine, and authorisation epochs over  $[t^-, t^+]$  intervals) and either applies the corresponding state update or records the operation as skipped. State is materialised using CRDTs (MVReg, OR-Set, and LWW-Map), and decisions are recorded in a hash-chained audit trail.

resolves concurrency using the HLC tuple with a stable identifier tie-break so that replicas with the same operation set compute the same replay order regardless of arrival patterns.

Each operation is then routed through an authorisation gate rather than embedding policy checks in the storage or networking layers. The authorisation gate consists of cooperating components shown in Figure 4.6. TrustView maintains issuer keys and status-list material derived from trust-

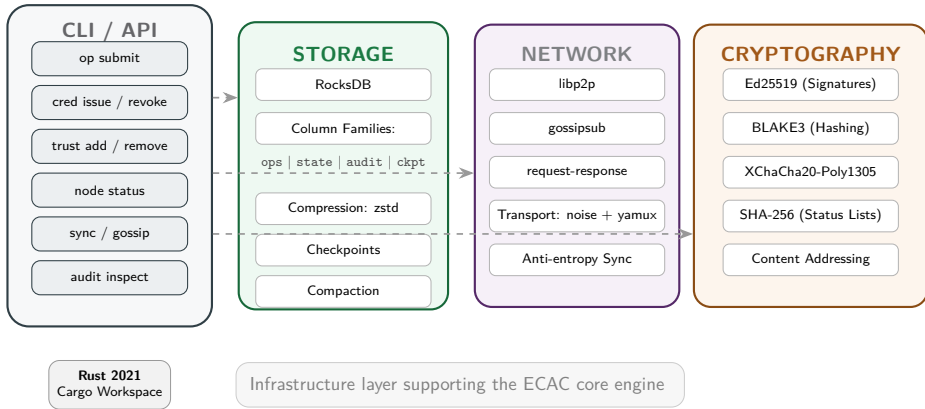


Figure 4.7: The implementation persists the append-only operation log, materialised CRDT state, audit records, and replay checkpoints in RocksDB column families. Nodes exchange operations using libp2p (gossipsub and request-response) with an anti-entropy sync protocol. Operations are signed with Ed25519 and content-addressed using BLAKE3; status-list chunks are verified with SHA-256. A CLI/API layer drives operation submission, credential and trust management, synchronisation, and audit inspection in a Rust 2021 Cargo workspace.

management operations. The policy evaluator consumes the current trust view together with the operation under replay and consults the current authorisation epoch intervals derived from grants and revocations. The gate produces a boolean decision per operation: on success the engine applies the state update; otherwise the operation is recorded as skipped.

Applied operations update the CRDT state layer, and the query surface reads from the materialised state rather than from the raw log (QUERY). In parallel, the engine records both applied and skipped decisions in a hash-linked audit stream stored alongside the other RocksDB column families, enabling post-hoc inspection from local persisted artefacts.

The repository exposes these capabilities through a CLI/API layer (Figure 4.7). The CLI binaries wrap the library engine to submit operations, manage credentials and trust anchors, trigger synchronisation and replay, and inspect audit and checkpoint state, while keeping the core usable as an embedded library.



---

## CHAPTER 5

---

# Conclusion

*“All models are wrong, but some are useful.”*

*George E. P. Box*

Two high-level questions guided this thesis; how to engineer cooperation over access rules in decentralised, multi-stakeholder settings, and how to obtain secure, consistent enforcement of those rules without surrendering local autonomy or system adaptability. The first question concerns how parties reach an agreement worth enforcing; the second concerns how that agreement survives translation into operational mechanisms across platforms and deployment conditions. This chapter answers the research questions in terms of what the artefacts demonstrate, and then discusses the assumptions, limitations, and methodological shortcuts that bound the meaning of those results.

**RQ-A:** *How can mutually acceptable access-control agreements be systematically established in decentralised systems where governance, policy, and enforcement are distributed across independent actors with conflicting interests?*

The negotiation prototype addresses the agreement problem by making policy choice concrete enough to analyse and critique. It represents stakeholders, evaluation criteria, and candidate rule sets as explicit entities, and implements a multi-criteria optimisation pipeline that computes individual and aggregate utilities. The practical outcome is a repeatable way to move from vague disagreement to documented recommendation: the tool takes structured input, produces an explicit trade-off surface (for example in the Pareto diagram), and leaves the final choice with the human stakeholders. In that sense, the work demonstrates, through artefacts, how access-rule choice can be supported by explicit models under stated assumptions that is transparent enough to critique and extend.

**RQ-B:** *What design principles enable security and consistency in federated access control without sacrificing stakeholder autonomy or system adaptability?*

The Kubernetes testbed and EQuack address the enforcement question in two different operating regimes.

In the cloud-native regime, the Kubernetes testbed demonstrates a verify-before-deploy path that fits into existing CI/CD practice. By translating cluster configurations into SMT con-

straints and checking invariants before deployment, it lets each operator keep local control over their cluster while offering stronger guarantees than best-practice documents or linters. The admission webhook and XACML decision point complement this with conservative runtime checks at API time, enforcing a deny-overrides discipline in the face of misconfiguration or drift.

EQuack extends the enforcement story to intermittently connected and offline settings. By combining a signed operation log, deterministic replay, and deny-wins semantics over grants and revocations, it demonstrates a design that achieves convergence under intermittent connectivity assumptions. Taken together, these results support a pragmatic answer to the second thesis question: it is possible to increase assurance and consistency without imposing a single central policy authority, but the strength and scope of the guarantees depend on explicit assumptions about administration, connectivity, and threat model.

## 5.1 Discussion on Assumptions and Limitations

Chapter 1 was explicit about several assumptions, and the meaning of the results depends on them. The assumption of honest-but-fallible administrators is central to both the negotiation framework and the Kubernetes pipeline. The software is built for people who try to configure systems sensibly but make mistakes, not for insiders who actively work against the system. If an administrator uses the tools to push a malicious configuration that nevertheless satisfies the stated invariants, the pipeline will sign off. This is not an accidental omission; formal verification can only ever prove conformance to an explicit specification, not to an unwritten notion of “doing the right thing.” Within that boundary, the contribution is real; beyond it, a different class of mechanism would be needed.

Eventual message delivery underpins EQuack. The convergence arguments rely on the idea that operations, including grants, revocations, and key updates, will eventually reach all honest nodes. In practice, this is a decent fit for many edge and data-space deployments where connectivity is intermittent but not permanently absent. Under a permanent partition, or for devices that never rejoin, the guarantees are simply not available. This is a fundamental trade-off: enforcing globally coherent rules across permanently separated worlds is impossible without additional out-of-band channels or a different availability model.

The thesis deliberately excludes physical compromise. None of the mechanisms described here survive a determined attacker with root access on a node or physical access to hardware. In that situation, keys can be extracted, logs can be forged, and local enforcement bypassed. This is one of the fundamental assumptions that cannot be engineered within the software layer; defending against such attackers would require hardware roots of trust, trusted execution environments, or similar mechanisms that are explicitly out of scope.

Cryptographic primitives are likewise taken to behave as advertised. EQuack relies on digital signatures and hash functions to provide integrity and authenticity for operations and audit chains, and the negotiation and verification tools rely on the basic integrity of their execution environment. If those primitives are broken or implemented incorrectly, the guarantees collapse. This is not unique to this work; it is the same baseline assumption that underlies most of modern secure systems design. What can be engineered is careful library choice, algorithm agility, and defensive implementation;

what cannot be guaranteed is that a future cryptanalytic breakthrough will not invalidate some of the underlying mathematics.

*Another assumption runs through both papers A and B; that someone can articulate the inputs these proposed systems require. The thesis treats policies, stakeholder preferences, and security invariants as though they can be stated explicitly enough to feed optimisation or verification engines. In practice, eliciting such inputs is messy. Stakeholders may be internally inconsistent, strategically vague, or genuinely unsure of what they want. Paper A acknowledges this implicitly by appealing to IDSA (International Data Spaces Association)-style data-usage policies as a structuring device, but IDSA semantics are themselves coarse-grained and leave wide latitude for interpretation.*

Paper B makes an analogous bet, *the verification engine works only when someone has first written a security invariant worth enforcing*. There is no guarantee that such a person exists, or that the invariant they produce faithfully captures operational intent rather than a convenient approximation of it.

These gaps are methodological shortcuts rather than fundamental barriers; future work could explore template libraries, interactive elicitation tools, or stepwise refinement interfaces that help stakeholders discover and articulate what they actually care about. But acknowledging the shortcuts matters. The research loop sketched in Chapter 1 was followed more cleanly in the EQUACK strand than in the negotiation and Kubernetes work, where available machinery (MCDM methods, SMT solvers, webhooks) bent the problem statements toward tractable forms. *Being explicit about where tool availability has shaped the questions is itself part of doing honest systems research.*

One thread I did not do justice to is *trust negotiation*. Chapters 1 and 2 mention it, but I only realised fairly late that it matches the shape of the real disagreements that motivated this work. Paper A is good at choosing between candidate rule sets once stakeholders can state their preferences, but many cross-organisation deadlocks happen earlier than that; the sticking point is often *what each side is willing to disclose, in what order, and under what conditions*. That is exactly what trust negotiation was built for, and it is a gap in the current artefacts rather than a solved problem. If this work is extended, trust negotiation is an obvious place to dig deeper.

Together, the contributions do not solve Data Spaces governance in any complete sense, but they do offer implementable pieces that map closely onto specific, recurring pain points in those environments.

## 5.2 Future Work

Looking forward, not all future directions are equally interesting. There is solid but unglamorous engineering work to be done in integrating the artefacts into a single pipeline; taking outputs from the negotiation framework, translating them into platform-specific policies, checking those with the verification pipeline, and pushing them into an offline-capable enforcement substrate such as EQUACK. Packaging that as a usable system, with sensible CI/CD hooks, policy translation tooling, and end-to-end tests across federated domains, would be valuable and is an obvious next step. The same is true for robustness and scale; incremental SMT checking for large clusters, negotiation schemes that cope with many stakeholders, richer key and confidentiality management in EQUACK, and better interfaces for specifying invariants and preferences will all make the ideas in this thesis more deployable.

The higher payoff lies elsewhere. One promising direction is cross-domain, end-to-end verifi-

cation; instead of reasoning about a single Kubernetes cluster or a single log, the goal would be to prove properties of access requests that cross several independently administered domains, each with its own policy language and trust assumptions. That requires ways to translate policies between domains without breaking security properties, and to produce machine-checkable explanations of why an end-to-end property holds or fails.

Another is access control for decentralised AI agents. As autonomous agents act on behalf of organisations and individuals in data spaces and edge deployments, they will need to negotiate for access, make decisions when cut off from their principals, and justify those decisions after the fact. The negotiation framework could be extended so that agents, not humans, are the primary negotiators. The verification ideas could be applied to agent policies rather than cluster configurations.

Both of these directions push the thesis themes into more ambitious territory and are more than just scaling up what already exists.

### 5.3 Closing Remarks

Stepping back, the thesis takes the stance that multi-stakeholder access control in distributed data ecosystems is not an inherently intractable socio-political tangle, nor is it something that can be solved by a single elegant model.

It is an engineering problem that sits at an uncomfortable intersection of organisational constraints, human preferences, and technical limits. The negotiation prototype offers a repeatable way to propose and justify candidate agreements; the verification pipeline shows how to catch and block common platform-level errors before they turn into incidents; and the EQUack core sketches a path to deterministic, auditable enforcement when connectivity and central coordination cannot be assumed.

None of these artefacts are definitive, and all of them sit on top of assumptions that need to be kept in view. But together they demonstrate how a particular combination of decision support, formal verification, and deterministic replay can be applied to multi-stakeholder access control, an integration not previously explored in this form. That reduction in uncertainty, increase in auditability, and explicit articulation of trade-offs is the main contribution of this licentiate, and the base on which the doctoral work intends to build.

---

## REFERENCES

---

- [1] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*, 3rd ed. CreateSpace Independent Publishing Platform, 2017.
- [2] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed. Addison-Wesley, 2011.
- [3] C. Troncoso, M. Isaakidis, G. Danezis, and H. Halpin, “Systematizing decentralization and privacy: Lessons from 15 years of research and deployments,” in *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, 2017, pp. 404–426.
- [4] J. Schneider, A. Blostein, B. Lee, S. Kent, I. Grosu, and N. Berendea, “A taxonomy for blockchain-based systems: Towards understanding the architecture and design space,” *IEEE Access*, vol. 7, pp. 181 537–181 557, 2019.
- [5] S. Cantor, J. Kemp, R. Philpott, and E. Maler, “Shibboleth architecture: Technical overview,” in *Working Draft*. Internet2, 2005.
- [6] A. Jøsang, R. Ismail, and C. Boyd, “A survey of trust and reputation systems for online service provision,” *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, 2007.
- [7] V. C. Hu, D. F. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, “Guide to attribute based access control (abac) definition and considerations,” 2014.
- [8] M. Janssen, P. Brous, E. Estevez, L. S. Barbosa, and T. Janowski, “Data governance: Organizing data for trustworthy artificial intelligence,” *Government Information Quarterly*, vol. 37, no. 3, p. 101493, 2020.
- [9] E. Curry, “The big data value chain: definitions, concepts, and theoretical approaches,” *New horizons for a data-driven economy: A roadmap for usage and exploitation of big data in Europe*, pp. 29–37, 2016.
- [10] B. Otto, “Designing data governance,” *IBM Center for The Business of Government*, 2011.
- [11] L. Ohno-Machado, S.-A. Sansone, G. Alter, I. Fore, J. Grethe, H. Xu, A. Gonzalez-Beltran, P. Rocca-Serra, A. E. Guraraj, E. Bell *et al.*, “Sharing patient health records based on collaboration between healthcare providers and patients,” in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. IEEE, 2011, pp. 830–837.

- [12] L. D. Stein, B. M. Knoppers, P. Campbell, G. Getz, and J. O. Korbel, "Implementing a multi-institutional data sharing infrastructure for clinical research: challenges, opportunities, and lessons learned," *JAMA*, vol. 311, no. 22, pp. 2267–2268, 2014.
- [13] M. Wieland and H. Schwarz, "Data governance requirements for the industrial internet of things," in *2021 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, 2021, pp. 1–8.
- [14] N. Purtova, "The law of everything: broad concept of personal data and future of eu data protection law," *Law, Innovation and Technology*, vol. 10, no. 1, pp. 40–81, 2018.
- [15] C. Azkan, L. Iggena, I. Gür, B. Otto, and F. Möller, "Managing data governance for data-driven innovation: challenges and opportunities in electric vehicle domain," *Data and Policy*, vol. 2, p. e17, 2020.
- [16] T. Lundqvist, A. De Blanche, and H. R. H. Andersson, "Thing-to-thing electricity micro payments using blockchain technology," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–6.
- [17] T. A. Pardo, J. R. Gil-Garcia, and L. F. Luna-Reyes, "Data ownership and the balancing of rights in smart cities," *Government Information Quarterly*, vol. 33, no. 3, pp. 529–534, 2016.
- [18] L. Edwards, "Privacy, security and data protection in smart cities: A critical eu law perspective," *European Data Protection Law Review*, vol. 2, p. 28, 2016.
- [19] S. A. De Chaves, C. B. Westphall, and F. R. Lamin, "Control cloud data using multi-clouds," in *2012 IEEE 4th International Conference on Cloud Computing Technology and Science*. IEEE, 2012, pp. 266–273.
- [20] S. Pearson and A. Charlesworth, "Accountability as a way forward for privacy protection in the cloud," in *Cloud Computing*. Springer, 2013, pp. 131–155.
- [21] A. C. Squicciarini, C. Caragea, and R. Balakavi, "Collective privacy management in social networks and online communities," in *Handbook of Mobile Data Privacy*. Springer, 2018, pp. 67–85.
- [22] J. M. Such and N. Criado, "Resolving multi-party privacy conflicts in social media," vol. 28, no. 7. IEEE, 2016, pp. 1851–1863.
- [23] R. S. Sandhu and P. Samarati, "Access control: principle and practice," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, 1994.
- [24] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.
- [25] B. W. Lampson, "Protection," *ACM SIGOPS Operating Systems Review*, vol. 8, no. 1, pp. 18–24, 1974.

- [26] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, 1976.
- [27] M. Bishop, *Computer security: art and science*. Addison-Wesley Professional, 2003.
- [28] D. E. Bell and L. J. LaPadula, "Secure computer systems: Mathematical foundations," *MITRE Technical Report*, vol. 2547, 1973.
- [29] R. S. Sandhu, "Lattice-based access control models," *Computer*, vol. 26, no. 11, pp. 9–19, 1993.
- [30] K. J. Biba, "Integrity considerations for secure computer systems," *MITRE Technical Report*, vol. MTR-3153, 1977.
- [31] D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.
- [32] N. Li, B. N. Groszof, and J. Feigenbaum, "Delegation logic: A logic-based approach to distributed authorization," vol. 6, no. 1. ACM, 2003, pp. 128–171.
- [33] K. E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu, "Trust negotiation for ad hoc coalitions," in *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM, 2001, pp. 140–149.
- [34] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.
- [35] Kubernetes, "Using rbac authorization," <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>, 2023, accessed: 2024-12-14.
- [36] E. Bertino, P. A. Bonatti, and E. Ferrari, "Access control for collaborative systems," in *Proceedings of the 9th ACM symposium on Access control models and technologies*. ACM, 2004, pp. 1–10.
- [37] M. Shehab, E. Bertino, and A. Ghafoor, "Secure data exchange between federated cloud providers," in *2008 IEEE International Conference on Services Computing*, vol. 2. IEEE, 2008, pp. 437–444.
- [38] A. Kapadia, J. Al-Muhtadi, R. H. Campbell, and M. D. Mickunas, "Attribute-based secure collaboration in dynamic federations," vol. 7, no. 5. Elsevier, 2009, pp. 925–943.
- [39] E. Yuan and J. Tong, "Attributed based access control (abac) for web services," in *Proceedings of the IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005, pp. 561–569.
- [40] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, 2015.

- [41] N. Li, J. C. Mitchell, and W. H. Winsborough, "Practical attribute-based access control for web services," in *22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE, 2006, pp. 169–180.
- [42] G. Hughes and T. Bultan, "Automated verification of access control policies," vol. 55, no. 6. Elsevier, 2011, pp. 1333–1347.
- [43] K. Fiesler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 196–205.
- [44] A. Lazouski, F. Martinelli, and P. Mori, "Usage control models: Survey," *Computer Science Review*, vol. 4, no. 2, pp. 81–99, 2010.
- [45] A. Pretschner, M. Hilty, and D. Basin, "Representation-independent data usage control," in *Data and Applications Security XX*. Springer, 2006, pp. 122–140.
- [46] M. Hilty, D. Basin, and A. Pretschner, "Distributed usage control," vol. 49, no. 9. ACM, 2006, pp. 39–44.
- [47] P. Kumari, A. Pretschner, J. Peschla, and J.-M. Kuhn, "Models for adaptive information flow control," *2012 Tenth Annual International Conference on Privacy, Security and Trust*, pp. 130–137, 2012.
- [48] J. M. Such, N. Criado, and A. G.-F. Navarro, "Policy negotiation for multi-party access control in social networks," in *2012 Third International Conference on Emerging Security Technologies*. IEEE, 2012, pp. 43–50.
- [49] R. Sinha, S. Palit, and P. Kumaraguru, "Kubernetes security: An analysis of cves and their severity," in *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020, pp. 1–9.
- [50] P. Raj and S. Thiagarajan, "Security analysis of kubernetes: A review," vol. 18, no. 6, 2020.
- [51] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu, "Negotiating disclosure of sensitive credentials," in *Proceedings of the 2nd Conference on Privacy Enhancing Technologies*, 2002, pp. 1–12.
- [52] T. Yu, M. Winslett, and K. E. Seamons, "Automated trust negotiation," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. ACM, 2003, pp. 88–91.
- [53] N. Li and J. C. Mitchell, "A practical trust management system," vol. 36, no. 2. IEEE, 2003, pp. 88–90.
- [54] T. Yu, M. Winslett, and K. E. Seamons, "Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation," vol. 6, no. 1. ACM, 2003, pp. 1–42.

- [55] W. H. Winsborough, K. E. Seamons, and V. E. Jones, "Protecting sensitive policies during trust negotiation," in *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*. IEEE, 2002, pp. 80–87.
- [56] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," vol. 28, no. 10. ACM, 1985, pp. 1030–1044.
- [57] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE, 1996, pp. 164–173.
- [58] P. A. Bonatti and P. Samarati, "A uniform framework for regulating service access and information release on the web," *Journal of Computer Security*, vol. 10, no. 3, pp. 241–271, 2002.
- [59] K. E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu, "Requirements for policy languages for trust negotiation," in *Proceedings of 3rd International Workshop on Policies for Distributed Systems and Networks*. IEEE, 2002, pp. 68–79.
- [60] A. J. Lee, M. Winslett, and K. J. Perano, "Trust management and trust negotiation," vol. 40, no. 2. ACM, 2008, pp. 1–48.
- [61] G. Bruns, D. S. Dantas, and M. Huth, "Combining attribute based and role based access control," in *Proceedings of the 12th ACM symposium on Access control models and technologies*. ACM, 2007, pp. 45–55.
- [62] H.-X. Lin, B.-X. Fang, T. Huang, and Q.-W. Wei, "Secure distributed cooperative firewall algorithm," *Journal of Computer Science and Technology*, vol. 22, no. 4, pp. 571–581, 2007.
- [63] OASIS, "extensible access control markup language (xacml) version 3.0," OASIS Standard, 2013, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html>.
- [64] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, and A. Trombetta, "Design and implementation of xacml-based access control in a distributed multimedia system," vol. 5, no. 4. ACM, 2009, pp. 1–28.
- [65] P. A. Bonatti, C. Duma, N. Fuchs, W. Nejdl, D. Olmedilla, J. Peer, and N. Shahmehri, "Semantic web policies—a discussion of requirements and research issues." Springer, 2006, pp. 712–724.
- [66] C.-L. Hwang and K. Yoon, *Multiple attribute decision making: methods and applications*. Springer, 1981.
- [67] K. Miettinen, *Nonlinear multiobjective optimization*. Springer, 1999.
- [68] A. Pretschner and S. Wenzel, "Multi-objective decision support for data sharing agreements," in *Proceedings of the 16th ACM symposium on Access control models and technologies*. ACM, 2011, pp. 43–52.

- [69] I. Saenko and I. Kotenko, "Choice of access control model and method at development of software applications," in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EICoN Rus)*. IEEE, 2020, pp. 1716–1721.
- [70] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001.
- [71] F. Cuppens, N. Cuppens-Bouahia, and A. Ghorbel, "Organization based access control," in *2008 IEEE International Conference on Pervasive Computing and Communications*. IEEE, 2008, pp. 120–129.
- [72] D. F. Ferraiolo, V. Atluri, and S. Gavrila, "Scalable and flexible enterprise access control," pp. 1–2, 2009.
- [73] E. Yuan and J. Tong, "Attributed based access control (abac) for web services," vol. 43, no. 7. IEEE, 2010, pp. 85–88.
- [74] Q. Zhang, L. Cheng, and R. Boutaba, "Security-aware virtual machine allocation in the cloud: A game theoretic approach," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*. IEEE, 2011, pp. 556–563.
- [75] D. Zerkle and K. Levitt, "Netfilter: Requirements and design of a firewall toolkit," vol. 26, no. 6. Elsevier, 1997, pp. 981–994.
- [76] A. Mayer, A. Wool, and E. Ziskind, "Reachability analysis for access control policy verification," in *2006 IEEE Symposium on Security and Privacy*. IEEE, 2006, pp. 133–147.
- [77] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: Up and running*, 3rd ed. O'Reilly Media, 2022.
- [78] K. Hightower, B. Burns, and J. Beda, *Kubernetes: The definitive guide to hands-on deployment, configuration, and application development*. O'Reilly Media, 2017.
- [79] Kubernetes, "Authenticating," <https://kubernetes.io/docs/reference/access-authn-authz/authentication/>, 2023, accessed: 2024-12-14.
- [80] —, "Using admission controllers," <https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/>, 2023, accessed: 2024-12-14.
- [81] N. Sturdevant and A. Baez, "Securing the configuration of kubernetes cluster using admission control," *Journal of Cybersecurity and Privacy*, vol. 1, no. 1, pp. 96–114, 2021.
- [82] Kubernetes, "Network policies," <https://kubernetes.io/docs/concepts/services-networking/network-policies/>, 2023, accessed: 2024-12-14.
- [83] Project Calico, "Calico network policy," <https://docs.projectcalico.org/>, 2023, accessed: 2024-12-14.

- [84] F. Yang, W. Wang, J. Li, and Y. Zhou, “Detecting rbac privilege escalation risks in kubernetes,” in *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2022, pp. 81–90.
- [85] M. Hausenblas and L. Rice, “Kubernetes security,” 2018.
- [86] L. Rice, “Container security: Fundamental technology concepts that protect containerized applications.” O’Reilly Media, 2020.
- [87] K. R. Patil, T. Maillart, A. L. Timan, and K. Levitt, “Understanding security mistakes developers make: Qualitative analysis from build it, break it, fix it,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 109–126.
- [88] S. Garetson, “Kubesecc: Security risk analysis for kubernetes resources,” <https://kubesecc.io/>, 2019, accessed: 2024-12-14.
- [89] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Model checking*, 2nd ed. MIT Press, 2018.
- [90] A. Biere, M. Heule, H. Van Maaren, and T. Walsh, *Handbook of satisfiability*. IOS press, 2009, vol. 185.
- [91] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.
- [92] I. Malavolta, R. Verdecchia, B. Filipovic, M. Bruntink, and P. Lago, “Continuous software compliance,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 153–162.
- [93] J. Chen, J. Huang, V. Sharma, and X. S. Wang, “Detecting violations of access control and information flow policies in data processing systems,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1671–1688.
- [94] T. Nelson, C. Barratt, D. J. Dougherty, K. Fislser, and S. Krishnamurthi, “The margrave tool for firewall analysis,” in *USENIX Large Installation System Administration Conference (LISA)*, 2010, pp. 1–8.
- [95] R. Shambaugh, A. Weiss, and N. Foster, “Detecting and validating policy-based access control in cloud native applications,” in *Proceedings of the 12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud ’20)*. USENIX, 2020.
- [96] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
- [97] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, “Security, privacy and trust in internet of things: The road ahead,” *Computer Networks*, vol. 76, pp. 146–164, 2015.

- [98] P. Kumar, A. Braeken, A. Gurtov, J. Iinatti, and P. H. Ha, “A survey on access control in the age of internet of things,” *IEEE Access*, vol. 7, pp. 4988–5022, 2019.
- [99] V. C. Hu, D. Ferraiolo, R. Kuhn, R. N. Kacker, and Y. Lei, “Collaborative access control in openstack,” in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2015, pp. 125–132.
- [100] A. Nehme, G. Nain, Y. Le Traon, and P. W. Fong, “Context-aware access control for cyber-physical systems,” in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2014, pp. 329–338.
- [101] M. Sutton, D. Greene, and P. Hosom, “Security challenges in the industrial internet of things,” *IEEE Security & Privacy*, vol. 13, no. 5, pp. 70–73, 2015.
- [102] J. Sametinger, J. Rozenblit, R. Lysecky, and P. Ott, “Security challenges for medical devices,” *Communications of the ACM*, vol. 58, no. 4, pp. 74–82, 2015.
- [103] R. Neisse, G. Steri, and I. Nai-Fovino, “Enforcement of security policy rules for the internet of things,” in *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2014, pp. 165–172.
- [104] S. Cirani, L. Davoli, G. Ferrari, R. Leone, P. Medagliani, M. Picone, and L. Veltri, “Internet of things: Architectures, protocols and standards,” 2014.
- [105] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, “Conflict-free replicated data types,” in *Symposium on Self-Stabilizing Systems*. Springer, 2011, pp. 386–400.
- [106] —, “A comprehensive study of convergent and commutative replicated data types,” *Technical Report 7506, INRIA*, 2011.
- [107] M. Kleppmann, “Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems,” 2017.
- [108] P. S. Almeida, A. Shoker, and C. Baquero, “Delta state replicated data types,” vol. 111. Elsevier, 2018, pp. 162–173.
- [109] A. Bieniussa, M. Zawirski, N. Preguiça, M. Shapiro, C. Baquero, V. Balesgas, and S. Duarte, “A brief guide to the usage of crdts,” in *19th Brazilian Symposium on Multimedia and the Web*. ACM, 2013, pp. 13–16.
- [110] M. Kleppmann and A. R. Beresford, “Making crdts byzantine fault tolerant,” in *Proceedings of the 9th ACM International Systems and Storage Conference*. ACM, 2016, pp. 1–7.
- [111] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205–220, 2007.

- 
- [112] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing update conflicts in bayou, a weakly connected replicated storage system," *ACM SIGOPS Operating Systems Review*, vol. 29, no. 5, pp. 172–182, 1995.
- [113] S. S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone, "Logical physical clocks and consistent snapshots in globally distributed databases," *Technical Report*, 2014.
- [114] M. Zawirski, A. Bieniusa, N. Preguiça, S. Duarte, V. Balegas, C. Baquero, and M. Shapiro, "Replicated abstract data types: Building blocks for collaborative applications," vol. 71, no. 3. Elsevier, 2011, pp. 354–368.
- [115] R. Anderson, "Security engineering: A guide to building dependable distributed systems," 2008.
- [116] M. Kleppmann and A. R. Beresford, "A conflict-free replicated json datatype," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2733–2746, 2017.
- [117] A. R. Yumerefendi and J. S. Chase, "Strong accountability for network storage," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, 2007, pp. 77–92.
- [118] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," vol. 41, no. 6. ACM, 2007, pp. 175–188.
- [119] D. Terry, "Replicated data consistency explained through baseball," *Communications of the ACM*, vol. 56, no. 12, pp. 82–89, 2013.





Department of SRT  
Division of EISLAB

---

ISSN 1402-1757  
ISBN 978-91-8048-966-9 (print)  
ISBN 978-91-8048-967-6 (pdf)

Luleå University of Technology 2026